Architecture, qualité et lignes de produits : établissement d'une architecture de référence dans un domaine applicatif

Grégory Zubowicz

le cnam

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

PARIS

MEMOIRE

Présenté en vue d'obtenir

le DIPLOME D'INGENIEUR CNAM

SPECIALITE : Informatique

OPTION : Réseaux, systèmes et multimédia

par

Grégory Zubowicz

Architecture, qualité et lignes de produits :

Etablissement d'une architecture de référence dans un domaine applicatif

Soutenu le 15 décembre 2015

JURY

PRESIDENT : Yann Pollet Professeur CNAM, membre du Cedric équipe Vespa

MEMBRES :

Catherine Dubois	Professeur ENSIIE
Laurent Denizot	CEO Egidium Technologies
Nicolas Trèves	Professeur CNAM, membre du Cedric équipe SEMpIA
Nicole Levy	Professeur CNAM, membre du Cedric équipe CPR
Véronique Legrand	Professeur INSA-Lyon, membre du Laboratoire CITI

Table des matières

Remerciements

Ce mémoire est l'aboutissement d'un long chemin, commencé avant même mon parcours au CNAM. Je dois évidemment remercier ceux qui m'ont permis de suivre ce parcours.

Ma famille, pour son soutien. Je remercie particulièrement mon père adoptif, Jacques, à qui je pense tous les jours, et dont je m'efforce de suivre l'exemple. Ma mère, pour son exemple de courage.

Patricia, Philippe et Romain Grosset, grâce à qui j'ai pu rebondir et entamer mes études en informatique, trouver un emploi, m'installer et renouer avec celle qui est aujourd'hui ma femme. Sans eux je ne serais certainement pas à la place où je suis.

Ma femme pour ne m'avoir jamais pressé ni exprimé d'agacement ou d'impatience malgré ces cinq années au CNAM à travailler le soir et les week-end. Son absence de doutes quant à ma réussite. Pour toujours me donner envie d'en faire plus.

Mme Mast, CNAM Versailles, pour sa disponibilité et sa bonne humeur.

Mme Levy, pour avoir eu suffisamment confiance en moi pour me confier ce sujet et pour m'avoir guidé tout au long de ce mémoire.

Liste des abréviations

API	Application Programming Interface.
DeltaJ	Delta Oriented Programming for Java.
DOP	Delta Oriented Prorgramming.
DSL	Domain Specific Language.
DSML	Domain-Specific Modeling Language.
EMF	Eclipse Modeling Framework.
GEF	Graphical Editing Framework.
GMF	Graphical Modeling Framework.
HITS	Hyper-link Induced Topic Search
HTTPS	HyperText Transfer Protocol Secure.
IHM	Interface Homme-Machine.
MDA	Model Driven Architecture.
MDE	Model Driven Engineering.
MOFM2T	MOF Models to Text Transformation Language.
NAT/PAT	Network Adress Translation/Port Adress Translation.
OCL	Object Constraint Language
OMG	Object Management Group.
P2P	Pear To Pear.
P2P privé	Pear to Pear privé.
QR codes	Quich Response code
QVT	Query,View,Transformation.
REST	Representational State Transfert.
SPL	Software product Lines.
UPnP	Universal Plug and Play
Wi-Fi	Wireless Fidelity.

Glossaire

Terme	Description	Source
Acceleo	Acceleo est un générateur de code source de la fondation Eclipse permettant de mettre en œuvre l'approche MDA (Model driven architecture) pour réaliser des applications à partir de modèles basés sur EMF. Il s'agit d'une implémentation de la norme de l'Object Management Group (OMG) pour les transformations de modèle vers texte (M2T) Model to Text.	https://fr.wikipedia.org/wiki/Acceleo
Adresse IP	Une adresse IP (avec IP pour *Internet Protocol*) est un numéro d'identification qui est attribué de façon permanente ou provisoire à chaque appareil connecté à un réseau informatique utilisant l'Internet Protocol. L'adresse IP est à la base du système d'acheminement (le routage) des messages sur Internet.	https://fr.wikipedia.org/wiki/Adresse_IP
API	Une interface de programmation (souvent désignée par le terme API pour Application Programming Interface) est un ensemble normalisé de classes, de méthodes ou de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels. Elle est offerte par une bibliothèque logicielle ou un service web, le plus souvent accompagnée d'une description qui spécifie comment des programmes consommateurs peuvent se servir des fonctionnalités du programme fournisseur.	https://fr.wikipedia.org/wiki/Interface_de_programmation
DeltaJ	Delta Oriented Programming for Java est une approche modulaire d'implémentation des SPL fournissant un accès aux aspects orienté objet du langage Java. C'est un language de programmation introduisant les principes du DOP dans Java. Basé sur le framework XText, il est disponible en tant que plug-in Eclipse.	https://www.tu-braunschweig.de/isf/research/deltas/, http://www.researchgate.net/profile/Sandro_Schulze/publication/266661748_DeltaJ_1.5_delta-oriented_programming_for_Java_1.5/links/543e1ed70cf240f04d10d324.pdf, http://www.researchgate.net/publication/266661748_DeltaJ_1.5_delta-oriented_programming_for_Java_1.5

DOP	Delta Oriented Prorgramming. Approche modulaire d'implémentation des SPL. Les caractéristiques d'un produit logiciel sont mappées à des « Delta modules » contenant le code source de la caractéristique. Les produits sont ensuite définis par une sélection de caractèristiques	
DSL	Un langage dédié (Domain specific language) est un langage de programmation dont les spécifications sont conçues pour répondre aux contraintes d'un domaine d'application précis. Il s'oppose conceptuellement aux langages de programmation classiques (ou généralistes) comme le Java ou le C, qui tendent à traiter un ensemble de domaines.	https://fr.wikipedia.org/wiki/Langage_dédié
DSML	Domain-Specific Modeling Language. Méthode d'ingienerie logicielle de design et développement de systèmes. Il invite à l'utilisation systèmatique de DSL (domain specific language) pour représentater les différents aspects d'un système.	https://en.wikipedia.org/wiki/Domain-specific_modeling
EMF	Le projet Eclipse Modeling Framework (EMF) est un framework de modélisation, une infrastructure de génération de code et des applications basées sur des modèles de données structurées. Partant d'une spécification décrite généralement sous la forme d'un modèle en XMI, EMF fournit des outils permettant de produire des classes Java représentant le modèle avec un ensemble de classes pour adapter les éléments du modèle afin de pouvoir les visualiser, les éditer avec un système de commandes et les manipuler dans un éditeur.	https://fr.wikipedia.org/wiki/Eclipse_Modeling_Framework
Framework	En programmation informatique, un *framework* ou *structure logicielle* est un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture).	https://fr.wikipedia.org/wiki/Framework
GEF	Framework développé pour la plateform Eclipse. Utilisée pour créer des éditeurs graphiques pour divers types de schémas (réseaux électriques, graphes et arbres). Habituellement utilisé comme composante de GMF, qui combine EMF et GEF pour permettre de générer le code d'un modèle de donnée et d'un éditeur de diagrammes.	https://en.wikipedia.org/wiki/Graphical_Editing_Framework
GMF	Graphical Modeling Framework (GMF) est un framework de l'environnement de travail Eclipse. Il fournit une infrastructure permettant l'exécution d'éditeurs graphiques basés sur les frameworks EMF et GEF.	https://fr.wikipedia.org/wiki/Graphical_Modeling_Framework

HITS	L'algorithme HITS, pour Hyperlink-Induced Topic Search, est un algorithme qui permet de mesurer l'autorité d'une page Web par rapport à d'autres. Il a été créé en 1999 par Jon Kleinberg. Il est parfois considéré comme précurseur de l'algorithme PageRank qui, comme HITS, a pour but, sur la base d'un graphe, d'assigner un score à celles-ci de façon à identifier les page ayant le plus d'"importance"	https://fr.wikipedia.org/wiki/Algorithme_HITS
HTTPS	L'HyperText Transfer Protocol Secure, plus connu sous l'abréviation https — littéralement « protocole de transfert hypertexte sécurisé » — est la combinaison du http avec une couche de chiffrement comme SSL ou TLS.	https://fr.wikipedia.org/wiki/HyperText_Transfer_Protocol_Secure
Hotspot	Une borne Wi-Fi, ou un point Wi-Fi, ou un hotspot, est un matériel qui donne accès à un réseau sans fil Wi-Fi permettant aux utilisateurs de terminaux mobiles de se connecter à Internet. L'accès ainsi fourni peut être gratuit ou payant pour l'utilisateur.	https://fr.wikipedia.org/wiki/Borne_Wi-Fi
MDA	Model Driven Architecture. L'architecture dirigée par les modèles ou MDA (pour l'anglais Model Driven Architecture) est une démarche de réalisation de logiciels, proposée et soutenue par l'OMG. C'est une variante particulière de l'ingénierie dirigée par les modèles (IDM, ou MDE pour l'anglais Model Driven Engineering).	https://fr.wikipedia.org/wiki/Model_driven_architecture.
MDE	Model driven enginiering ou ingénierie dirigée par les modèles (IDM, en français) est une pratique d'ingénierie des systèmes utilisant les capacités des technologies informatiques pour décrire au travers de modèles, concepts, et langages, à la fois le problème posé (besoin) et sa solution.	https://fr.wikipedia.org/wiki/Ing%C3%A9nierie_dirig%C3%A9e_par_les_mod%C3%A8les
MOFM2T	MOF Models to Text Transformation Language (ou MOF2Text) est une spécification de l'Object Management Group (OMG) dans le cadre du développement à base de modèles. La spécification s'attache à décrire comment transformer un modèle en texte, par exemple du code source ou de la documentation. MOFM2T est l'une des parties du Model Driven Architecture (MDA) de l'OMG et ainsi cette spécification réutilise de nombreux concepts de l'architecture de méta-modélisation de l'OMG. MOFM2T est utilisé pour exprimer des transformations de modèle vers texte.	https://fr.wikipedia.org/wiki/MOF_Models_to_Text_Transformation_Language

NAT/PAT	Network Adress Translation/Port Adress Translation. En réseau informatique, on dit qu'un routeur fait du Network Address Translation (NAT) (« translation d'adresse réseau »1) lorsqu'il fait correspondre les adresses IP internes non uniques et souvent non routables d'un intranet à un ensemble d'adresses externes uniques et routables. Ce mécanisme permet notamment de faire correspondre une seule adresse externe publique visible sur Internet à toutes les adresses d'un réseau privé, et pallie ainsi l'épuisement des adresses IPv4.	https://fr.wikipedia.org/wiki/Network_address_translation
OMG	L'Object Management Group est une association américaine à but non lucratif créée en 1989 dont l'objectif est de standardiser et promouvoir le modèle objet sous toutes ses formes. L'OMG est notamment à la base des standards UML (Unified Modelling Language) et IDL (Interface Definition Language).	https://fr.wikipedia.org/wiki/Object_Management_Group
P2P privé	Les réseaux de type P2P privé (Private P2P en anglais) sont des réseaux qui permettent seulement à quelques ordinateurs se faisant mutuellement confiance d'effectuer du partage de fichiers en pair à pair. Ceci peut être réalisé en utilisant un serveur central pour authentifier des clients, dans ce cas la fonctionnalité est semblable à un serveur FTP privé (mode client-serveur), mais les fichiers sont transférés directement entre les clients (mode pair à pair). En alternative, les utilisateurs peuvent échanger avec leurs amis un mot de passe ou une clef publique de cryptographie asymétrique pour former un réseau décentralisé.	https://fr.wikipedia.org/wiki/P2P_priv%C3%A9
QR codes	Quick Response code, type de code-barres en deux dimensions (ou code matriciel datamatrix) constitué de modules noirs disposés dans un carré à fond blanc. L'agencement de ces points définit l'information que contient le code.	https://fr.wikipedia.org/wiki/Code_QR
QVT	Query,View,Transformation est un standard défini par l'OMG utilisé pour exprimer des transformations de modèle vers modèle (M2M).	
REST	REST (representational state transfer) est un style d'architecture pour les systèmes hypermédia distribués, créé par Roy Fielding en 2000 dans le chapitre 5 de sa thèse de doctorat1. Il trouve notamment des applications dans le World Wide Web.	https://fr.wikipedia.org/wiki/Representational_State_Transfer
SPL	L'ingénierie des lignes de produits logiciels (Software product Lines, SPL): modèle de développement dont l'objectif est l'obtention d'un ensemble de produits logiciels répondant aux besoins spécifiques d'un segment de marché particulier.	

UPnP	Universal Plug and Play, est un protocole réseau promulgué par l'UPnP Forum. Le but de l'UPnP est de permettre à des périphériques de se connecter aisément et de simplifier la mise en œuvre de réseaux à la maison (partages de fichiers, communications, divertissements) ou dans les entreprises.	https://fr.wikipedia.org/wiki/Universal_Plug_and_Play

Introduction

A la fin des années 70 les termes « crise du logiciel » étaient employés. Ils faisaient référence à une baisse significative de la qualité des logiciels dont la venue coïncide avec l'augmentation de la puissance de calcul des ordinateurs permettant de réaliser des logiciels beaucoup plus complexes qu'auparavant.

Les premières tentatives de création de logiciels de grande ampleur ont vite montré les limites d'un travail informel d'ingénieurs logiciel : les produits réalisés ne sont pas terminés dans les temps, coûtent plus cher que prévu et ne sont pas fiables. Des études se sont penchées sur la recherche de méthodes de travail adaptées à la complexité inhérente aux logiciels modernes et ont donné naissance au génie logiciel, tendant à rationaliser la production du logiciel et son suivi. (1)

Aujourd'hui, les objectifs restent les mêmes mais ne concernent plus seulement le développement d'un seul logiciel à la fois. Le génie logiciel aborde les problèmes liés à la conception et au développement de lignes ou de familles de produits, à la recherche d'économies de temps et de coûts par la planification de la réutilisation de composants logiciels. Bien que la notion de ligne ou famille de produit ait été abordée assez tôt (David Parnas, l'abordait déjà en 1976 dans On the design and development of program families) ce n'est seulement que depuis deux décennies environ que ce paradigme est mis en avant, notamment en Europe, depuis la création de communautés comme ESAPS (2001) puis Café (2).

L'ingénierie des lignes de produits logiciels (Software Product Lines, SPL) est un modèle de développement dont l'objectif est l'obtention d'un ensemble de produits logiciels répondant aux besoins spécifiques d'un segment de marché particulier. Ces logiciels, qui répondent à des besoins approchants, ont des caractéristiques communes. L'ingénierie des SPL va donc chercher à favoriser la réutilisation.

Les produits d'une SPL sont ainsi développés à partir d'éléments communs qui forment le noyau de leurs architectures. Ils possèdent néanmoins des éléments qui n'apparaissent pas dans l'ensemble des produits de la SPL, justifiés par des besoins spécifiques à chaque membre de la ligne de produits. La notion de variabilité désigne ces caractéristiques qui différencient les produits d'une SPL. Sa gestion est un objectif essentiel de l'ingénierie des SPL. Les deux phases principales de l'ingénierie des SPL sont l'ingénierie de domaine et la « dérivation » de produit (ou ingénierie d'application). La première consiste à collecter,

organiser et capitaliser les expériences faites dans la conception de systèmes sous la forme d'éléments réutilisables dans l'élaboration de nouveaux systèmes. Cette phase consiste en une analyse du domaine dont le but est d'identifier les points communs et les points de variabilités entre les membres de la SPL et la création d'une architecture définie en termes de composants. La seconde phase concerne la composition d'un nouveau produit particulier. Il s'agit d'opérer certains choix parmi les éléments de variation disponibles en fonction des fonctionnalités attendues pour la future application. Le problème ici est de gérer les contraintes qu'imposent certains choix : soit ils en excluent soit ils en impliquent d'autres plus ou moins fortement.

Ces deux aspects des SPL ont été abondamment traités ces dernières années et diverses approches ont été proposées. Certaines proposent de manipuler les lignes de produits en recherchant dans plusieurs produits les points communs et les points de variations en analysant le code lui-même (feature location, recherche de caractéristiques, entre autres). Ces techniques donnent des résultats. Toutefois plusieurs obstacles leur font défaut : il a été observé que lorsque la proportion de fonctionnalités communes partagées par deux produits logiciels est d'environ 60 à 75 %, leurs implémentations ne partagent qu'environ 30% de code en commun (Kentaro Yoshimura, Dharmalingam Ganesan, et DirkMuthig. Assessing merge potential of existing engine control systems into a product line, 2006). De plus, il n'est plus rare de trouver maintenant des produits logiciels de centaines de milliers de lignes de code. L'identification des caractéristiques communes à partir du code n'est donc pas aisée. Ce ne sera pas la voie empruntée au cours de ce mémoire. Celui-ci propose l'adoption d'une approche pour l'élaboration d'une ligne de produits logiciels en travaillant, non pas au niveau code, mais au niveau architecture logicielle.

L'architecture logicielle décrit d'une manière symbolique et schématique les différents éléments d'un ou de plusieurs systèmes informatiques, leurs interrelations et leurs interactions (3). Elle constitue le plus gros livrable d'un processus logiciel après le produit (le logiciel lui-même). En effet, la phase de conception devrait consommer autour de 40 % de l'effort total de développement et devrait être supérieure ou égale, en effort, à la phase de codage. Les deux objectifs principaux de toute architecture logicielle sont la réduction des coûts et l'augmentation de la qualité du logiciel (3). Ce sont ces deux objectifs qui seront poursuivis au long de ce mémoire :

- La réduction des coûts sera principalement réalisée par l'élaboration de l'architecture d'une ligne de produits permettant la réutilisation de composants logiciels et la diminution du temps

de développement. D'un point de vue architectural, afin d'obtenir une ligne de produits, il faut commencer par identifier le noyau commun à l'ensemble des produits qui formeront la SPL et les « variants » que chacun de ses membres apporte. C'est ce que N. Levy *et al.* (4) appellent l'architecture de référence de la ligne de produits. Pour l'obtenir, deux méthodes se retrouvent dans l'ingénierie des lignes de produits : la première consiste pour l'architecte logiciel à créer l'architecture de référence à partir des connaissances du domaine. C'est une démarche de développement ex nihilo. La seconde, que nous suivrons, prend appui sur des produits déjà existants dont les éléments communs sont factorisés. Les éléments optionnels sont déterminés par la même occasion. Une fois ces deux classes de composants (noyau et optionnel) établis, l'ensemble des produits de la ligne de produits peut être déduite par dérivation.

- La qualité sera prise en compte en suivant les recommandations de la norme ISO 25010 conformément à une méthode proposée par N. Levy *et al.* (5).

Ce stage fut effectué au sein du laboratoire CEDRIC, équipe CPR (Conception et Programmation Raisonnées) dans le but d'appliquer les recherches effectuées en matière de familles de produits et de qualité logicielle. J'ai eu l'opportunité de les appliquer dans le cadre d'une entreprise personnelle visant à la production d'applications Android de partage de contenus décentralisés, avec des contraintes de ressources serveurs limitées. Ces applications seront décrites dans le premier chapitre de ce mémoire (I)

Le problème central de l'ingénierie des familles de produits est la construction de l'architecture de référence, à partir de laquelle les produits peuvent être dérivés. Pour ce faire, N. Levy *et al.* (4) proposent d'analyser la structure d'applications existantes et la manière dont leurs composants communiquent. Ce sera l'objet du chapitre II de ce mémoire. La démarche suivie sera celle que suivrait une entité ayant déjà un catalogue d'architectures de produits logiciels. Le but est de permettre de réaliser le premier objectif de l'élaboration d'une architecture, mentionné plus haut (réduction des coûts).

Le second objectif, garantir la qualité, sera rempli en suivant les recommandations prescrites par la norme ISO 25010 tel que le propose N. Levy *et al.* 2014 (5) en se concentrant sur des exigences de qualité pour garantir une architecture de référence de qualité dont l'ensemble des nouveaux membres de la famille de produit hériteront. Ceci sera exposé au chapitre III. Nous verrons que la méthode proposée, en plus de garantir la qualité, s'intègre naturellement dans le cadre d'un travail préparatoire à la dérivation de produits telle que recommandée par P. O'Leary *et al*. (6).

Enfin, une méthode de gestion de la variabilité et de dérivation parmi celles existantes sera proposée (chapitre IV). Cet aspect de l'ingénierie des lignes de produits n'a pas été traité dans les recherches qu'il m'était demandé d'appliquer dans le cadre de ce mémoire. La proposition résulte de recherches documentaires effectuées indépendamment.

Ce mémoire est présenté dans l'objectif d'obtenir le diplôme d'ingénieur en informatique de l'école d'ingénieurs du Conservatoire National des Arts et Métiers, dans la poursuite du cursus IRSM (informatique, parcours réseaux, systèmes et multimédia) parcours systèmes sûrs embarqués et mobiles. Il reflète un des aspects principaux du travail d'ingénieur et mobilise certaines des qualités qui lui sont nécessaires : il lui faut être capable de rechercher les connaissances requises à la réalisation d'un projet, de se former dans un domaine qu'il ne connaît pas encore et d'appliquer les recommandations de la doctrine. Cette dernière bénéficiant du retour d'information quant à l'application concrète de ses recherches. Dans cet esprit, les difficultés liées à l'approche proposée seront donc synthétisées en conclusion (chapitre V).

I Présentation des applications

Cette première partie est une présentation des applications sur lesquelles reposera l'ensemble de ce travail. Dans un premier temps une présentation succincte sera faite (I.1), puis une représentation de l'architecture (I.2) et une matrice de connectivité (I.3) seront données pour chacune.

I.1 Description

Nous travaillerons à partir de trois applications nommées : BitPool, BitStream et BitRivers. En voici une description succincte.

I.1.1 BitPool

Application de partage de fichiers entre appareils mobiles se connectant l'un à l'autre dans un réseau local. L'utilisateur interagit avec l'application via L'IHM Gestion de partage et consultation, aussi bien pour partager que pour récupérer des fichiers. Depuis cette interface l'utilisateur pourra aussi, au besoin, lancer deux modes de connexion :

- Wi-Fi P2P, qui permet d'établir un lien direct entre deux appareils sans passer par l'intermédiaire d'un hotspot Wi-Fi.

- Hotspot Wi-Fi, qui permettra à l'appareil mobile de jouer le rôle d'hotspot Wi-Fi.

Par défaut cependant, les appareils peuvent se connecter au même réseau Wi-Fi, déjà existant si l'endroit dans lequel ils se trouvent propose ce service. Des QR codes peuvent être générés et échangés pour transmettre les informations de connexions de manière plus conviviale. L'application permettra à l'utilisateur d'administrer les personnes autorisées à échanger des fichiers avec lui et leurs droits sur les fichiers qu'il souhaite partager.

I.1.2 BitStream

En plus de ce qui est possible avec BitPool, BitStream permet une utilisation « nomade » ou distante. Les utilisateurs n'ont plus besoins d'être dans le même réseau local. Un serveur distant (LINK) fait le lien entre appareils mobiles de personnes qui se connaissent déjà. Les personnes qui souhaitent partager des fichiers à distance doivent s'échanger un identifiant unique obtenu auprès du serveur à la première utilisation. Lorsqu'une personne ouvre l'application, elle s'enregistre auprès du serveur avec cet identifiant unique (ID unique). Le serveur obtient aussi son adresse IP publique. Le serveur crypte cette IP et la stocke dans une table, ce qui donne un couple ID unique, adresse IP. Un utilisateur souhaitant consulter les fichiers d'un autre interroge le serveur distant (LINK) et obtient l'IP publique cryptée de l'utilisateur dont il connait l'ID unique. Les deux communiqueront directement. Le serveur LINK ne fait rien d'autre et ne stocke pas d'informations au-delà d'un temps raisonnable. Dans un contexte de communication hors réseau local, des mécanismes de sécurité sont présents. Un composant UPnP permettra la configuration du routeur d'accès à Internet des appareils mobiles, pour résoudre les problèmes de NAT/PAT.

I.1.3 BitRivers

Cette application n'est conçue que pour le partage distant. Les composants hotspot Wi-Fi et Wi-Fi P2P disparaissent. Seul est ajouté le composant représentant l'API REST BitTorrent Sync qui permettra de tirer parti des possibilités offertes par les réseaux pear to pear BitTorrent.

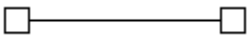
Cette API donne la possibilité de créer une clé correspondant au fichier ou au dossier à partager. Cette paire clé/hôte du fichier permet de retrouver les personnes ayant les fichiers voulus (grâce à la clé). La clé peut être partagée de plusieurs manières, notamment par QR code et liens HTTPs. Elle peut être protégée par son créateur de deux manières : en exigeant que son approbation soit donnée avant qu'elle ne soit partagée par un pair avec un autre pair ou en

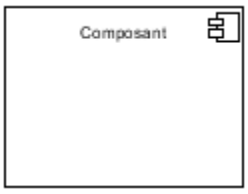
définissant l'expiration du lien selon une date et/ou un nombre d'utilisations que son créateur choisit.

L'intérêt du serveur LINK ici est de permettre la création d'un réseau de partage P2P privé, un réseau de confiance plus restreint qu'un réseau totalement ouvert P2P si on le souhaite. Toutefois une ouverture totale reste possible si voulue par le créateur de la clé. De plus le partage de la clé peut aussi être rendue superflu dans le cadre d'un réseau de contacts : un client qui obtiendra l'IP d'un de ses contacts pourra l'interroger pour obtenir la liste des fichiers disponibles en téléchargement et les clés correspondantes.

I.2 Représentation par diagramme

Pour continuer ce travail il est nécessaire de s'appuyer sur un outil visuel, facilement exploitable pour observer les composants et connecteurs. Ces représentations sont des représentations abstraites des architectures logicielles des applications servant de base à ce mémoire. Elles suivent en grande partie le formalisme des diagrammes de composants UML.

 : Interface et lien entre composants



: Composant

Les applications présentes sur chaque appareil mobile sont identiques. Toutefois, par souci de simplicité, seule la partie client ou la partie serveur est représentée afin de mettre en évidence les composants impliqués dans l'échange de fichiers entre les appareils selon leurs rôles.

I.2.1 BitPool

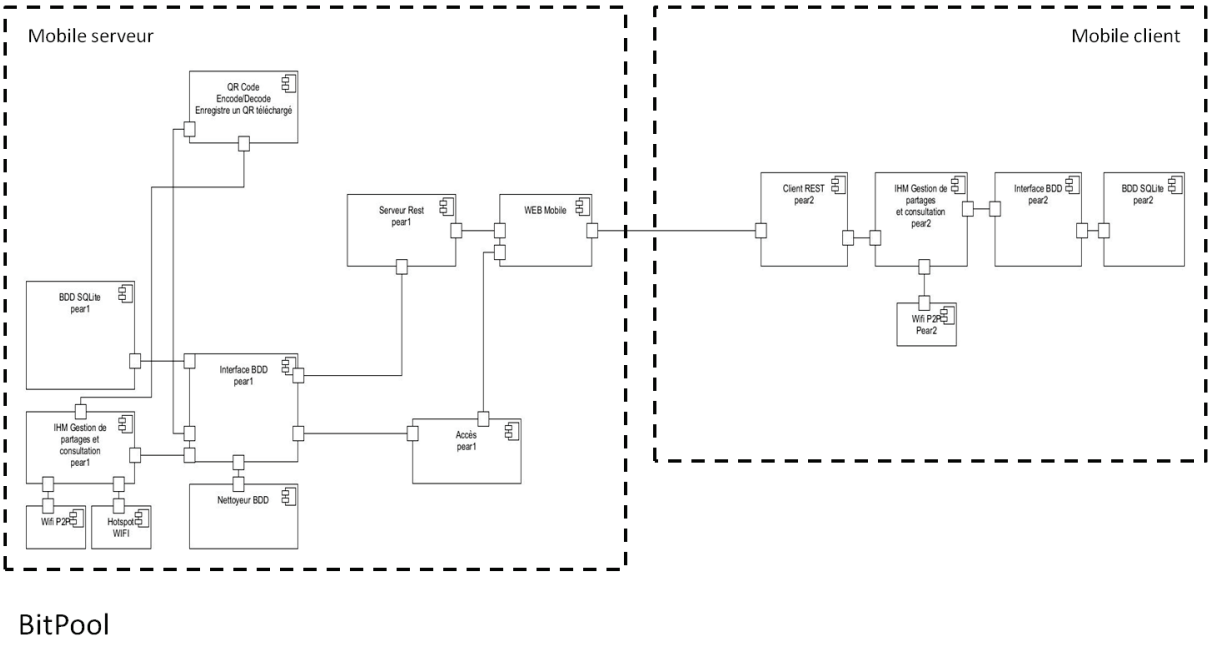


Figure 1 – Représentation de l'architecture BitPool

I.2.2 BitStream

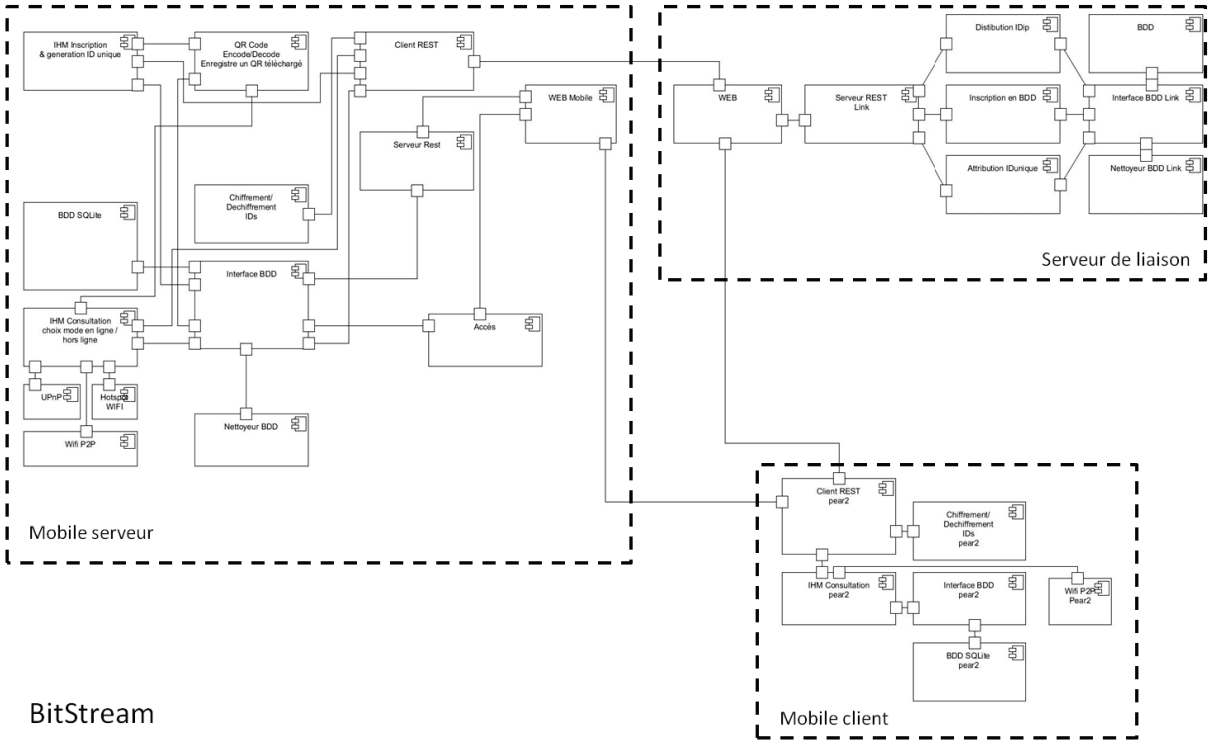


Figure 2 – Représentation de l'architecture BitStream

I.2.3 BitRivers

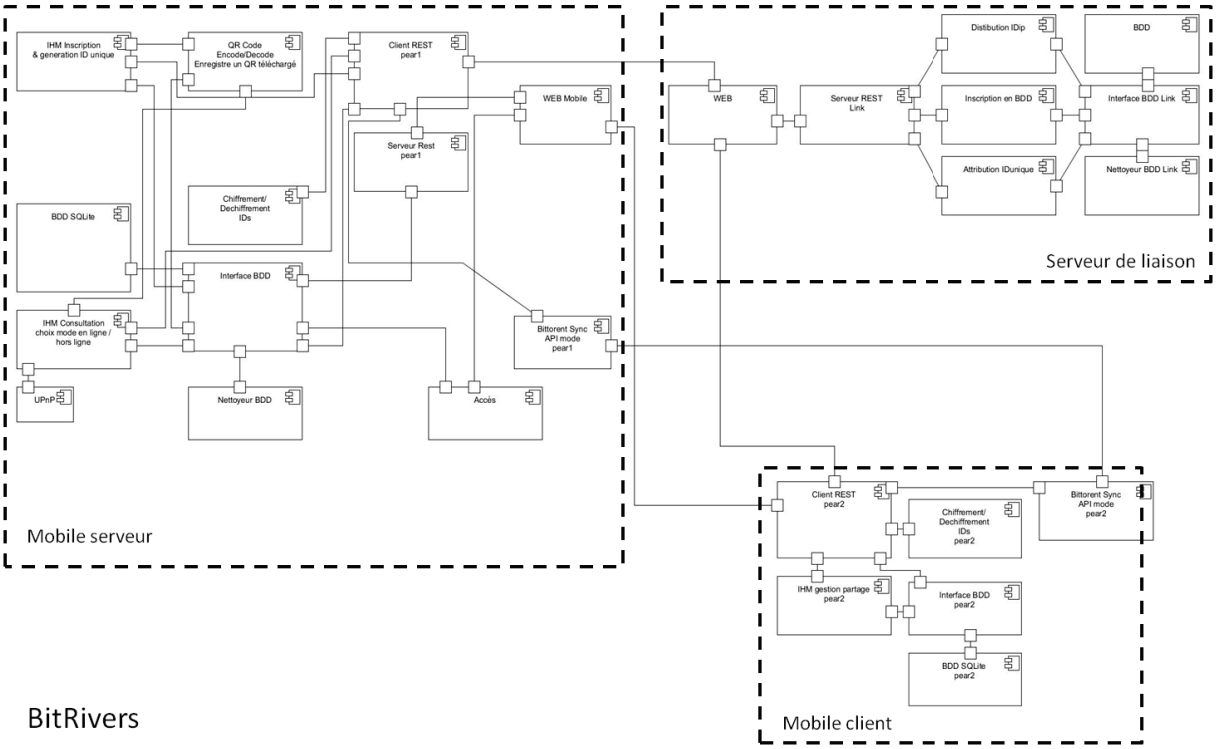


Figure 3 – Représentation de l'architecture BitRivers

I.3 Représentation par matrices de connectivités

N. Levy *et al* (4) proposent d'étudier les composants et connecteurs des applications servant de base à notre étude pour déterminer s'ils ont des similarités. Ils considèrent l'architecture d'une application comme un graphe de composants reliés par des arcs orientés (leurs connecteurs). Pour représenter ce graphe, ils invitent à utiliser des matrices de connectivité comme support de travail sur les architectures logicielles. Elles mettront en évidence les connexions entre composants. Leur élaboration est aussi un travail préparatoire à la définition du noyau et des variants de l'architecture de référence. Voici plus en détail comment sont élaborées ces matrices (4) : une configuration architecturale est un couple S = (C, f) où C est un ensemble de composants, f est une relation qui associe à tout couple (ci, cj) du produit cartésien C x C une expression de chemin de c_i à c_j, chemin pouvant être vide et noté Ø. Une configuration peut donc aussi être définie par un couple (I, M),

I : une fonction d'indexation C ✕ {1, …, N}, associant à tout c_i de C un indice i compris entre 1 et N. On associe donc à chaque composant un indice de 1 à |C|.

M est alors la matrice carrée de dimension N = Card (C) telle que M [i, j] = f (c_i, c_j). M [1, |C|]^2, M [i, j] = f(ci, cj). Pour les matrices de connectivité M suivantes : M[i,j] = 1 si le composant i est directement connecté au composant j via un connecteur non précisé ; 0 sinon.

I.3.1 BitPool

BitPool		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	BDD SQLite	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	IHM Gestion de partages et consultation	0	0	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0
3	Hotspot Wifi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	Wifi P2P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	QR Code	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
6	Interface BDD	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	Nettoyeur BDD	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
8	Serveur REST	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
9	Accès pear1	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
10	WEB Mobile	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0
11	client REST pear2	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
12	IHM Gestion de partages et consultation - pear2	0	0	0	0	0	0	0	0	0	0	**1**	0	**1**	0	**1**
13	Interface BDD pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
14	BDD SQLite pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	Wifi P2P pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tableau 1 – Matrice de connectivité BitPool

I.3.2 BitStream

BitStream		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1	IHM inscription	0	0	0	0	0	0	**1**	0	**1**	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	BDD SQLite pear1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	IHM Gestion de partages et consultation - pear1	0	0	0	**1**	**1**	**1**	**1**	0	**1**	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	UPnP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	Hotspot Wifi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	Wifi P2P pear1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	QR Code	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	Chiffrement/Decriffrement Ids	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	Interface BDD pear1	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	Nettoyeur BDD	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	Client REST pear1	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	Serveur REST pear1	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	Accès pear1	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	WEB Mobile pear1	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	WEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0
16	Serveur REST Link	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0
17	Distribution Idip	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
18	Inscription en BDD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
19	Attribution Idunique	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
20	BDD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	Interface BDD Link	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
22	Nettoyeur BDD Link	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
23	client REST pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	**1**	0	0	0
24	IHM Gestion de partages et consultation - pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	**1**	0	**1**
25	Chiffrement/Decriffrement Ids pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	Interface BDD pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
27	BDD SQLite pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	Wifi P2P pear 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tableau 2 – Matrice de connectivité BitStream

I.3.3 BitRivers

BitRivers	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1 IHM inscription	0	0	0	0	**1**	0	**1**	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2 BDD SQLite pear1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3 IHM Gestion de partages et consultation - pear1	0	0	0	**1**	**1**	0	**1**	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4 UPnP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5 QR Code	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6 Chiffrement/Decriffrement Ids	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7 Interface BDD pear1	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8 Nettoyeur BDD	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9 Client REST pear1	0	0	0	0	0	**1**	**1**	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
10 Serveur REST pear1	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11 Accès pear1	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12 WEB Mobile pear1	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13 BTSync pear1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14 WEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0
15 Serveur REST Link	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0
16 Distribution Idip	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
17 Inscription en BDD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
18 Attribution Idunique	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
19 BDD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20 Interface BDD Link	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
21 Nettoyeur BDD Link	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
22 client REST pear2	0	0	0	0	0	0	0	0	0	0	0	**1**	0	**1**	0	0	0	0	0	0	0	0	0	**1**	**1**	0	**1**
23 IHM Gestion de partages et consultation - pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	**1**	0	0
24 Chiffrement/Decriffrement Ids pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25 Interface BDD pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
26 BDD SQLite pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27 BTSync pear2	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tableau 3 – Matrice de connectivité BitRivers

II Définition de l'architecture de référence

A partir des produits logiciels proposés, nous allons tenter d'appliquer la méthode indiquée au fil de l'article « Définition d'une architecture de référence à l'aide de matrices de connectivité » (N. Levy *et al* (4)) afin d'obtenir une architecture noyau commune à nos applications et les éléments d'architecture optionnels, appelés « variants », que l'on retrouve dans les différentes applications. Ceci sera la base de la famille de produits à partir de laquelle d'autres produits logiciels pourront être dérivés.

Pour obtenir l'architecture noyau, il nous est recommandé de nous aider de « matrices de connectivités réduites » qui mettent en évidence les liens entre composants d'intérêt pour chaque produit logiciel, en faisant abstraction de la technique de communication utilisée entre ces composants (II.1).

Une fois ces matrices obtenues, il est proposé d'effectuer une fusion de ces représentations matricielles afin de déterminer les composants communs aux applications considérées (II.2), ce qui permettra d'exprimer chaque produit logiciel en termes de décomposition noyau/parties variantes (représentant les éléments propres au produit. II.3). Enfin, une expression de l'architecture de référence sera donnée (II.4).

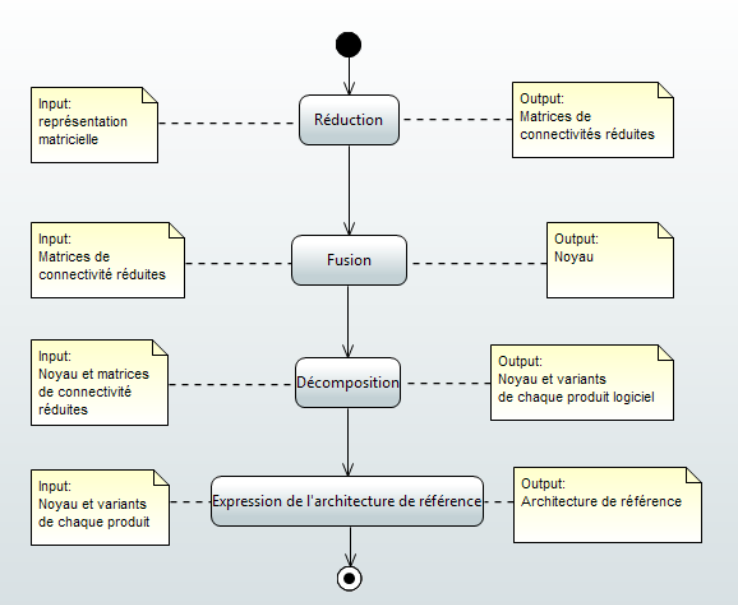


Figure 4 – Diagramme d'activité : définition de l'architecture de référence

II.1 Représentation matricielle réduite

N. Levy *et al.* (4) généralisent la représentation par matrice de connectivité pour mémoriser les chemins entre composants : C[i,j] = $ch_{i,j}$, $ch_{i,j}$ étant le chemin permettant d'aller du composant i au composant j. Si $ch_{i,j}$ est le chemin vide, les deux composants ne communiquent pas. Cette représentation sera utilisée pour la suite.

II.1.1 Méthode

Comme déjà cité en préambule, N. Levy *et al.* (4) s'intéressent aux chemins de communication entre composants similaires, en faisant abstraction de la technique de communication utilisée. Il va donc falloir séparer les composants servant à la communication et les autres. Les premiers seront des composantes des différents chemins techniques $ChT_{i,j}$.

II.1.1.1 **Définition des chemins techniques**

N. Levy *et al.* (4) définissent deux types de composants : les composants techniques et les composants applicatifs. Ils séparent l'ensemble des composants C d'une architecture (que l'on peut représenter par un graphe dont les sommets sont des composants) en deux classes A et T tels que C = A U T et A ∩ T = Ø (A, pour les composants applicatifs et T pour les composants techniques). La répartition dans les deux classes de composants est faite par l'architecte logiciel.

Ce après quoi, la notion de chemin technique est introduite. Un chemin technique est un DAG (directed acyclic graph, sans circuit donc) avec une seule entrée et une seule sortie dont les sommets sont des composants applicatifs $A_i \in A$. Il peut comporter des branches parallèles (notées +) (4).

L'identification des chemins techniques se fait par calcul de la fermeture transitive de la matrice de connectivité en appliquant la règle de composition * suivante entre un chemin ChT et un nouveau connecteur technique T :

ChT' = ChT * T

– Si Out (ChT) ≠ In (T), alors ChT' = Ø. Les chemins techniques ne sont composés que de composants techniques.

– Sinon si Out (ChT) ∈ A, alors ChT' = ChT. Un composant applicatif clôt un chemin technique.

– Sinon si Out(T) existe déjà dans ChT, alors ChT' = ChT. Un chemin technique est un DAG, il ne doit pas comporter de cycle.

– Sinon ChT' = concaténation (ChT, T).

La première tâche à effectuer sera donc de classer les composants en deux classes A (pour les composants applicatifs) et T (les composants techniques), puis de déterminer les chemins techniques reliant les composants applicatifs.

II.1.1.2 **Définition de la matrice de connectivité réduite**

Une fois définis les chemins techniques, on va simplifier l'architecture considérée en remplaçant chaque chemin technique par un composant technique et réduire la matrice en supprimant les lignes et colonnes de ces composants. C'est ce que N. Levy *et al.* (4) appellent la représentation matricielle réduite d'un système.

Dans le graphe représentant notre application, les chemins techniques deviendront des arcs entre nos composants applicatifs. M étant la matrice de connectivité de la configuration, on détermine donc la matrice C des chemins C[i, j] entre composants applicatifs, dite matrice de connectivité réduite par la formule :

$C = M + M2 + \ldots. + Mp$ où p est la taille de la matrice M, c'est-à-dire le nombre total de composants, lequel détermine la longueur maximale des chemins. On aura l'ensemble simplifié des chemins techniques entre les composants applicatifs, sans cycle. Le graphe correspondant à la matrice C reliera par un arc les sommets appartenant à (A) pour lesquels il existe un chemin de longueur quelconque entre eux dans le graphe d'origine.

II.1.2 Application

II.1.2.1 **Définition des classes de composants A et T**

La répartition dans les deux classes de composants est faite par l'architecte logiciel. Pour comprendre ce qui a motivé les choix faits, une définition de ce qui sera considéré comme un composant technique est proposée ci-après. Le résultat de l'intersection entre les deux classes étant l'ensemble vide, ce qui n'est pas composant technique, est composant applicatif.

Un composant technique permet la réalisation de fonctionnalités ou d'exigences attendues par l'utilisateur, mais non clairement exprimées ou identifiées par celui-ci. Il ne remplit pas de

fonction perçue par l'utilisateur final. Voici pour chaque application les classes A, de composants applicatifs et T, de composants techniques.

II.1.2.1.1 **BitPool**

A	T
BDD SQLite	Interface BDD
IHM Gestion Partages	Serveur REST
HotSpot Wifi	WEB Mobile
Wifi P2P	Client REST
QR Code	Interface BDD Pear2
Nettoyeur BDD	
Accès	
IHM Gestion Partages Pear2	
BDD SQLlite Pear2	
Wifi P2P Pear2	

Tableau 4 – Classes de composants BitPool

II.1.2.1.2 **BitStream**

A	T
IMH Insciption	Interface BDD
BDD SQLite	Client REST
IHM Gestion Partages	Serveur REST
UpnP	WEB Mobile
HotSpot Wifi	WEB
Wifi P2P	Serveur REST Link
QR Code	Interface BDD Link
Chiffrement/Dechiffrement IP	Client REST Pear2
Nettoyeur BDD	Interface BDD Pear2
Accès	
Distribution Idip	
Inscription en BDD	
Attribution Idunique	
BDD	
Nettoyeur BDD Link	
IHM Gestion Partages Pear2	
Chiffrement/Déchiffrement IP Pear2	
BDD SQLlite Pear2	
Wifi P2P Pear2	

Tableau 5 – Classes de composants BitStream

II.1.2.1.3 **BitRivers**

A	T
IMH Insciption	Interface BDD
BDD SQLite	Client REST
IHM Gestion Partages	Serveur REST
UpnP	WEB Mobile
QR Code	WEB
Chiffrement/Dechiffrement IP	Serveur REST Link
Nettoyeur BDD	Interface BDD Link
Accès	Client REST Pear2
BT Sync API mode	Interface BDD Pear2
Distribution Idip	
Inscription en BDD	
Attribution Idunique	
BDD	
Nettoyeur BDD Link	
IHM Gestion Partages Pear2	
Chiffrement/Déchiffrement IP Pear2	
BDD SQLlite Pear2	
BT Sync API mode Pear2	

Tableau 6 – Classes de composants BitRivers

II.1.2.2 **Indexation de la matrice de connectivité**

A partir de cette définition des chemins techniques, on peut à nouveau représenter nos architectures par des matrices de connectivité en mettant en évidence les composants techniques. Les colonnes et les lignes de composants appartenant à la classe T sont colorées en bleu. Les liens entre composants techniques, sont indiqués en un bleu plus foncé. Par ailleurs les composants sont numérotés et se suivent ainsi : en reprenant les diagrammes donnés lors de la présentation des applications (I.2), on numérote de 1 à Card(C) (C, l'ensemble des composants de l'application) en partant de haut en bas, de gauche à droite.

BitPool		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	BDD SQLite	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	IHM Gestion de partages et consultation	0	0	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0
3	Hotspot Wifi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	Wifi P2P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	QR Code	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
6	Interface BDD	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	Nettoyeur BDD	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
8	Serveur REST	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
9	Accès	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
10	WEB Mobile	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0
11	client REST pear2	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
12	IHM Gestion de partages et consultation - pear2	0	0	0	0	0	0	0	0	0	0	**1**	0	**1**	0	**1**
13	Interface BDD pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
14	BDD SQLite pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	Wifi P2P pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Chemins techniques - BitPool

composants techniques

1 lien entre composants techniques

Tableau 7 – Composants BitPool

BitStream		1	2	4	5	6	7	8	10	11	12	13	14	15	16	18	19	20	21	22	24	25	26	27	28	29	30	31	33
1	IHM inscription							1		1		1																	
2	BDD SQLite pear1																												
4	IHM Gestion de partages et consultation				1	1	1	1		1		1																	
5	UPnP																												
6	Hotspot Wifi																												
7	Wifi P2P																												
8	QR Code									1																			
10	Chiffrement/Decriffrement Ids																												
11	Interface BDD		1																										
12	Nettoyeur BDD									1																			
13	Client REST								1	1						1													
14	Serveur REST									1																			
15	Accès									1																			
16	WEB Mobile												1	1															
18	WEB																1												
19	Serveur REST Link																	1	1	1									
20	Distribution Idip																					1							
21	Inscription en BDD																					1							
22	Attribution Idunique																					1							
24	BDD																												
25	Interface BDD Link																				1								
26	Nettoyeur BDD Link																					1							
27	client REST pear2														1	1										1			
28	IHM Gestion de partages et consultation - pear2																							1			1		1
29	Chiffrement/Decriffrement Ids pear2																												
30	Interface BDD pear2																											1	
31	BDD SQLite pear2																												
33	Wifi P2P pear 2																												

Chemins techniques - BitStream

composants techniques

1 lien entre composants techniques

Tableau 8 – Chemin techniques BitStream

BitRivers		1	2	4	5	8	10	11	12	13	14	15	16	17	18	19	20	21	22	24	25	26	27	28	29	30	31	32
1	IHM inscription					1		1		1																		
2	BDD SQLite pear1																											
4	IHM Gestion de partages et consultation - pear1				1	1		1		1																		
5	UPnP																											
8	QR Code							1																				
10	Chiffrement/Decriffrement Ids																											
11	Interface BDD pear1		1																									
12	Nettoyeur BDD							1																				
13	Client REST pear1						1	1						1	1													
14	Serveur REST pear1							1																				
15	Accès pear1							1																				
16	WEB Mobile										1	1																
17	BTSync pear1																											
18	WEB															1												
19	Serveur REST Link																1	1	1									
20	Distribution Idip																				1							
21	Inscription en BDD																				1							
22	Attribution Idunique																				1							
24	BDD																											
25	Interface BDD Link																			1								
26	Nettoyeur BDD Link																				1							
27	client REST pear2												1		1										1	1		1
28	IHM Gestion de partages et consultation - pear2																						1			1		
29	Chiffrement/Decriffrement Ids pear2																											
30	Interface BDD pear2																										1	
31	BDD SQLite pear2																											
32	BTSync pear2													1														

Chemins techniques - BitRivers

composants techniques

1 lien entre composants techniques

Tableau 9 – Chemins techniques BitRivers

II.1.2.3 **Détermination des chemins techniques**

II.1.2.3.1 **Méthode visuelle**

A partir du travail préliminaire précédent, nous pouvons établir les chemins techniques. On remarque que

$\forall a \in (A), t \in (T)$. Un 1 placé à l'intersection d'une :

- ligne blanche et d'une colonne bleue : indique un lien d'un composant applicatif vers un composant technique a → t

- colonne blanche et d'une ligne bleue : indique un lien d'un composant technique à une composant applicatif t → a

- ligne et colonne bleue : indique un lien entre composants techniques

t → t

Un chemin technique partant d'un composant applicatif vers un autre, voici un exemple visuel depuis la ligne 12 de la matrice de connectivité de BitPool. Le premier « 1 » rencontré, à la colonne 11, pointe vers un composant technique (la colonne 11 est bleue). Je me replace sur la ligne de ce composant technique, la ligne 11. Le premier « 1 » pointe aussi vers un composant technique ([11,10]). Je me place à la ligne 10 et continue récursivement. Si un « 1 » se trouve à l'intersection d'une colonne blanche, c'est que ce composant pointe vers un composant applicatif. C'est donc la fin du chemin technique. Dans cet exemple, le chemin technique complet est [12-11-10-8-6-1].

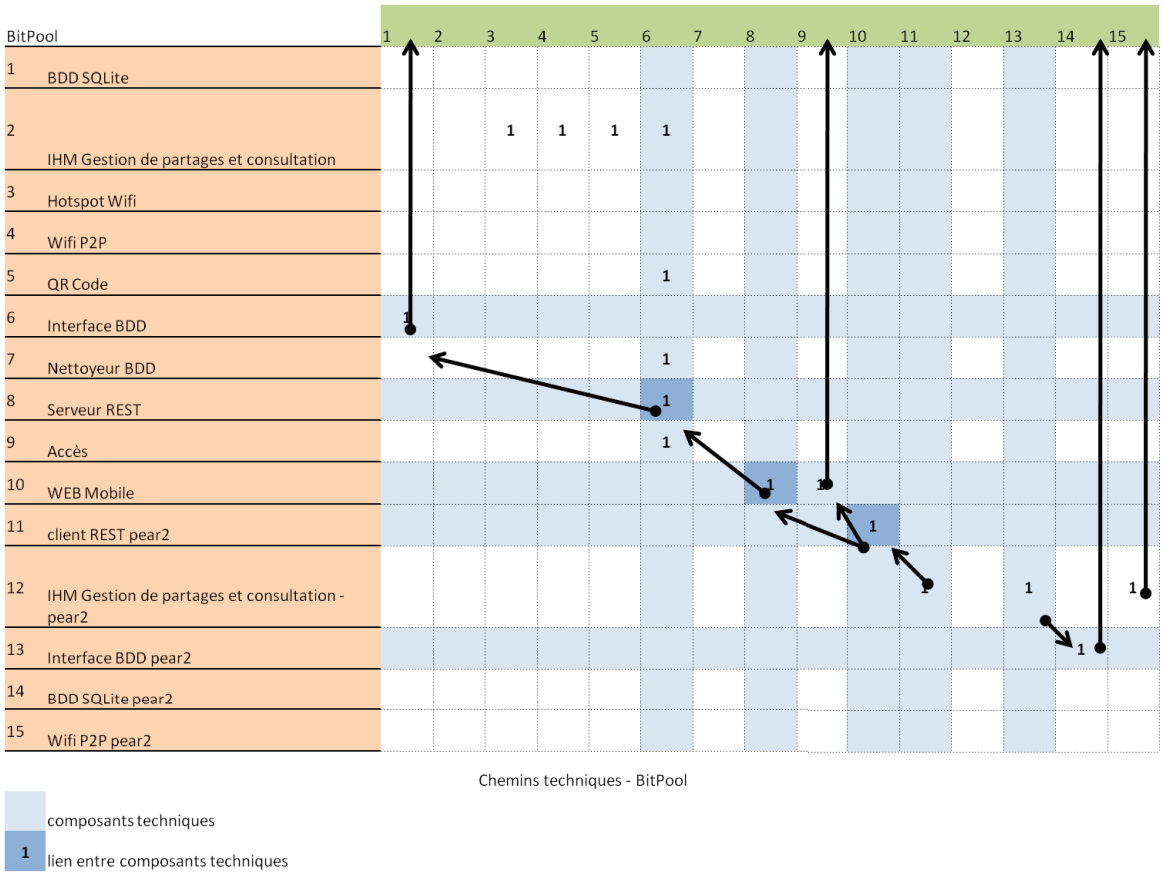


Figure 5 – Détermination visuelle d'un chemin technique.

II.1.2.3.2 **Outil Java**

J'ai développé un outil JAVA[1] permettant de déterminer les chemins techniques possibles à partir de la matrice de connectivité d'un produit et de l'indication de ses composants techniques.

La matrice et l'indication sur les composants techniques sont à fournir dans un fichier texte sauvegardé sur le disque, tous deux séparés par une ligne vide. Par exemple pour BitPool :

[1] Le diagramme de classe et le code source sont disponnibles à cette adresse https://github.com/gregoryzu/irsm

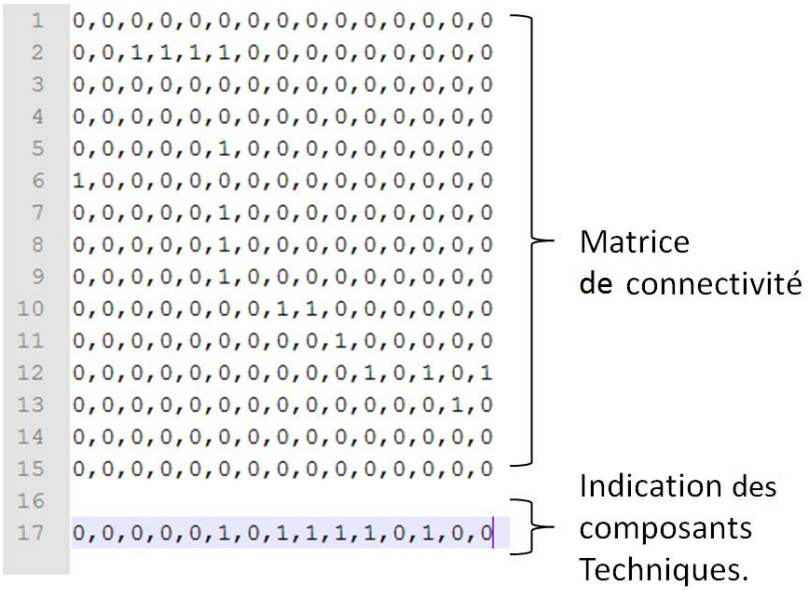


Figure 6 – Exemple de fichier texte

En lançant ensuite l'application, une première fenêtre apparaît pour charger le fichier texte en mémoire.

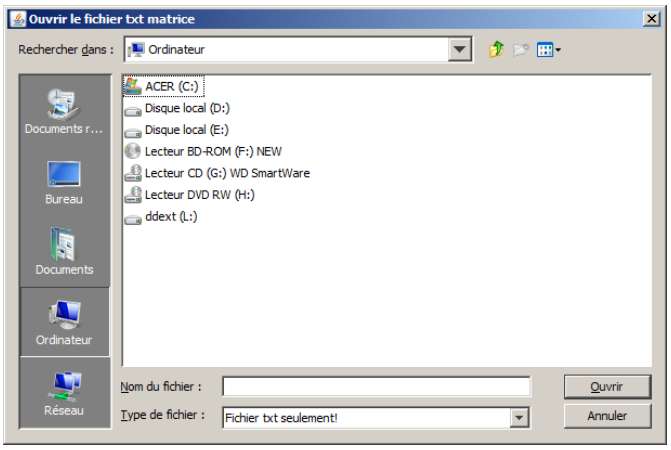


Figure 7 - Choix du fichier

Une fois celui-ci chargé, l'application fait le reste et affiche les chemins techniques sous forme d'arborescence. Sous les chemins techniques, un JEditorPane affiche une page html contenant un bref descriptif de ce qui est fait.

Figure 8 – IHM

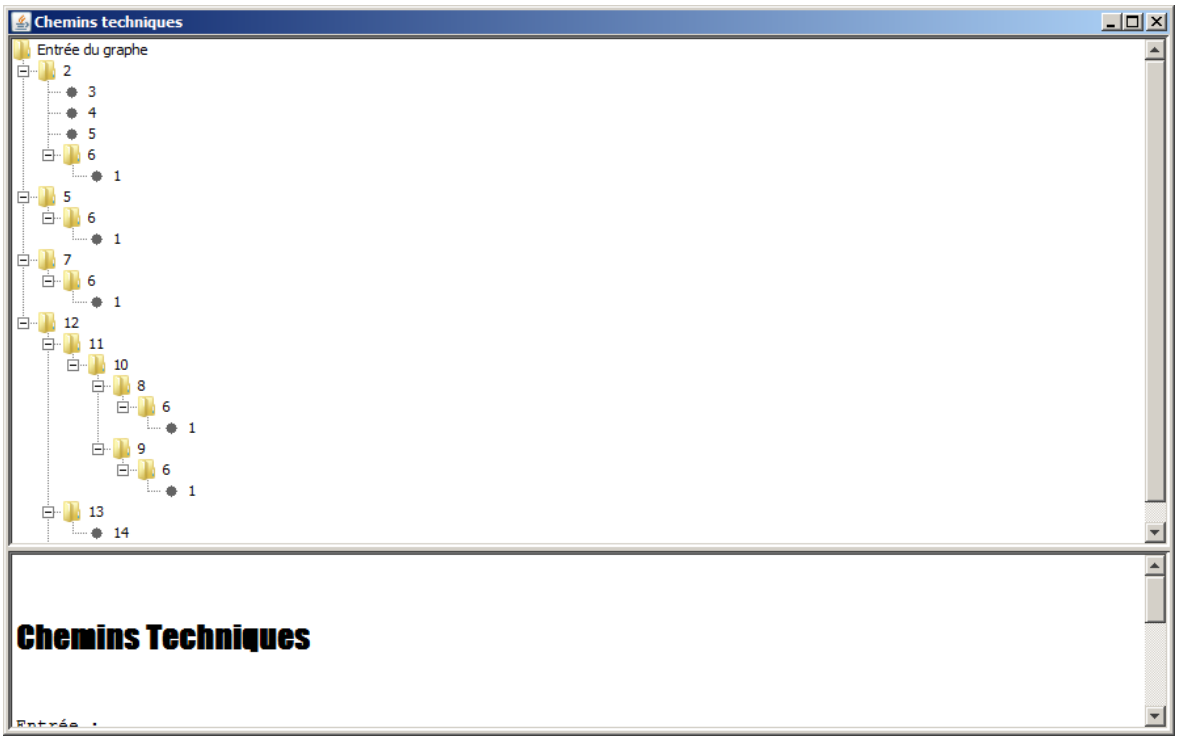


Figure 9 – Aperçu des chemins techniques

Chaque chemin technique commence par un composant applicatif et se termine par un composant applicatif. Tous les composants applicatifs sont repris en tant que « racine » de l'ensemble des chemins qu'il ouvre. Il est aussi indiqué les connections directes entre applications pour information. Par exemple [2-5] est indiqué. 5 ouvre lui aussi un chemin

technique [5-6-1]. Il est donc repris en tant que racine. Ce cas montre qu'il existe trois types de composants applicatifs :

- Ceux qui ne sont que des « racines ». Aucun autre composant ne s'y connecte. Ils sont facilement identifiables par le fait que la colonne qui les représente dans la matrice ne contient que des « 0 ».

- Ceux qui ne sont que des feuilles. Ils ne se connectent à aucun autre composant applicatif. Ils sont aussi facilement identifiables. La ligne qui les représente n'indique que des « 0 ».

- Ceux auxquels d'autres composants se connectent, comme 5 et qui ouvrent un chemin technique. Ainsi le chemin [2-5-6-1] doit être décomposé en deux chemins : [2-5] et le chemin technique [5-6-1].

II.1.2.4 **Abstraction des chemins techniques**

Cette tâche s'effectue en trois temps. Tout d'abord, un chemin technique sera identifié par la suite des composants techniques traversés dans la communication d'un composant applicatif à un autre, séparés par un point. Pour obtenir la matrice de connectivité réduite C depuis la matrice de connectivité M de nos applications, les lignes et les colonnes représentant les composants d'un chemin technique dans M seront supprimées. Enfin, l'expression de connectivité entre composants applicatif sera :

$C[i,j] = ChT_{i,j}$ si les composants applicatifs i et j communiquent par l'intermédiaire d'un chemin technique, 1 s'ils communiquent directement, Ø sinon.

II.1.3 Résultat sur les applications

Les chemins barrés représentent des chemins techniques qui, bien que possibles, ne sont pas pertinents dans le cadre de nos applications.

II.1.3.1 **BitPool**

BitPool	1	2	3	4	5	6	7	8	9	10
1 BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
2 IHM Gestion de partages et consultation	**Interface BDD**	Ø	**1**	**1**	**1**	Ø	Ø	Ø	Ø	Ø
3 Hotspot Wifi	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
4 Wifi P2P	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
5 QR Code	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6 Nettoyeur BDD	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7 Accès	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
8 IHM Gestion de partages et consultation - pear2	**Client Rest P2.WEB Mobile.Serveur REST.Interface BDD**	Ø	Ø	Ø	Ø	Ø	**Client Rest P2.WEB Mobile**	Ø	**Interface BDD Pear2**	**1**
9 BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
10 Wifi P2P pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tableau 10 – Représentation matricielle réduite BitPool

II.1.3.2 **BitStream**

BitStream	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1 IHM inscription		**Interface BDD + ~~Client REST.Interface BDD~~**		Ø	Ø	Ø	**1**	*Client REST*	Ø	Ø	*Client REST.WEB.Serveur REST Link*	*Client REST.WEB.Serveur REST Link*	**Client REST.WEB.Serveur REST Link**	Ø	Ø	Ø	Ø	Ø	Ø
2 BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
3 IHM Gestion de partages et consultation		**Interface BDD + ~~Client REST.Interface BDD~~**		**1**	**1**	**1**	**1**	*Client REST*	Ø	Ø	**Client REST.WEB.Serveur REST Link**	**Client REST.WEB.Serveur REST Link**	*Client REST.WEB.Serveur REST Link*	Ø	Ø	Ø	Ø	Ø	Ø
4 UPnP	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
5 Hotspot Wifi	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6 Wifi P2P	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7 QR Code	Ø	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
8 Chiffrement /Decriffrement Ids	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
9 Nettoyeur BDD	Ø	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
10 Accès	Ø	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
11 Distribution Idip	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
12 Inscription en BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
13 Attribution Idunique	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
14 BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
15 Nettoyeur BDD Link	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
16 IHM Gestion de partages et consultation - pear2	Ø	**Client REST Pear2.WEB Mobile.Serveur REST.Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Client REST pear2.WEB Mobile**	**Client REST Pear2.WEB. Serveur REST Link**	**Client REST Pear2.WEB. Serveur REST Link**	*Client REST Pear2.WEB. Serveur REST Link*	Ø	Ø	Ø	*Client REST Pear2*	**Interface BDD Pear2**	1
17 Chiffrement /Decriffrement Ids pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
18 BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
19 Wifi P2P Pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tableau 11 – Représentation matricielle réduite BitStream

II.1.3.3 **BitRivers**

BitRivers		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	IHM inscription	Ø	**Interface BDD +** ~~**Client REST.Interface BDD**~~	Ø	Ø	**1**	*Client REST*	Ø	Ø	Ø	*Client REST.WEB.Serveur REST Link*	*Client REST.WEB.Serveur REST Link*	**Client REST.WEB.Serveur REST Link**	Ø	Ø	Ø	Ø	Ø	Ø
2	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
3	IHM Gestion de partages et consultation		**Interface BDD +** ~~**Client REST.Interface BDD**~~	Ø	**1**	**1**	*Client REST*	Ø	Ø	**Client REST**	**Client REST.WEB.Serveur REST Link**	**Client REST.WEB.Serveur REST Link**	*Client REST.WEB.Serveur REST Link*	Ø	Ø	Ø	Ø	Ø	Ø
4	UPnP	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
5	QR Code		**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6	Chiffrement /Decriffrement Ids	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7	Nettoyeur BDD		**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
8	Accès		**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
9	BTSync	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
10	Distribution Idip	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
11	Inscription en BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
12	Attribution Idunique	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
13	BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø		Ø	Ø	Ø	Ø	Ø
14	Nettoyeur BDD Link	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
15	IHM Gestion de partages et consultation - pear2	Ø	**Client REST pear2.WEB Mobile.Serveur REST.Interface BDD**	Ø	Ø	Ø	Ø	Ø	**Client REST pear2.WEB Mobile**	Ø	**Client REST pear2.WEB. Serveur REST Link**	**Client REST pear2.WEB. Serveur REST Link**	*Client REST pear2.WEB. Serveur REST Link*	Ø	Ø	Ø	*Client REST pear 2*	**Interface BDD pear 2** ~~**+ Client REST pear2.Interface BDD pear2**~~	**Client REST Pear2**
16	Chiffrement /Decriffrement Ids pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
17	BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
18	BTSync pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**1**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tableau 12 – Représentation matricielle réduite BitRivers

Voici un diagramme d'activité reprenant les étapes de l'élaboration d'une matrice de connectivité réduite.

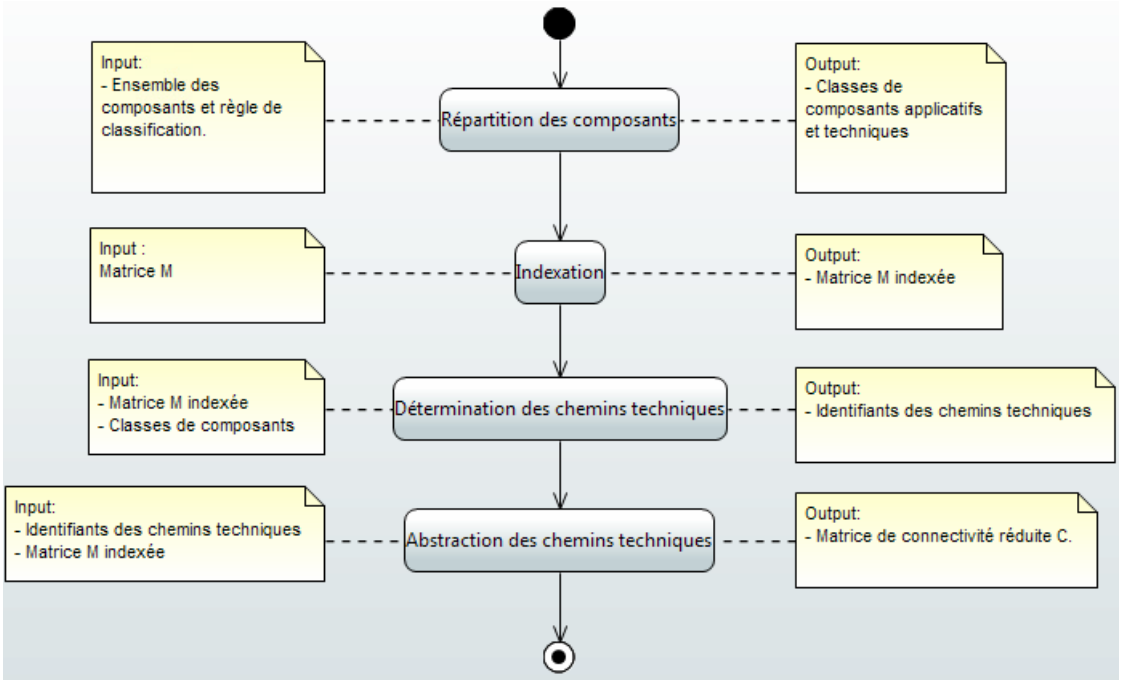


Figure 10 – Diagramme d'activité : élaboration d'une matrice de connectivité réduite C.

II.2 Définition du noyau de l'architecture de référence

A partir du travail préliminaire précédent, nous allons pouvoir suivre la démarche proposée par N. Levy *et al.* (4) et déterminer le noyau commun à plusieurs produits logiciels d'une ligne de produits. Cette section présente la méthode pour y parvenir à partir des matrices de connectivité réduites de nos applications.

II.2.1 Fusion des représentations matricielles réduites

Le noyau d'une architecture de référence est obtenu par l'intersection des composants de tous les produits servant de base à la ligne de produits. Soit S1 et S2 deux configurations architecturales d'un produit logiciel. Le noyau associé à S1 et S2 sera donc la configuration définie par (C_{Noyau}, F_{Noyau}) telle que :

$C_{Noyau} = C1 \cap C2$ et F_{Noyau} (ci, cj) = Ch1 (ci, cj) | Ch2 (ci, cj)

Le calcul du noyau s'effectue sur les matrices de la manière suivante (en supposant que les deux matrices de départ C1 et C2 sont en forme minimale) :

Début

Déterminer l'ensemble $C_1 \cap C_2$ des composants communs à S_1 et à S_2

Projeter la matrice M_1 sur les dimensions déterminées par $C_1 \cap C_2$ ($\rightarrow M_{1/c1 \cap c2}$)

Projeter la matrice M_2 sur les dimensions déterminées par $C_1 \cap C_2$ ($\rightarrow M_{2/c1 \cap c2}$)

Choisir une fonction d'indexation I : $C_1 \cap C_2 \cap \{1, \ldots, \text{Card}(C_1 \cap C_2)\}$

Réordonner $M_{1/c1 \cap c2}$ et $M_{2/c1 \cap c2}$ selon l'indexation commune I

Pour tout (i, j) Faire

$$M_{Noyau}(i,j) = M_{1/c1 \cap c2}(i,j) \mid M_{2/c1 \cap c2}(i,j)$$

Fin Pour tout

Mettre sous forme minimale la matrice M_{Noyau}

Fin (4)

On voit quatre étapes principales. On cherche d'abord à déterminer les composants communs aux deux configurations architecturales S_1 et à S_2. Puis on projette leurs matrices sur les dimensions résultant de cette intersection afin d'obtenir les matrices de connectivité de ces composants dans S_1 et S_2. On choisit une fonction d'indexation commune à ces matrices puis on les ordonne de manière adéquate. Enfin, pour tout (i,j) on vérifie qu'il existe un chemin dans l'une ou l'autre configuration architecturale. A l'aide de ces chemins, la matrice noyau commun est élaborée en reprenant les composants communs indexés à l'aide de l'indexation déterminée plus tôt. Pour tout (i,j) on indique le chemin de S_1 et/ou celui de S_2.

Le noyau de l'architecture de référence de S1 et de S2 représentera donc l'ensemble des composants communs à S1 et S2 et leurs connexions tel que : lorsque que deux composants existent dans S1 et S2 et avec un chemin technique identique entre eux dans S1 et S2, ces composants et ce chemin se retrouveront dans le noyau. De même pour deux composants présents à la fois dans S1 et S2, mais connectés par deux chemins techniques différents (ChO1 et ChO2), ces composants seront présents dans le noyau mais la connexion sera indiquée par un chemin (technique) optionnel noté ChO1 | ChO2 (exprimé précédemment formellement par l'opération ou sur $M_{1/c1 \cap c2}(i,j) \mid M_{2/c1 \cap c2}(i,j)$). Lors de la dérivation, le chemin présent dans une application entre ces deux composants pourra être soit ChO1 soit chO2. N. Levy *et al.* (4) désignent cette variabilité par le terme de variabilité intra noyau.

La représentation matricielle réduite du noyau fait à nouveau appel à une abstraction des chemins techniques. Un chemin technique noyau (ChT_{noyau}) sera identifié par la suite des chemins techniques que l'on retrouve dans C1 et/ou C2, séparés par un |. Enfin, l'expression de connectivité entre composants applicatifs noyau sera :

$M_{noyau}[i,j] = ChT_{noyau\ i,j}$ si les composants i et j communiquent par un chemin technique, 1 s'ils communiquent directement, Ø sinon.

II.2.2 Application

La méthode proposée repose sur l'intersection. Celle-ci étant commutative et associative, le même résultat est obtenu, peu importe l'ordre dans lequel est appliquée la méthode sur les différents produits logiciels. Toutefois, l'ordre choisi dans l'exécution peut influencer la rapidité d'exécution. Dans le cadre de ce mémoire, je choisis de faire l'intersection des configurations architecturales deux à deux, en commençant par les deux applications ayant le moins de composants techniques. Puis le résultat sera utilisé pour faire l'intersection avec l'application restante. Voici, un rappel des étapes à suivre sous forme de diagramme d'activité.

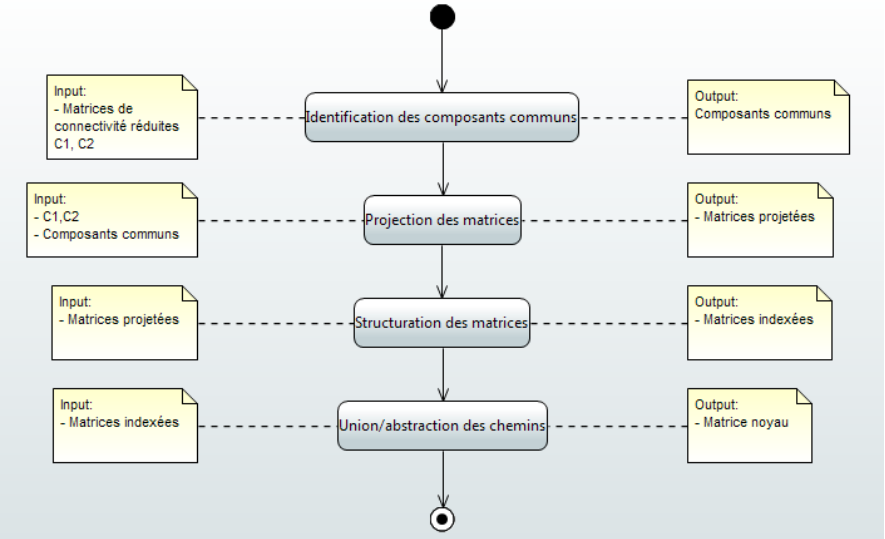


Figure 11 – Diagramme d'activité : définition du noyau de l'architecture de référence

La première opération à mener est celle de l'identification des composants « similaires » entre les deux configurations architecturales. Un ensemble de composants similaires ainsi identifiés formeront une classe d'équivalence. Une telle classe sera identifiée par un nom. Dans notre cas, cette identification est immédiate et le nom s'avère être le nom commun à tous les composants mis en relation. Notre cas n'a pas posé de problème puisque les trois applications ont été définies tout en sachant à l'avance qu'elles feraient partie d'une ligne de produits. Hors

de ce cadre idéal, l'article (4) propose des pistes de recherche. L'évaluation d'une telle similarité entre composants se base sur :

1) l'identité des exigences fonctionnelles allouées aux composants,

2) l'identité des « rôles architecturaux »,

3) accessoirement, la similarité du nom. Les similarités ainsi identifiées doivent être validées par un architecte expert du domaine.

Détaillons ce qui sera fait pour chaque noyau commun à deux applications. La tâche d'élaboration du noyau a été faite en ne considérant seulement deux configurations architecturales à la fois.

II.2.2.1 $C_{BitPool} \cap C_{BitStream}$

J'ai commencé par $C_{BitPool} \cap C_{BitStream}$. Leurs composants communs sont les suivants :

BDD SQLite
IHM Gestion de partages et consultation
Hotspot Wifi
Wifi P2P
QR Code
Nettoyeur BDD
Accès
IHM Gestion de partages et consultation - pear2
BDD SQLite pear2
Wifi P2P pear2

Tableau 13 – Composants communs $C_{BitPool} \cap C_{BitStream}$

A partir de ces composants, je peux projeter les matrices $C_{BitPool}$ et $C_{BitStream}$. Je reprends la matrice de connectivité réduite de BitStream et mets en évidence les composants communs avec BitPool :

BitStream	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1 IHM inscription		**Interface BDD + ~~Client REST.Interface BDD~~**		Ø	Ø	Ø	**1**	*Client REST*	Ø	Ø	*Client REST.WEB.Serveur REST Link*	*Client REST.WEB.Serveur REST Link*	**Client REST.WEB.Serveur REST Link**	Ø	Ø	Ø	Ø	Ø	Ø
2 BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
3 IHM Gestion de partages et consultation		**Interface BDD + ~~Client REST.Interface BDD~~**		**1**	**1**	**1**	**1**	*Client REST*	Ø	Ø	**Client REST.WEB.Serveur REST Link**	**Client REST.WEB.Serveur REST Link**	*Client REST.WEB.Serveur REST Link*	Ø	Ø	Ø	Ø	Ø	Ø
4 UPnP	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
5 Hotspot Wifi	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6 Wifi P2P	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7 QR Code	Ø	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
8 Chiffrement/ Decriffrement Ids	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
9 Nettoyeur BDD	Ø	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
10 Accès	Ø	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
11 Distribution Idip	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
12 Inscription en BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
13 Attribution Idunique	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
14 BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
15 Nettoyeur BDD Link	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
16 IHM Gestion de partages et consultation - pear2	Ø	**Client REST Pear2.WEB Mobile.Serveur REST.Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Client REST pear2.WEB Mobile**	**Client REST Pear2.WEB.Serveur REST Link**	**Client REST Pear2.WEB.Serveur REST Link**	*Client REST Pear2.WEB.Serveur REST Link*	Ø	Ø	Ø	*Client REST Pear2*	**Interface BDD Pear2**	1
17 Chiffrement/ Decriffrement Ids pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
18 BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
19 Wifi P2P Pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tableau 14 – Mise en avant des composants communs sur la matrice $C_{BitStream}$

Voici à nouveau la matrice débarrassée des composants applicatifs non communs :

BitStream		2	3	5	6	7	9	10	16	18	19
2	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
3	IHM Gestion de partages et consultation	**Interface BDD + ~~Client REST.Interface BDD~~**		**1**	**1**	**1**	Ø	Ø	Ø	Ø	Ø
5	Hotspot Wifi	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6	Wifi P2P	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7	QR Code	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
9	Nettoyeur BDD	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
10	Accès	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
16	IHM Gestion de partages et consultation - pear2	**Client REST Pear2.WEB Mobile.Serveur REST.Interface BDD**	Ø	Ø	Ø	Ø	Ø	**Client REST pear2.WEB Mobile**	Ø	**Interface BDD Pear2**	1
18	BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
19	Wifi P2P Pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tableau 15 – Projection de la matrice $C_{BitStream}$

Rien n'est à faire pour $C_{BitPool}$, celui-ci étant composé de l'ensemble des composants communs à ces deux architectures. On peut maintenant choisir une fonction d'indexation commune. Je reprends celle de $C_{BitPool}$.

Lors de l'union des chemins, on constate que les chemins entre composants similaires sont les mêmes dans $C_{BitPool}$ et $C_{BitStream}$. Il n'y a donc pas de chemins alternatifs à considérer. Le chemin technique dans la matrice $M_{Noyau(BitPool,BitStream)}$ sera le même que pour les matrices de connectivité réduites de $C_{BitPool}$ et $C_{BitStream}$.

N(BitPool,BitStream)		1	2	3	4	5	6	7	8	9	10
1	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
2	IHM Gestion de partages et consultation	**Interface BDD**	Ø	**1**	**1**	**1**	Ø	Ø	Ø	Ø	Ø
3	Hotspot Wifi	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
4	Wifi P2P	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
5	QR Code	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6	Nettoyeur BDD	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7	Accès	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
8	IHM Gestion de partages et consultation - pear2	**Client REST Pear2.WEB Mobile.Serveur REST.Interface BDD**	Ø	Ø	Ø	Ø	Ø	**Client REST pear2.WEB Mobile**	Ø	**Interface BDD Pear2**	1
9	BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
10	Wifi P2P pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tableau 16 – Matrice de connectivité réduite $C_{BitPool} \cap C_{BitStream}$

II.2.2.2 $M_{Noyau(BitPool, BitStream)} \cap C_{BitRivers}$

De la même manière, la liste des composants communs est la suivante :

BDD SQLite
IHM Gestion de partages et consultation
QR Code
Nettoyeur BDD
Accès
IHM Gestion de partages et consultation - pear2
BDD SQLite pear2

Tableau 17 – Liste des composants communs à toutes les applications.

A partir de ces composants, je peux projeter les matrices $M_{Noyau(BitPool,BitStream)}$ et $C_{BitRivers}$. Je reprends la matrice de connectivité réduite de BitRivers et mets en évidence les composants communs avec $M_{Noyau(BitPool, BitStream)}$:

BitRivers		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	IHM inscription	Ø	**Interface BDD + ~~Client REST.Interface BDD~~**	Ø	Ø	**1**	*Client REST*	Ø	Ø	Ø	*Client REST.WEB.Serveur REST Link*	*Client REST.WEB.Serveur REST Link*	**Client REST.WEB.Serveur REST Link**	Ø	Ø	Ø	Ø	Ø	Ø
2	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
3	IHM Gestion de partages et consultation		**Interface BDD + ~~Client REST.Interface BDD~~**	Ø	**1**	**1**	*Client REST*	Ø	Ø	**Client REST**	**Client REST.WEB.Serveur REST Link**	**Client REST.WEB.Serveur REST Link**	*Client REST.WEB.Serveur REST Link*	Ø	Ø	Ø	Ø	Ø	Ø
4	UPnP	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
5	QR Code		**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6	Chiffrement/ Decriffrement Ids	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7	Nettoyeur BDD		**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
8	Accès		**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
9	BTSync	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
10	Distribution Idip	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
11	Inscription en BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
12	Attribution Idunique	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
13	BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø		Ø	Ø	Ø	Ø	Ø
14	Nettoyeur BDD Link	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
15	IHM Gestion de partages et consultation - pear2	Ø	**Client REST pear2.WEB Mobile.Serveur REST.Interface BDD**	Ø	Ø	Ø	Ø	Ø	**Client REST pear2.WEB Mobile**	Ø	**Client REST pear2.WEB.Serveur REST Link**	**Client REST pear2.WEB.Serveur REST Link**	*Client REST pear2.WEB.Serveur REST Link*	Ø	Ø	Ø	*Client REST pear 2*	**Interface BDD pear 2 + ~~Client REST pear2.Interface BDD pear2~~**	**Client REST Pear2**
16	Chiffrement/ Decriffrement Ids pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
17	BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
18	BTSync pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**1**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tableau 18 - Mise en avant des composants communs sur la matrice $C_{BitRivers}$

Voici à nouveau la matrice débarrassée des composants applicatifs non communs :

BitRivers		2	3	5	7	8	15	17
2	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø
3	IHM Gestion de partages et consultation	**Interface BDD + ~~Client REST.Interface BDD~~**	Ø	**1**	Ø	Ø	Ø	Ø
5	QR Code	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø
7	Nettoyeur BDD	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø
8	Accès	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø
15	IHM Gestion de partages et consultation - pear2	**Client REST pear2.WEB Mobile.Serveur REST.Interface BDD**	Ø	Ø	Ø	**Client REST pear2.WEB Mobile**	Ø	**Interface BDD pear 2 + ~~Client REST pear2.Interface BDD pear2~~**
17	BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tableau 19 – Projection de la matrice $C_{BitRivers}$

Pour $M_{Noyau(BitPool, BitStream)}$ on obtient :

N(BitPool,BitStream)		1	2	5	6	7	8	9
1	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø
2	IHM Gestion de partages et consultation	**Interface BDD**	Ø	**1**	Ø	Ø	Ø	Ø
5	QR Code	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø
6	Nettoyeur BDD	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø
7	Accès	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø
8	IHM Gestion de partages et consultation - pear2	**Client REST Pear2.WEB Mobile.Serveur REST.Interface BDD**	Ø	Ø	Ø	**Client REST pear2.WEB Mobile**	Ø	**Interface BDD Pear2**
9	BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tableau 20 – Projection de la matrice $M_{Noyau(BitPool, BitStream)}$

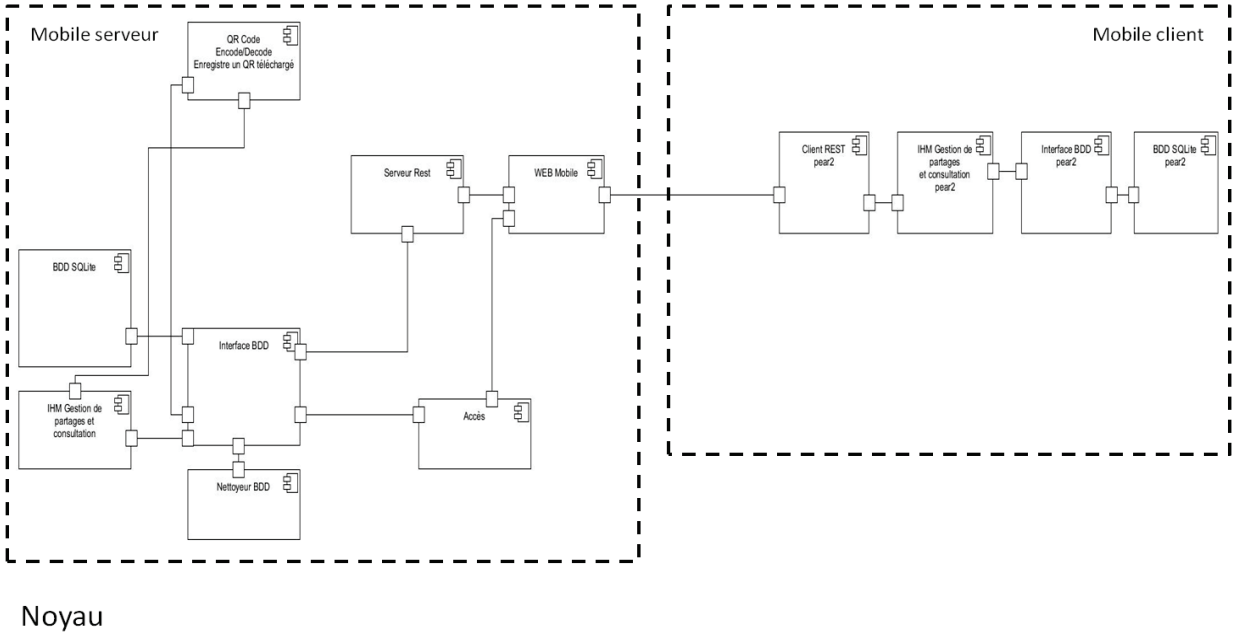
On peut maintenant choisir une fonction d'indexation commune. Les composants seront numérotés de 1 à 7 dans l'ordre de leur apparition dans la matrice projetée $C_{BitRivers}$.

Lors de l'union des chemins, on constate que les chemins entre composants similaires sont les mêmes dans les deux matrices. Il n'y a donc pas de chemins alternatifs à considérer. Voici une proposition de matrice représentant le noyau commun à BitPool, BitStream, BitRivers. Les chemins techniques entre composants applicatifs sont indiqués. Un 1 indique toujours un lien direct entre composants applicatifs.

N(BitPool,BitStream,BitRivers)		1	2	3	4	5	6	7
1	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø
2	IHM Gestion de partages et consultation	**Interface BDD**	Ø	**1**	Ø	Ø	Ø	Ø
3	QR Code	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø
4	Nettoyeur BDD	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø
5	Accès	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø
6	IHM Gestion de partages et consultation - pear2	**Client REST pear2.WEB Mobile.Serveur REST.Interface BDD**	Ø	Ø	Ø	**Client REST pear2.WEB Mobile**	Ø	**Interface BDD pear 2**
7	BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tableau 21 – Matrice de connectivité Noyau

En voici une représentation sous forme de diagramme.



Noyau

Figure 12 – Noyau

La méthode proposée pour obtenir ce résultat est globalement simple à suivre et bien balisée. Il suffit de suivre les étapes indiquées. On entrevoit ici qu'une bonne partie du travail peut être automatisé, de par le caractère formel de la méthode. Le résultat obtenu est probant : dans le cadre des applications servant de base à ce mémoire, les composants sont facilement identifiables et le résultat peut être vérifié. Je ne commente pas davantage ici ce diagramme ni la méthode pour l'obtenir. Un commentaire à ce sujet peut être trouvé en conclusion (V). Le commentaire quant à cette méthode serait trop pauvre ici. La méthode ne peut pas encore

être mise en relation avec le travail qui suit dans les prochains chapitres. Or, son apport se mesure davantage dans un cadre global.

II.3 Décomposition en noyau et variant

II.3.1 Décomposition d'un produit en noyau et variant

Soit un ensemble de produits d'une famille de produit $E = \{S_1, S_2,\ldots,S_N\}$. Le noyau commun N_E de cette famille, dont la méthode d'obtention a été vue précédemment, va permettre de décomposer chaque produit en deux parties :

- N_E

- Le restant du produit, nommé partie variante dans (4), éléments spécifiques au produit. Pour un produit S_i de E, il sera noté $(S_i - N_E)$. Sa représentation matricielle est telle que :

$(S_i - N_E)[i,j] = S_i[i,j]$, si $N_E[i,j] = Ø$. (on n'ajoute le chemin indiqué dans la matrice de connectivité du produit dans la matrice de connectivité du variant que s'il n'en existe pas dans celle du noyau entre i et j.)

$(S_i - N_E)[i,j] = Ø$ si $N_E[i,j] \neq Ø$. (Corolaire, si au moins un chemin existe déjà dans le noyau, la matrice de connectivité du variant ne peut l'indiquer. Le variant ne doit représenter que la partie variante et ne reprend pas ce qui a été inclus dans le noyau.)

On note qu'il est donc possible d'exprimer la partie variante en reprenant des composants qui existent dans la partie noyau. Toutefois il ne faut indiquer que ceux dont un chemin les relie à un composant du variant.

II.3.2 Décomposition du variant d'un produit par ensembles d'exigences.

Le variant obtenu précédemment comporte l'ensemble des composants introduits par des exigences spécifiques au S_i considéré. On pourrait simplement se contenter d'avancer qu'un produit S_i ne comporte qu'une extension au noyau, le variant. Toutefois N. Levy *et al.* (4) proposent d'adopter une granularité possiblement plus fine et de décomposer, s'il le faut, le variant en plusieurs variants justifiés par un ensemble d'exigences cohérent spécifique du produit ainsi :

De manière générale, on peut considérer que l'ensemble $Ex_S = \{ex_{S,i}\}$ des exigences spécifiques du produit S est décomposable en une union de sous-ensembles $\cup_k Ex_{S,k}$ où

chaque $Ex_{S,\,k}$ est un sous-ensemble indépendant d'exigences, auquel va correspondre une extension architecturale ajoutée à N_E.

On décompose la partie variante (S_i - N) du produit S_i sous la forme :

$S_i - N = \Sigma_j \Delta S_{i,\,j}$ où les $\Delta S_{i,\,j}$ sont les différentes composantes architecturales spécifiques du produit S_i associées aux ensembles d'exigences $Ex_{S,\,j}$

Nous pouvons écrire $S_i = N \oplus \Sigma_i \Delta S_{i,\,j}$

On suppose ici que, pour un produit donné S_i, chaque $\Delta S_{i,j}$ a au moins un nœud commun avec le noyau N_E. (4)

Il va donc s'agir de regrouper les composants optionnels en ensembles pour chaque S_i. Ces ensembles comprennent les composants dont la présence est justifiée par un sous-ensemble indépendant d'exigences. Le parallèle entre une composante architecturale et l'exigence qui l'introduit est formalisé. La ligne de produit pourra être obtenue par la composition cohérente du noyau et de certaines extensions. (4)

On introduit ici le lien entre exigence et fonction d'où il découle un ensemble de composants regroupés en variants. En effet la composition d'un produit particulier sera la réponse à des exigences bien spécifiées.

II.3.3 Application

L'architecture de référence sera composée du noyau et des variants apportés par chaque application. Il nous faut d'abord déterminer leur variant, puis effectuer une opération de décomposition sur ceux-ci.

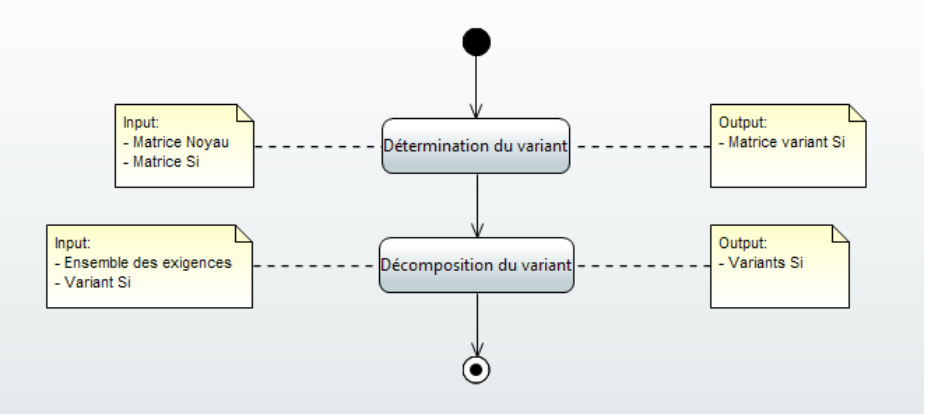


Figure 13 – Diagramme d'activité : obtention des variants.

II.3.3.1 **Détermination du variant de chaque produit**

Voici une proposition de diagramme d'activité pour la détermination du variant d'un produit S_i.

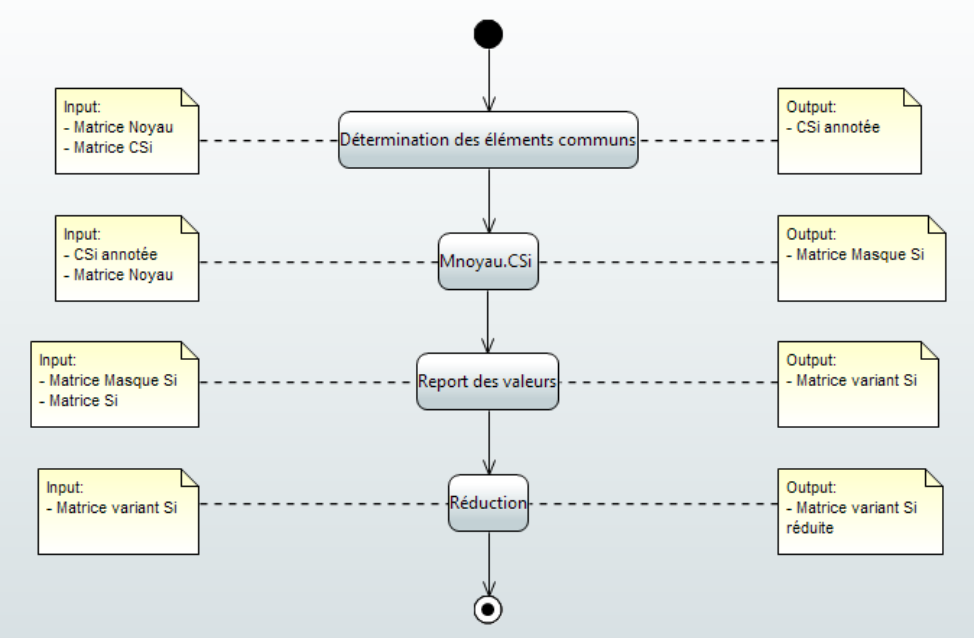


Figure 14 – Diagramme d'activité : Détermination du variant.

L'activité de détermination des éléments communs peut reprendre la méthode donnée lors de la détermination de composants communs entre deux configurations architecturales pour le noyau.

L'article introduit une définition du produit scalaire de deux représentations matricielles par $S_1.S_2 = \sum_{i,j} S_1[i,j].\ S_2[i,j]$ où $S_1[i,j].\ S_2[i,j]$ est défini par $S_1[i,j].\ S_2[i,j] = 0$ si d'un des deux $S_i[i,j] = Ø$, ou si les chemins $S_1[i,j]$ et $S_2[i,j]$ sont indépendants. $S_1[i,j].\ S_2[i,j] = 1$ sinon. L'article ajoute encore que :

- $N_E\ .\ (S_i - N_E) = 0$

- La somme $S_i = N_E + (S_i - N_E)$ notée $S_i = N_E \oplus (S_i - N_E)$ est orthogonale.

L'activité $M_{noyau}.C_{Si}$ s'en inspire. Ainsi en effectuant ce produit sur les matrices représentant N_E (M_{noyau}) et S_i (C_{Si}) on peut obtenir une matrice révélatrice d'éléments communs, un masque à appliquer sur la matrice C_{Si}.

L'activité report des valeurs, reporte les spécificités de S_i dans la matrice de connectivité du variant (M_Δ) en appliquant le masque (M_{masque}) obtenu plus tôt sur la matrice de C_{Si}. Nous suivrons cette table des valeurs :

valeur M_{masque}	valeur M_Δ
0	valeur C_{si}
1	Ø

Tableau 22 – Table des valeurs : opération report.

Pour toute valeur à 1 dans M_{masque}, remplacer par Ø dans M_Δ. Un chemin entre deux composants communs au noyau et S_i présent dans les deux représentations ne nous intéresse pas au niveau du variant. Pour toute valeur à 0, reporter la valeur présente dans la matrice C_{Si} du produit considéré.

L'activité de réduction supprime les lignes et colonnes correspondant aux composants qui ne nous intéressent plus dans le variant. Celles des composants communs avec le noyau qui :

- n'ont aucun lien avec un composant non présent dans le noyau.

- n'ont pas entre eux un chemin indépendant dans S_i.

Ces chemins sont indiqués par des Ø présents à la fois sur l'ensemble de la ligne et sur l'ensemble de la colonne du composant commun au noyau dans M_Δ.

Nous obtenons ainsi la matrice de connectivité du variant d'un produit. Voici à suivre les matrices pour chaque produit présenté dans ce mémoire.

II.3.3.1.1 **Variant BitPool**

BitPool		1	2	3	4	5	6	7	8	9	10
1	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
2	IHM Gestion de partages et consultation	**Interface BDD**	Ø	**1**	**1**	**1**	Ø	Ø	Ø	Ø	Ø
3	Hotspot Wifi	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
4	Wifi P2P	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
5	QR Code	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6	Nettoyeur BDD	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7	Accès	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
8	IHM Gestion de partages et consultation - pear2	**Client Rest P2.WEB Mobile.Serveur REST.Interface BDD**	Ø	Ø	Ø	Ø	Ø	**Client Rest P2.WEB Mobile**	Ø	**Interface BDD Pear2**	**1**
9	BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
10	Wifi P2P pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Composants communs à $C_{BitPool}$ et M_{noyau}

Tableau 23 – Détermination des éléments communs pour $C_{BitPool}$

BitPool		1	2	3	4	5	6	7	8	9	10
1	BDD SQLite	0	0	0	0	0	0	0	0	0	0
2	IHM Gestion de partages et consultation	1	0	0	0	1	0	0	0	0	0
3	Hotspot Wifi	0	0	0	0	0	0	0	0	0	0
4	Wifi P2P	0	0	0	0	0	0	0	0	0	0
5	QR Code	1	0	0	0	0	0	0	0	0	0
6	Nettoyeur BDD	1	0	0	0	0	0	0	0	0	0
7	Accès	**1**	0	0	0	0	0	0	0	0	0
8	IHM Gestion de partages et consultation - pear2	**1**	0	0	0	0	0	**1**	0	**1**	**0**
9	BDD SQLite pear2	0	0	0	0	0	0	0	0	0	0
10	Wifi P2P pear2	0	0	0	0	0	0	0	0	0	0

Composants communs à $C_{BitPool}$ et M_{noyau}

Tableau 24 – M_{masque} BitPool

BitPool		1	2	3	4	5	6	7	8	9	10
1	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
2	IHM Gestion de partages et consultation	Ø	Ø	**1**	**1**	Ø	Ø	Ø	Ø	Ø	Ø
3	Hotspot Wifi	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
4	Wifi P2P	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
5	QR Code	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6	Nettoyeur BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7	Accès	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
8	IHM Gestion de partages et consultation - pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**1**
9	BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
10	Wifi P2P pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Composants communs à $C_{BitPool}$ et M_{noyau}

Tableau 25 – $M_{\Delta BitPool}$

ΔBitPool		2	3	4	8	10
2	IHM Gestion de partages et consultation	Ø	**1**	**1**	Ø	Ø
3	Hotspot Wifi	Ø	Ø	Ø	Ø	Ø
4	Wifi P2P	Ø	Ø	Ø	Ø	Ø
8	IHM Gestion de partages et consultation - pear2	Ø	Ø	Ø	Ø	**1**
10	Wifi P2P pear2	Ø	Ø	Ø	Ø	Ø

Composants communs à $C_{BitPool}$ et M_{noyau}

Tableau 26 - $M_{\Delta BitPool}$ réduite

II.3.3.1.2 **Variant BitStream**

$C_{BitStream}$		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	IHM inscription	Ø	**Interface BDD + ~~Client REST.Interface BDD~~**	Ø	Ø	Ø	Ø	**1**	*Client REST*	Ø	Ø	*Client REST.WEB.Serveur REST Link*	*Client REST.WEB.Serveur REST Link*	**Client REST.WEB.Serveur REST Link**	Ø	Ø	Ø	Ø	Ø	Ø
2	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
3	IHM Gestion de partages et consultation	Ø	**Interface BDD + ~~Client REST.Interface BDD~~**		**1**	**1**	**1**	**1**	*Client REST*	Ø	Ø	**Client REST.WEB.Serveur REST Link**	**Client REST.WEB.Serveur REST Link**	*Client REST.WEB.Serveur REST Link*	Ø	Ø	Ø	Ø	Ø	Ø
4	UPnP	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
5	Hotspot Wifi	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6	Wifi P2P	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7	QR Code	Ø	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
8	Chiffrement/ Decriffrement Ids	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
9	Nettoyeur BDD	Ø	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
10	Accès	Ø	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
11	Distribution Idip	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
12	Inscription en BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
13	Attribution Idunique	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
14	BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
15	Nettoyeur BDD Link	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
16	IHM Gestion de partages et consultation - pear2	Ø	**Client REST Pear2.WEB Mobile.Serveur REST.Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Client REST pear2.WEB Mobile**	**Client REST Pear2.WEB.Serveur REST Link**	**Client REST Pear2.WEB.Serveur REST Link**	*Client REST Pear2.WEB.Serveur REST Link*	Ø	Ø	Ø	*Client REST Pear2*	**Interface BDD Pear2**	1
17	Chiffrement/ Decriffrement Ids pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
18	BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
19	Wifi P2P Pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Composants communs à $C_{BitStream}$ et M_{noyau}

Tableau 27 - Détermination des éléments communs pour $C_{BitStream}$

$M_{masque\ BitStream}$		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	IHM inscription	0	**0**	0	0	0	0	**0**	0	0	0	0	0	**0**	0	0	0	0	0	0
2	BDD SQLite	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	IHM Gestion de partages et consultation	0	**1**	0	**0**	**0**	**0**	**1**	0	0	0	**0**	**0**	0	0	0	0	0	0	0
4	UPnP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	Hotspot Wifi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	Wifi P2P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	QR Code	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	Chiffrement/ Decriffrement Ids	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	Nettoyeur BDD	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	Accès	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	Distribution Idip	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0
12	Inscription en BDD	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0
13	Attribution Idunique	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0
14	BDD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	Nettoyeur BDD Link	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0
16	IHM Gestion de partages et consultation - pear2	0	**1**	0	0	0	0	0	0	0	**1**	**0**	**0**	0	0	0	0	0	**1**	0
17	Chiffrement/ Decriffrement Ids pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	BDD SQLite pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	Wifi P2P Pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Composants communs à $C_{BitStream}$ et M_{noyau}

Tableau 28– M_{masque} BitStream

$M_{\Delta BitStream}$		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	IHM inscription	Ø	**Interface BDD + ~~Client REST.Interface BDD~~**	Ø	Ø	Ø	Ø	**1**	*Client REST*	Ø	Ø	*Client REST.WEB.Serveur REST Link*	*Client REST.WEB.Serveur REST Link*	**Client REST.WEB.Serveur REST Link**	Ø	Ø	Ø	Ø	Ø
2	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
3	IHM Gestion de partages et consultation	Ø	Ø	Ø	**1**	**1**	**1**	Ø	*Client REST*	Ø	Ø	**Client REST.WEB.Serveur REST Link**	**Client REST.WEB.Serveur REST Link**	*Client REST.WEB.Serveur REST Link*	Ø	Ø	Ø	Ø	Ø
4	UPnP	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
5	Hotspot Wifi	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6	Wifi P2P	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7	QR Code	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
8	Chiffrement/ Decriffrement Ids	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
9	Nettoyeur BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
10	Accès	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
11	Distribution Idip	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
12	Inscription en BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
13	Attribution Idunique	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
14	BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
15	Nettoyeur BDD Link	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
16	IHM Gestion de partages et consultation - pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Client REST Pear2.WEB.Serveur REST Link**	**Client REST Pear2.WEB.Serveur REST Link**	*Client REST Pear2.WEB.Serveur REST Link*	Ø	Ø	Ø	*Client REST Pear2*	Ø
17	Chiffrement/ Decriffrement Ids pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
18	BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
19	Wifi P2P Pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Composants communs à $C_{BitStream}$ et M_{noyau}

Tableau 29 - $M\Delta_{BitStream}$

ΔBitStream		1	2	3	4	5	6	7	8	11	12	13	14	15	16	17	19
1	IHM inscription	Ø	**Interface BDD + ~~Client REST.Interface BDD~~**	Ø	Ø	Ø	Ø	**1**	*Client REST*	*Client REST.WEB.Serveur REST Link*	*Client REST.WEB.Serveur REST Link*	**Client REST.WEB.Serveur REST Link**	Ø	Ø	Ø	Ø	Ø
2	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
3	IHM Gestion de partages et consultation	Ø	Ø	Ø	**1**	**1**	**1**	Ø	*Client REST*	**Client REST.WEB.Serveur REST Link**	**Client REST.WEB.Serveur REST Link**	*Client REST.WEB.Serveur REST Link*	Ø	Ø	Ø	Ø	Ø
4	UPnP	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
5	Hotspot Wifi	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6	Wifi P2P	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7	QR Code	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
8	Chiffrement/ Decriffrement Ids	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
11	Distribution Idip	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
12	Inscription en BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
13	Attribution Idunique	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
14	BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
15	Nettoyeur BDD Link	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
16	IHM Gestion de partages et consultation - pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Client REST Pear2.WEB.Serveur REST Link**	**Client REST Pear2.WEB.Serveur REST Link**	*Client REST Pear2.WEB.Serveur REST Link*	Ø	Ø	Ø	*Client REST Pear2*	1
17	Chiffrement/ Decriffrement Ids pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
19	Wifi P2P Pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Composants communs à $C_{BitStream}$ et M_{noyau}

Tableau 30 - $M_{\Delta BitStream}$ réduite

II.3.3.1.3 **Variant BitRivers**

$C_{BitRivers}$		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	IHM inscription	Ø	**Interface BDD + ~~Client REST.Interface BDD~~**	Ø	Ø	**1**	*Client REST*	Ø	Ø	Ø	*Client REST.WEB.Serveur REST Link*	*Client REST.WEB.Serveur REST Link*	**Client REST.WEB.Serveur REST Link**	Ø	Ø	Ø	Ø	Ø	Ø
2	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
3	IHM Gestion de partages et consultation	Ø	**Interface BDD + ~~Client REST.Interface BDD~~**	Ø	**1**	**1**	*Client REST*	Ø	Ø	**Client REST**	**Client REST.WEB.Serveur REST Link**	**Client REST.WEB.Serveur REST Link**	*Client REST.WEB.Serveur REST Link*	Ø	Ø	Ø	Ø	Ø	Ø
4	UPnP	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
5	QR Code		**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6	Chiffrement/ Decriffrement Ids	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7	Nettoyeur BDD	Ø	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
8	Accès	Ø	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
9	BTSync	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
10	Distribution Idip	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
11	Inscription en BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
12	Attribution Idunique	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
13	BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø		Ø	Ø	Ø	Ø	Ø
14	Nettoyeur BDD Link	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
15	IHM Gestion de partages et consultation - pear2	Ø	**Client REST pear2.WEB Mobile.Serveur REST.Interface BDD**	Ø	Ø	Ø	Ø	Ø	**Client REST pear2.WEB Mobile**	Ø	**Client REST pear2.WEB.Serveur REST Link**	**Client REST pear2.WEB.Serveur REST Link**	*Client REST pear2.WEB.Serveur REST Link*	Ø	Ø	Ø	*Client REST pear 2*	**Interface BDD pear 2 + ~~Client REST pear2.Interface BDD pear2~~**	**Client REST Pear2**
16	Chiffrement/ Decriffrement Ids pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
17	BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
18	BTSync pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**1**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Composants communs à $C_{BitRivers}$ et M_{noyau}

Tableau 31 - Détermination des éléments communs pour $C_{BitRivers}$

$M_{masque\ BitRivers}$		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	IHM inscription	0	**0**	0	0	**0**	**0**	0	0	0	**0**	**0**	**0**	0	0	0	0	0	0
2	BDD SQLite	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	IHM Gestion de partages et consultation	0	**1**	0	**0**	**1**	**0**	0	0	**0**	**0**	**0**	**0**	0	0	0	0	0	0
4	UPnP	0	Ø	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	QR Code	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	Chiffrement/ Decriffrement Ids	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	Nettoyeur BDD	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	Accès	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	BTSync	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	Distribution Idip	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0
11	Inscription en BDD	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0
12	Attribution Idunique	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0
13	BDD	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0
14	Nettoyeur BDD Link	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0
15	IHM Gestion de partages et consultation - pear2	0	**1**	0	0	0	0	0	**1**	0	**0**	**0**	0	0	0	0	0	**1**	**0**
16	Chiffrement/ Decriffrement Ids pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	BDD SQLite pear2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	BTSync pear2	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0

Composants communs à $C_{BitRivers}$ et M_{noyau}

Tableau 32 – $C_{masque\ BitRivers}$

$M_{\Delta BitRivers}$		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	IHM inscription	Ø	**Interface BDD + ~~Client REST.Interface BDD~~**	Ø	Ø	**1**	*Client REST*	Ø	Ø	Ø	*Client REST.WEB.Serveur REST Link*	*Client REST.WEB.Serveur REST Link*	**Client REST.WEB.Serveur REST Link**	Ø	Ø	Ø	Ø	Ø	Ø
2	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
3	IHM Gestion de partages et consultation	Ø	Ø	Ø	**1**	Ø	*Client REST*	Ø	Ø	**Client REST**	**Client REST.WEB.Serveur REST Link**	**Client REST.WEB.Serveur REST Link**	*Client REST.WEB.Serveur REST Link*	Ø	Ø	Ø	Ø	Ø	Ø
4	UPnP	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
5	QR Code	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6	Chiffrement/ Decriffrement Ids	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7	Nettoyeur BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
8	Accès	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
9	BTSync	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
10	Distribution Idip	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
11	Inscription en BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
12	Attribution Idunique	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
13	BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø		Ø	Ø	Ø	Ø	Ø
14	Nettoyeur BDD Link	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø	Ø
15	IHM Gestion de partages et consultation - pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Client REST pear2.WEB.Serveur REST Link**	**Client REST pear2.WEB.Serveur REST Link**	*Client REST pear2.WEB.Serveur REST Link*	Ø	Ø	Ø	*Client REST pear 2*	Ø	**Client REST Pear2**
16	Chiffrement/ Decriffrement Ids pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
17	BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
18	BTSync pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**1**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Composants communs à $C_{BitRivers}$ et M_{noyau}

Tableau 33 - $M_{\Delta BitRivers}$

$M_{\Delta BitRivers}$		1	2	3	4	5	6	9	10	11	12	13	14	15	16	18
1	IHM inscription	Ø	**Interface BDD +** ~~**Client REST.Interface BDD**~~	Ø	Ø	**1**	*Client REST*	Ø	*Client REST.WEB.Serveur REST Link*	*Client REST.WEB.Serveur REST Link*	**Client REST.WEB.Serveur REST Link**	Ø	Ø	Ø	Ø	Ø
2	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
3	IHM Gestion de partages et consultation	Ø	Ø	Ø	**1**	Ø	*Client REST*	**Client REST**	**Client REST.WEB.Serveur REST Link**	**Client REST.WEB.Serveur REST Link**	*Client REST.WEB.Serveur REST Link*	Ø	Ø	Ø	Ø	Ø
4	UPnP	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
5	QR Code	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6	Chiffrement/ Decriffrement Ids	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
9	BTSync	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
10	Distribution Idip	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
11	Inscription en BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
12	Attribution Idunique	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
13	BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø		Ø	Ø	Ø	Ø
14	Nettoyeur BDD Link	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
15	IHM Gestion de partages et consultation - pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Client REST pear2.WEB.Serveur REST Link**	**Client REST pear2.WEB.Serveur REST Link**	*Client REST pear2.WEB.Serveur REST Link*	Ø	Ø	Ø	*Client REST pear 2*	**Client REST Pear2**
16	Chiffrement/ Decriffrement Ids pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
18	BTSync pear2	Ø	Ø	Ø	Ø	Ø	Ø	**1**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Composants communs à $C_{BitRivers}$ et M_{noyau}

Tableau 34 - $M_{\Delta BitRivers}$ réduite

II.3.3.2 **Décomposition du variant de chaque produit**

Il s'agit maintenant de décomposer le variant de chaque application en ensembles ou classes de composants. Ils sont introduits par des exigences spécifiques cohérentes et indépendantes.

Ici encore, la méthode repose sur la connaissance du domaine de l'architecte. Il est à noter que cette notion de classe de composants se rapproche de la notion de feature dans le cadre des approches de feature location (localisation de caractéristiques). Ce pan de recherche constate que bien qu'il soit très souvent spécifié les caractéristiques et fonctionnalités d'un produit logiciel dans la documentation qui l'accompagne, la relation entre ces fonctionnalités et leur implémentation correspondante ne l'est qu'occasionnellement. L'objectif principal des méthodes de feature location est de retrouver ce lien. Toutefois elles ne s'attachent pas spécifiquement à la recherche de caractéristiques dans le cadre des SPL.

Rajlich et Chen (7) expliquent qu'une caractéristique est composée de trois éléments : un nom, une intention et une extension. Le nom identifie la caractéristique, l'intention explique ce que signifie ou fait la caractéristique, et son extension correspond à l'ensemble d'artefacts logiciels qui l'implémentent. D'un point de vue un peu réducteur et pour simplifier ici, nous considérerons qu'il s'agit des composants logiciels. La localisation (location) est ce qui lie une intention à une extension (intention→ extension). Les approches de localisation de caractéristiques cherchent donc à établir un lien de traçabilité entre une caractéristique d'intérêt, spécifiée par l'utilisateur et les artéfacts qui l'implémentent. Ceci cadre bien avec la représentation matricielle réduite proposée par N. Levy *et al.* (4) que nous utilisons ici. Seules les connexions entre composants applicatifs sont étudiées, en faisant abstraction de la méthode de communication. La méthode proposée pour décomposer le variant en variants établit aussi ce lien entre des exigences que l'on peut apparenter à une intension, et la classe de composants, que l'on peut assimiler à une extension. Voici pour chaque produit, les variants associés.

II.3.3.2.1 **Variants BitPool**

Lorsque l'on observe le tableau 25, la matrice $M_{\Delta BitPool}$ réduite ne laisse que peu de composants dans le variant. HotSpot Wifi, Wifi P2P et Wi-Fi P2P pear2 et les IHM de

chaque mobile. Cette application est une application de partage de fichiers entre appareils mobiles sur un réseau local. Le seul rôle que remplissent ces composants ensemble est de permettre la création d'un réseau Wi-Fi pour un échange dans un réseau local. Aucune autre exigence exprimée au paragraphe I.1.1 n'est implémentée par ces composants. Il y a donc qu'une seule classe et un seul variant pour BitPool.

II.3.3.2.2 **Variants BitStream**

Lorsque l'on observe le tableau 29, la matrice $M_{\Delta BitStream}$ réduite laisse davantage de composants dans le variant. On retrouve les composants HotSpot Wifi, Wifi P2P et Wi-Fi P2P pear2 et les IHM de chaque mobile. Cette application est une application de partage de fichiers entre appareils mobiles dans et hors réseau local.

On retrouve la classe de composant permettant le partage sur un réseau local. Elle sera isolée. Une autre classe de composant peut être isolée par l'exigence de connexion à distance, lorsque deux mobiles ne sont plus dans le même réseau. Les composants IHM inscription (pear 1 et 2), UPnP, Chiffrement/Déchiffrement des IDs, Distribution ID IP, inscription en BDD, attribution IDunique, Nettoyeur BDD Link et Chiffrement/Déchiffrement des IDs pear2 implémentent spécifiquement cette caractéristique. Les composants IHM gestion de partage et consultation sont à nouveau à inclure dans cette classe de composants. Ils sont ce que certaines approches de feature location qui utilisent l'algorithme de classement des pages web HITS (Hyper-link Induced Topic Search) appelleraient un Hub. Ce composant est le point de départ de plusieurs chemins techniques. Ils peuvent être reliés à plusieurs ensembles de caractéristiques et par conséquent à plusieurs classes de l'application. L'application BitStream comporte deux variants.

II.3.3.2.3 **Variants BitRivers**

L'application BitRivers permet le partage entre mobiles hors réseau local, et potentiellement avec un grand nombre de pairs. Ces deux exigences se retrouvent donc en deux classes distinctes, deux variants. La connexion hors réseau est assurée par les mêmes composants que BitStream. Le partage en grand nombre implique l'implémentation par les composants BitTorrent Sync API mode (pear 1 et 2). L'IHM gestion/consultation est à

nouveau incluse. Elle est centrale et est le point de départ de beaucoup de chemins techniques.

II.4 Expression de l'architecture de référence

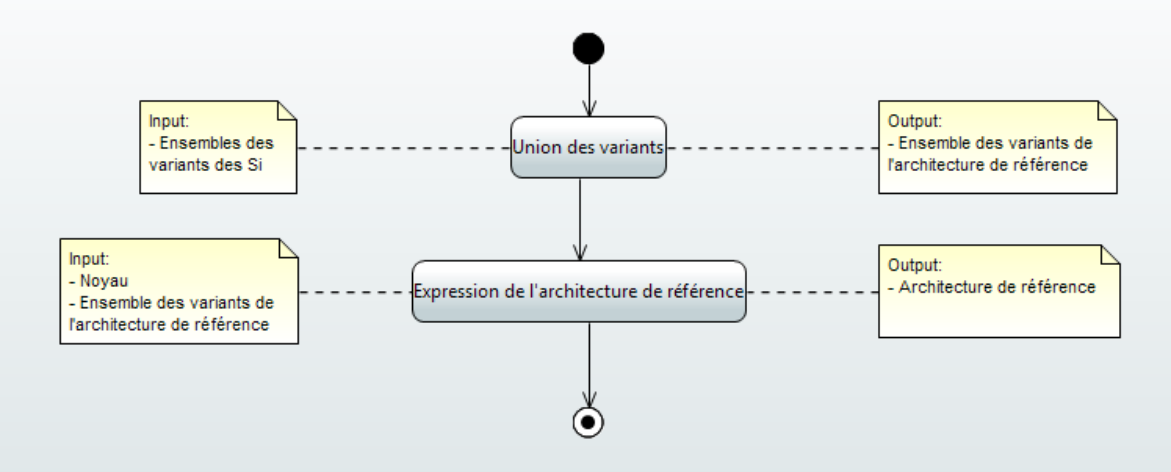


Figure 15 – Diagramme d'activité : Expression de l'architecture de référence.

L'architecture de référence servira de base à la dérivation de futurs produits logiciels. Ils seront composés du noyau et de divers variants. La décomposition des trois produits a montré qu'il existait des variants identiques entre BitPool et BitStream puis entre BitStream et BitRivers. Reporter ces variants plusieurs fois dans l'expression de l'architecture de référence n'apporte aucun avantage quant au but recherché.

II.4.1 Union des variants des produits logiciels

Si on considère chaque ensemble de variants $Ev_{Si} = \{V_1, \ldots, V_N\}$ de chaque S_i composé des variants V_k, l'ensemble des variants de l'architecture de référence $Ev_A = \{Ev_{s1} \cup Ev_{s2} \cup, \ldots, Ev_{sN}\}$.

Ainsi l'appartenance à telle ou telle application d'un variant n'est plus pertinente. Un variant sera davantage rattaché à la caractéristique fonctionnelle ou non fonctionnelle qu'il implémente. On ne retient ainsi plus que trois variants sur les cinq qu'apportent les trois applications.

II.4.2 Expression de l'architecture de référence

Si l'on reprend la famille de produit E = {BitPool, BitRivers, BitStream}, son noyau commun N_E et l'expression d'un système S_i sous la forme d'une somme $S_i = N_E \oplus (S_i - N_E)$. Tout comme il est possible de décomposer chaque S_i de E en deux parties NE et $(S_i - N_E)$, il est possible de décomposer l'architecture de référence en plusieurs parties, noyau et variants. L'architecture de référence de E sera définie par l'expression matricielle C_F =

N(BitPool,BitStream,BitRivers)	1	2	3	4	5	6	7
1 BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø
2 IHM Gestion de partages et consultation	**Interface BDD**	Ø	**1**	Ø	Ø	Ø	Ø
3 QR Code	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø
4 Nettoyeur BDD	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø
5 Accès	**Interface BDD**	Ø	Ø	Ø	Ø	Ø	Ø
6 IHM Gestion de partages et consultation - pear2	**Client REST pear2.WEB Mobile.Serveur REST.Interface BDD**	Ø	Ø	Ø	**Client REST pear2.WEB Mobile**	Ø	**Interface BDD pear 2**
7 BDD SQLite pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø

⊕ **VConnexionLocale** ✕

ΔConnexionLocale	1	2	3	4	5
4 IHM Gestion de partages et consultation	Ø	**1**	**1**	Ø	Ø
2 Hotspot Wifi	Ø	Ø	Ø	Ø	Ø
3 Wifi P2P	Ø	Ø	Ø	Ø	Ø
4 IHM Gestion de partages et consultation - pear2	Ø	Ø	Ø	Ø	**1**
5 Wifi P2P pear2	Ø	Ø	Ø	Ø	Ø

⊕ **VPartageGrandNombre** ✕

ΔPartageGrandNombre	1	2	3	4
1 IHM Gestion de partages et consultation	Ø	**Client REST**	Ø	Ø
2 BTSync	Ø	Ø	Ø	Ø
3 IHM Gestion de partages et consultation - pear2	Ø	Ø	Ø	**Client REST Pear2**
4 BTSync pear2	Ø	**1**	Ø	Ø

⊕ **VUtilisationNommade** ✕

ΔUtilisationNommade		1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	IHM inscription	Ø	**Interface BDD + ~~Client REST.Interface BDD~~**	Ø	Ø	**1**	*Client REST*	*Client REST.WEB.Serveur REST Link*	*Client REST.WEB.Serveur REST Link*	**Client REST.WEB.Serveur REST Link**	Ø	Ø	Ø	Ø	Ø
2	BDD SQLite	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
3	IHM Gestion de partages et consultation	Ø	Ø	Ø	**1**	Ø	*Client REST*	**Client REST.WEB.Serveur REST Link**	**Client REST.WEB.Serveur REST Link**	*Client REST.WEB.Serveur REST Link*	Ø	Ø	Ø	Ø	Ø
4	UPnP	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
5	QR Code	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6	Chiffrement/ Decriffrement Ids	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7	Distribution Idip	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
8	Inscription en BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
9	Attribution Idunique	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
10	BDD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
11	Nettoyeur BDD Link	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	**Interface BDD Link**	Ø	Ø	Ø	Ø
12	IHM Gestion de partages et consultation - pear2	Ø	Ø	Ø	Ø	Ø	Ø	**Client REST Pear2.WEB.Serveur REST Link**	**Client REST Pear2.WEB.Serveur REST Link**	*Client REST Pear2.WEB.Serveur REST Link*	Ø	Ø	Ø	*Client REST Pear2*	1
13	Chiffrement/ Decriffrement Ids pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
14	Wifi P2P Pear2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

III Qualité

III.1 Modèle de qualité logiciel

III.1.1 Préambule

D'une manière générale, lorsque l'on désigne un logiciel de qualité, on parle d'un « bon » logiciel. Ceci n'est évidemment pas une définition satisfaisante, des questions se posent : est-ce que la qualité est contenue objectivement dans le logiciel ? Ou est-ce que la qualité dépend de son utilité pour un utilisateur spécifique ? La définition est si peu aisée que L'ISO/IEC/IEEE définissent la qualité[2] d'au moins six manières différentes:

- "The degree to which a system, component or process meets specified requirements"

- "The ability of a product, service, system, component or process to meet customer or user needs, expectations or requirements"

- "The totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs"

- "Conformity to user expectations, conformity to user requirements, customer satisfaction, reliability and level of defects present"

- "The degree to which a set of inherent characteristics fulfils requirements"

- "The degree to which a system, component or process meets customer or user needs or expectations"

Ces six définitions reflètent les subtiles différences de significations que peut avoir la qualité logicielle. Quoiqu'il en soit, dans toutes les définitions on trouve la notion d'exigences ou de besoins devant être satisfaits. Mais de qui doivent émaner ces besoins ? On sait qu'un besoin peut être exprimé par un utilisateur ou un client. La qualité s'obtient-elle en satisfaisant les exigences de la personne utilisant le produit ou celles de la personne

[2] Dans le standard ISO/IEC/IEEE 24765 de 2010 : Systems and software engineering – vocabulary

qui paie pour lui ? Comment s'assurer de satisfaire tous les besoins ? Les besoins à satisfaire peuvent être exprimés implicitement. Même s'il on satisfait toutes les exigences explicites, un produit logiciel peut ne pas être de qualité puisqu'il ne répond pas aux attentes de l'utilisateur ou du client. De fait, même avec une analyse des besoins très élaborée, il est toujours possible de ne pas aborder une exigence très importante dans une spécification. Ceci nous amène à nouveau à nous demander comment s'y prendre pour obtenir un produit logiciel de qualité.

Pour mieux comprendre les besoins (des utilisateurs ou des clients) et les attributs des produits logiciels, différents aspects de la qualité que l'on doit examiner ont été définis au cours des recherches faites sur le sujet. Ces propositions sont formalisées à l'aide de ce qu'on appelle des modèles de qualité. Ces outils sont proposés pour aider l'architecte logiciel et l'analyste des besoins à aboutir à un produit de qualité, sans omettre aucune des exigences normalement attendues. Les modèles sont utilisés dans diverses phases de l'élaboration du produit logiciel. Pendant l'analyse des besoins, ils permettent à toutes les parties prenantes dans l'élaboration du produit d'utiliser des notions précisément définies. Avec ce vocabulaire commun, propice à une meilleure compréhension des attentes, ils aident les parties à mieux se mettre d'accord sur ce que signifie la qualité. Pendant l'implémentation, ils peuvent fournir des lignes conductrices afin de parvenir à une qualité logicielle accrue.

Parmi ces modèles, se trouve la norme ISO 25010 présentée en section III.1.2. Ce framework ne s'attache pas spécifiquement au cas de la qualité des lignes de produits et ne présente pas de guide d'utilisation. Il y est d'ailleurs mentionné que ses apports ne peuvent pas être appliqués à l'ensemble des produits logiciels. Il faut donc l'adapter, sans aide.

N. Levy *et al.* (5) répondent à ces deux difficultés en proposant un guide d'utilisation de ce modèle afin d'aboutir à la définition d'un modèle de qualité dans le cadre des familles de produit. Ils expliquent que la qualité est une préoccupation qui doit être prise en compte le plus tôt possible dans l'élaboration d'un système logiciel. Or, la conception de l'architecture d'un système logiciel est la première étape de sa réalisation. Il faut donc se soucier de la qualité du futur système, dès la conception de son architecture.

Nous avons en première partie commencé l'élaboration d'une architecture de référence qui nous permettra d'obtenir des architectures de systèmes individuels par une opération de dérivation. Cette opération sera abordée plus tard. En effet, il ne serait pas judicieux de reporter l'exigence de qualité au moment de la conception des systèmes

logiciels particuliers, après dérivation. Il faudrait alors repenser la qualité logicielle à chaque fois. Cela serait contre-productif et n'irait pas dans le sens des avantages apportés par l'ingénierie dirigée par les lignes de produits (temps et coûts de développement amoindris). C'est pourquoi l'exigence de qualité est considérée ici, immédiatement après l'obtention de l'architecture de référence.

L'objectif de la démarche proposée dans (5) est de capitaliser les connaissances acquises dans le développement de systèmes d'un même domaine fonctionnel[3] afin

---

[3] La notion de domaine fonctionnel s'applique aux systèmes qui remplissent des fonctions proches dans un domaine donné. La notion de domaine fonctionnelle est plus restrictive que la notion de domaine (le domaine fonctionnel « Gestion de la paie » est du domaine « Ressources Humaines »). Le domaine fonctionnel des applications servant de support à ce mémoire serait « partage de fichiers entre appareils mobiles ».

Le vocabulaire utilisé dans (4) et (5) entre « en collision ». Le premier emploie les termes « architecture de référence » pour l'expression de l'architecture du noyau et des variants que nous avons définis plus tôt. Le second fait référence à l'architecture prenant en compte les fonctionnalités et les exigences non fonctionnelles associées. Dans (5), l'architecture de référence peut être définie à deux niveaux de considération : au niveau domaine et au niveau système particulier. Le niveau domaine traite des exigences communes à l'ensemble des applications d'un domaine fonctionnel. Le niveau système reprend ces exigences parfois génériques en les adaptant s'il le faut à son contexte d'utilisation et ajoute ses propres exigences.

Il n'y a pas par définition une correspondance entre architecture noyau et architecture de référence de domaine. De même, il n'y a pas une correspondance entre architecture des variants et architecture de référence d'un système particulier. Nous n'évoluons pas au même niveau d'abstraction. Au niveau système le modèle de qualité spécifiera précisément les attributs et leur valeur. Ce n'est pas forcément le cas au niveau des variants dans (4), qui selon moi peuvent faire partie de la spécification d'une architecture de référence d'un domaine dans (5). Il y aurait dans cette architecture de référence domaine un invariant fonctionnel (le noyau) et des variants fonctionnels (les variants de (4)). Les architectures obtenues après dérivation devront être des architectures de référence système au sens de (5).

Pour ne plus risquer de confusion, les termes « architecture de qualité » seront utilisés en lieu et place des termes « architecture de référence » avec la signification qu'ils ont dans (5).

d'atteindre une architecture de qualité. N. Levy *et al.* (5), constatent que l'architecte logiciel, dans l'élaboration de l'architecture d'un système logiciel, réutilise « de manière plus ou moins explicite une démarche, des développements existants, des systèmes proches et des composants ». Ils proposent de formaliser cette réutilisation davantage et nous donnent les lignes directrices de la création d'architectures de référence de qualité.

III.1.2 La norme ISO 25010

La norme ISO 25010 fait partie de ces normes qui décomposent la qualité de manière hiérarchique en plusieurs caractéristiques et sous caractéristiques jusqu'à un niveau mesurable pour permettre d'évaluer la qualité. La norme parle de caractéristiques et de sous caractéristiques de qualité, puis d'attribut pour ce qui est mesurable. L'idée est de diviser le concept complexe et abstrait de qualité logicielle en plusieurs ensembles plus abordables.

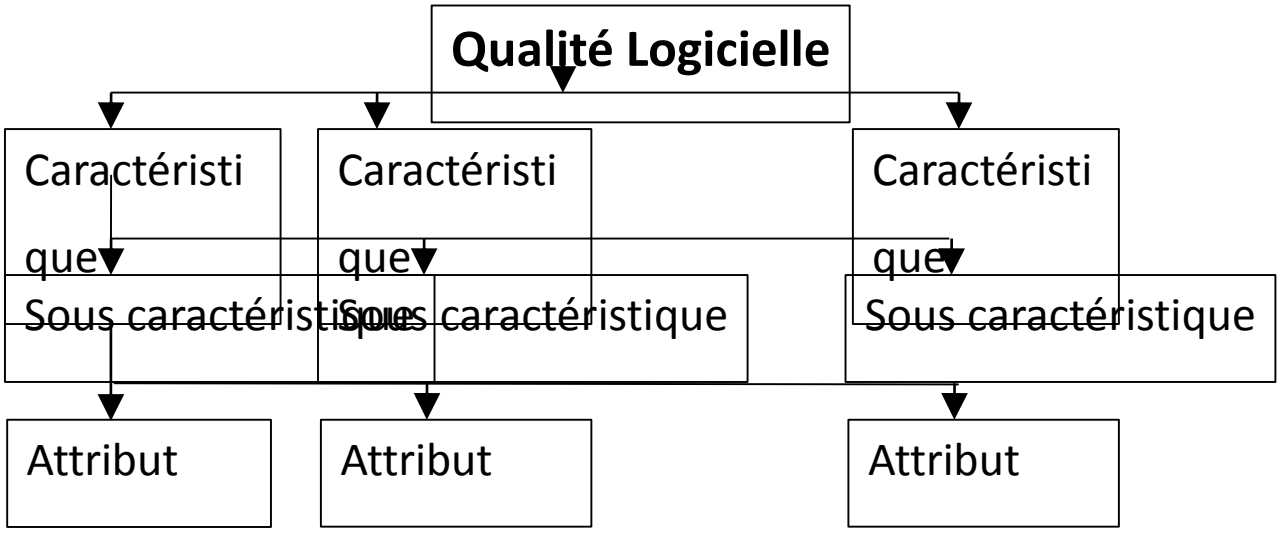


Figure 16 – Structure du modèle de qualité ISO 25010

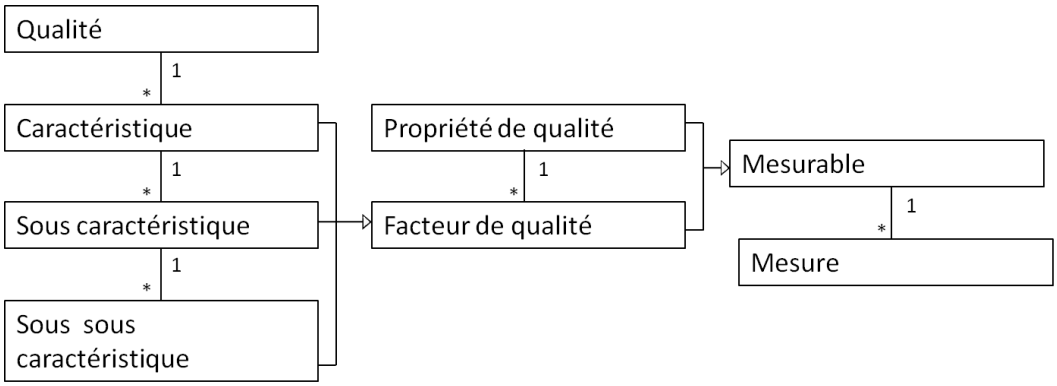


Figure 17 – Méta-modèle implicite ISO 25010

Dans ce méta-modèle, toutes les caractéristiques et sous-caractéristiques sont appelées « Facteur de qualité » qui peuvent éventuellement être mesurables par une ou plusieurs mesures. Si ce n'est pas le cas, les propriétés de qualité sont utilisées pour garantir le facteur de qualité. Les termes facteur de qualité et mesurable n'existent pas dans la norme. Ils sont là pour la bonne structuration du méta-modèle.

L'avancée faite dans la définition de modèles de qualité a mené à la mise en avant de facteurs de qualité bien connus. La norme introduit huit caractéristiques principales, décomposées en plusieurs caractéristiques qu'elle définit. Les caractéristiques décrivent les comportements attendus du système et la manière de mesurer l'adéquation de l'architecture. Le modèle de qualité ISO 25010 devient pour l'ingénieur logiciel une sorte de check-list avec laquelle il pourra s'assurer qu'il n'oublie aucun aspect de la qualité logicielle dans son produit. Toutefois, certaines caractéristiques sont difficiles à évaluer, comme la confidentialité ou l'intégrité, car abstraites et difficilement quantifiables. La norme met l'accent sur le fait que l'ensemble des aspects qu'elle présente n'est pas pertinent pour tous les types de logiciels possibles. La bonne adaptation du modèle de qualité au cas qui l'intéresse repose sur l'expérience et… la qualité de l'architecte logiciel.

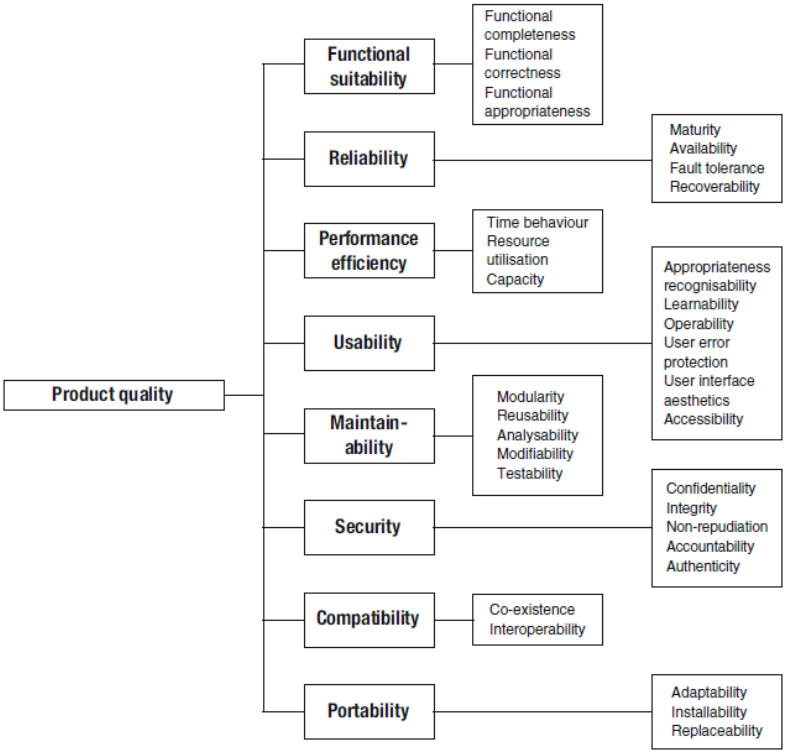


Figure 18 – Modèle de qualité ISO 25010

III.1.3 Démarche

P. O'Leary *et al.* (8), indiquent les principaux problèmes rencontrés lors de la phase de dérivation d'un produit logiciel dans une famille de produits et indiquent les mesures à prendre pour les éviter pendant une phase qu'ils appellent travail de pré-dérivation. Ils insistent notamment sur la gestion des exigences attendues. L'architecte logiciel doit comprendre les attentes de l'utilisateur (ou client), les traduire en caractéristiques du produit claires et s'assurer qu'elles sont effectivement implémentées dans le produit logiciel. Il faut donc arriver à établir un lien de traçabilité entre les exigences explicites ou implicites exprimées et les éléments implémentés ainsi que la manière dont ils répondent à ces exigences. C'est ce que propose de faire la méthode décrite ci-après.

La construction d'une architecture passe par l'analyse des besoins puis par sa conception qui découle de cette analyse. Ces deux étapes mènent à la production de deux modèles de qualité : un modèle de qualité fonctionnel décrivant les besoins du futur système lors de l'analyse des besoins (qui requière une connaissance liée au domaine que couvre le futur système), et, un modèle de qualité de l'architecture vérifiant que chaque critère de qualité sera bien implémenté dans le futur système. C'est pourquoi la démarche propose deux ensembles d'activités menées selon deux branches parallèles.

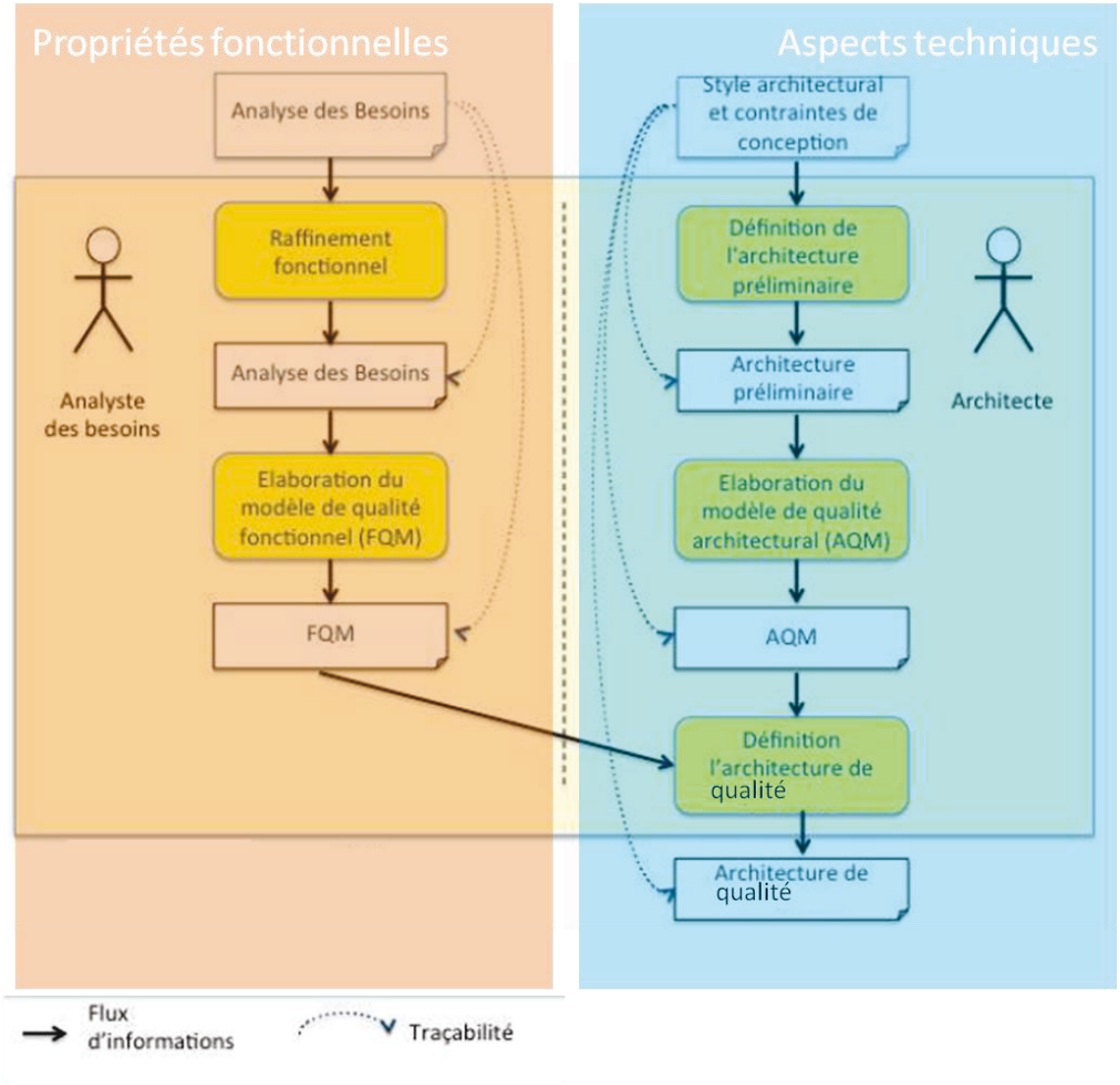


Figure 19 – Ensembles d'activités dans l'élaboration du modèle de qualité de l'architecture de référence.

Dans la pratique des interactions entre les deux branches sont possibles. On observe sur le schéma un lien de traçabilité entre les exigences formulées au préalable et les exigences de qualité des différents modèles.

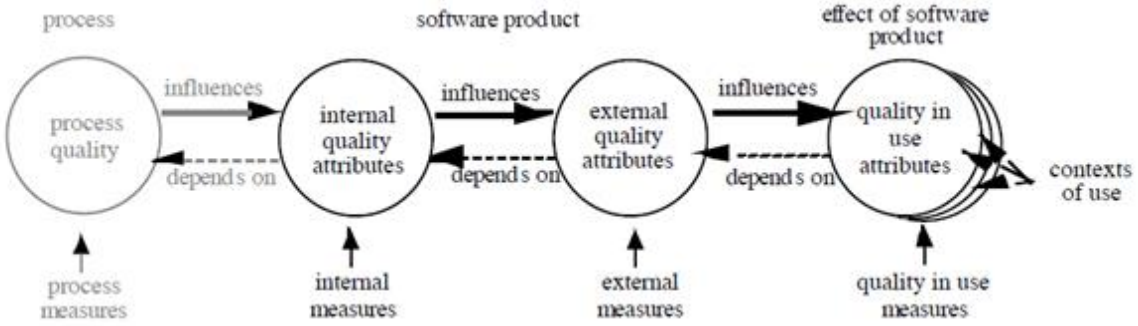
Les architectures existantes nous ont permis de créer une première architecture de référence. C'est l'étape à laquelle nous sommes à la fin de la partie II. Nous allons confronter cette architecture à la norme de qualité ISO 25010 en suivant la méthode décrite dans (5) mais en l'adaptant à notre situation. En pratique, cela permettrait à d'éventuels nouveaux utilisateurs de garantir la qualité de leur architecture de référence précédemment établie. Cette amélioration aura une répercussion positive sur les nouveaux produits dérivés de la famille de produit. Nous considérerons la qualité du noyau dans un premier temps (III.2, III.3) puis celle des variants (III.4).

III.2 Modèle de qualité du noyau

Bien qu'une première architecture de domaine existe, nous allons énumérer clairement les exigences fonctionnelles et non fonctionnelles ayant conduit à la création des divers produits de la famille en insistant sur les éléments communs constituant le noyau de l'architecture de référence de la famille de produits. Cette analyse des besoins permettra de déterminer le modèle de qualité du noyau (III.2.1,III.2.2). Par raffinement fonctionnel, nous obtiendrons le FQM (Functionnal Quality Model, modèle de qualité fonctionnel III.2.3) correspondant à notre architecture noyau préliminaire (III.2.4). Ensuite nous élaborerons l'AQM (Architecture Quality Model, modèle de qualité architectural, III.2.5). Aidé de ces éléments nous pourrons déterminer une architecture de domaine[4] de qualité pour le noyau (III.3)

III.2.1 Analyse des besoins et classification

L'utilisateur du produit évalue la qualité du produit par l'utilisation qu'il en fait (contrairement au client, qui peut ne pas être l'utilisateur et qui mesure la qualité au travers de la description qu'il a du produit). C'est lui qu'il faut satisfaire si l'on souhaite obtenir un produit de qualité. ISO définit plusieurs vues de la qualité logicielle : la qualité interne (qui s'attarde sur la qualité du code source, de la documentation), la qualité externe (qui traite de la qualité d'exécution du produit, avec des critères techniques de validation du produit) et la qualité d'utilisation (qui évalue la bonne réponse aux besoins de l'utilisateur dans son environnement de travail). Ces trois vues s'influencent comme le montre la figure suivante.



[4] N. Levy *et al.* définissent l'architecture de domaine ainsi :

« L'architecture de domaine est l'architecture d'un système « idéal » (abstrait, car en général non réalisé) qui satisferait l'ensemble des exigences communes à la classe des systèmes considéré. »

Figure 20 – Vues de la qualité logicielle selon ISO.

Il existe une dernière vue de la qualité logicielle : le processus de développement logiciel. Il influence la qualité du futur logiciel. On observe une double relation entre ces différentes vues. La bonne qualité du « process quality » influencera la bonne qualité « in use ». D'un autre côté, la bonne définition des exigences « in use » contribuera à identifier et définir les exigences de qualité du « process quality ».

L'analyse des besoins décrira ce que le système doit réaliser afin que nous puissions élaborer un modèle de qualité de l'architecture qui décrira comment l'architecture doit être conçue afin de répondre à l'ensemble des besoins exprimés. Avant d'arrêter notre architecture logicielle, il est très important de s'assurer qu'elle réponde à tous les besoins (fonctionnels et non fonctionnels). Il faut maintenant les lister.

III.2.1.1 **Champ d'analyse des besoins**

N. Levy *et al* (5) décrivent la démarche de construction d'une architecture de qualité ainsi :

« La démarche de construction d'une architecture de domaine prend pour base des exigences système (*system requirements*), qui comprennent :

– des exigences fonctionnelles propres au domaine, reflétant les fonctionnalités à fournir par le futur système ;

– des exigences non fonctionnelles (dont les « exigences de qualité »), portant sur la manière dont sont fournies ces fonctionnalités […] ;

– des exigences de conception, ensemble des contraintes que l'on impose, pour toute raison que ce soit, au futur système, en termes de conception, de moyens de réalisation, etc. […]

Les besoins fonctionnels et non fonctionnels sont considérés comme des propriétés inhérentes au logiciel qui déterminent sa qualité. » (5). Voici les fonctionnalités attendues pour les divers produits de la famille de produits. Pour plus de lisibilité dans les prochains développements, ces exigences vont être classifiées en exigences fonctionnelles et non fonctionnelles. De plus un indice sera associé à chacun d'entre eux. Il sera composé comme suit : les exigences non fonctionnelles seront identifiées par la lettre « a » suivie d'un point et d'un chiffre pour les différencier entre elles (a.x). De la même manière, les exigences fonctionnelles par un b suivi d'un point et d'un chiffre (b.x, où x est un chiffre).

L’objectif étant d’étudier les besoins ayant un impact sur le développement de l’architecture de qualité, seuls ceux-ci seront retenus. On ne considérera ici que les besoins communs à tous les systèmes de la famille de produits que nous élaborons.

III.2.1.2 **Application**

Voici donc un tableau reprenant la liste des exigences communes à BitPool, BitStream et BitRivers. Ce sont les exigences du noyau de l’architecture de référence. Ce tableau indique l’indice assigné à chaque exigence et une valeur binaire pour indiquer si cette exigence influence la conception de l’architecture. Les exigences sont classées en exigences fonctionnelles et non fonctionnelles.

indice	Exigence	Impacte sur l'architecture	Commentaire
a.1	Le système devra permettre de partager des données en économisant les éventuelles ressources serveurs	1	Nettoyeur de bases de données selon critères
a.2	Le système devra permettre de partager du contenu de manière décentralisée : pour que les données restent celles des utilisateurs	0	Absence de base de données pour stocker ces données
a.3	Le système ne devra pas permettre à une tierce personne de savoir ce qui est échangé par qui	1	Aucun suivi côté serveur de liaison
a.4	Répondre aux critères Android en temps de réponse	0	
a.5	Le système doit garantir une utilisation agréable et simple	1	Composants IHM + Respecter une charte visuelle
a.6	Le système doit être ouvert à d'éventuelles évolutions quant aux environnements des clients participant aux échanges	1	standards adoptés en terme de serveurs
a.7	Le système doit garantir une communication possible	1	redondance de composants de communicatons
a.8	Le système doit garantir la sécurité des données des utilisateurs	1	présence module accès + absence de données hebergées
a.9	Le systèle doit se protéger contre les échanges illégaux	0	vocabulaire. Charte de moralité
a.10	Le système doit être utilisable par la majorité des appareils	0	cf liste des release Android, et dév pour la majorité + langue + taille des écrans : adapter aux écrans
a.11	Le système doit garantir aux utilisateurs qu'ils réaliseront chaque action en un temps donné	0	charte des temps
a.12	Le système doit permettre à un certain de pairs d'être impliqué dans l'échange d'un fichier simultanément (multisource ou non)	1	à définir au niveau des systèmes particuliers
a.13	Le système doit être adapté au nombre de personnes avec lesquelles on envisage d'échanger (cercle restreint, élargi, sans limite à priori)	1	à définir au niveau systèmes particuliers
a.14	Le système devra être économe en terme d'énergie sur les systèmes Android	1	format des données d'échange
a.15	Le système devra s'adapter en fonction de la charge	1	milite pour l'utilisation de plateforme PAAS notamment
a.16	Le système doit être disponible	1	milite pour une plateforme type google engine + critère à définir pour chaque système particulier, notamment le temps de disponibilité devra être mesuré
a.17	Le système doit être tolérant aux pannes	1	Le choix de l'architecture matérielle et du support de l'application sera influencée. Milite pour l'adoption d'une plateforme PAAS
b.1	Le système doit permettre de lister et choisir les fichiers à partager	0	
b.2	Le système doit permettre de lister et administrer les utilisateurs autorisés à lister les fichiers disponibles	1	Base de données
b.3	Le système doit permettre aux utilisateurs d'exposer ses fichiers aux tiers	1	Composant serveur
b.4	Le système doit permettre aux utilisateurs de limiter l'accès aux seuls tiers autorisés/Contrôler l'accès aux fichiers	1	module accès + base de données
b.5	Le système doit permettre d'échanger ses informations de connexion par QR code	1	composant QR Code
b.6	Le système doit permettre de lister et récupérer les fichiers disponibles d'un pair	1	Client compatible avec le serveur d'un tiers

Tableau 35 – Exigences communes.

III.2.2 Le modèle de qualité du système pour le noyau

Il est indiqué dans (5) que « le modèle de qualité d'un système décrit ses besoins en termes de qualité. Nous […] proposons d'utiliser un tableau où pour chaque caractéristique et sous-caractéristique du standard, est précisé comment celle-ci doit être considérée dans le système : les besoins la mentionnant ; une priorité assignée (par exemple sur une échelle de 1 à 5, la priorité la plus forte étant de 1) ; la manière de l'évaluer (une mesure) ; un objectif valué fixé pour la mesure. Remarquons que parfois, la mesure peut ne pas être indiquée ou l'être très vaguement dans l'énoncé du besoin. Dans ce cas, l'architecture devra indiquer de quelle manière ces besoins auront été compris et pris en compte. Ces informations seront précisées dans le modèle de qualité de l'architecture (AQM). »

Voici l'ensemble des caractéristiques de qualité et leurs sous caractéristiques du modèle ISO 25010. Pour plus de lisibilité par la suite, un indice a été attribué à chacun d'entre eux. Il y sera fait référence.

functionnal suitability	**1**
Functional completeness	1.a
Functionnal correctness	1.b
Functionnal Appropriatness	1.c
performance efficiency	**2**
Time Behaviour	2.a
Resource utilisation	2.b
Capacity	2.c
compatibility	**3**
co-existance	3.a
interoperability	3.b
Usability	**4**
Appropriatness recognizability	4.a
Learnability	4.b
Operability	4.c
User error protection	4.d
User interface aesthetics	4.e
Accessibility	4.f

reliability	**5**
Maturity	5.a
Availability	5.b
fault tolerance	5.c
Recoverability	5.d
security	**6**
confidentiality	6.a
integrity	6.b
non-repudiation	6.c
accountability	6.d
authenticity	6.e
Maintainability	**7**
Modularity	7.a
Reusability	7.b
Analysability	7.c
Modifiability	7.d
Testability	7.e
Portability	**8**
Adaptability	8.a
Installability	8.b
Replaceability	8.c

Figure 21 – Indice des caractéristiques de qualité du modèle ISO 25010.

Pour fin d'illustration de l'apport de la norme ISO 25010, voici le modèle de qualité du système correspondant aux besoins exprimés avant l'étude de la norme. Ces besoins sont à l'origine de l'architecture noyau établie à la fin de la section II.4.

Caractèristiques / Sous caractèristiques ISO 25010	Besoin	priorité assignée	Attribut de qualité mesuré	objectif valué
2.a	a.4, a.11	2	Temps de réponse	à définir pour chaque produit selon une charte des temps
2.b	a.1, a.14	2	Nombre d'échanges, de demandes de traitements et volumes de données échangées et hébergées	à définir selon chaque produit de la famille
	a.2	1	Absence de données sauvegardées	attribut binaire : 1
2.c	a.12, a.13	2	Nombre de personnes	à définir selon chaque produit de la famille
3.a	a.4	2	Temps de réponse	moins d'une seconde
4.c	a.11	4	Temps pris par l'utilisateur pour lancer une tâche.	à définir pour chaque produit selon une charte des temps
4.e	a.5	5	Existence d'une charte visuelle respectant les bonnes pratiques en matière d'IHM	attribut binaire : 1
5.b	a.7	2	Disponibilité de multiple composants de communication	attribut binaire : 1
	a.16	1	Coté serveur, moyens techniques disponibles pour assurer la disponibilité du système	attribut binaire : 1
5.c	a.17	1	Coté serveur, moyens techniques disponibles pour assurer la tolérance aux pannes	attribut binaire : 1
6.a	a.3	2	Coté serveur, absence de moyens techniques pour garder une telle trace, coté client sécurisation des fluxs de data	attribut binaire : 1, 1
	a.8	2	Présence d'un composant de restriction d'accès	attribut binaire : 1
6.b	a.8	2	Présence d'un composant de restriction d'accès	attribut binaire : 1
7.b	a.6	4	Utilisation de format de données standard	attribut binaire : 1
8.a	a.10	5	Adapation des tamplates Android aux differents formats d'écrans	attribut binaire : 1
	a.15	2	Présence de moyen technique pour adapter les ressources disponibles à l'échelle des demandes	attribut binaire : 1

Tableau 36 – Modèle de qualité du système avant étude de la norme ISO 25010

L'étude de la norme ISO 25010 et de sa liste de caractéristiques nous amène à réfléchir à d'éventuels oublis. Il en résulte une amélioration de la liste des besoins (Tableau 37) puis du modèle de qualité (Tableau 38). Les ajouts sont indiqués en bleu :

indice	Exigence	Impact sur l'architecture	Commentaire
a.1	Le système devra permettre de partager des données en économisant les éventuelles ressources serveurs	1	Nettoyeur de bases de données selon critères
a.2	Le système devra permettre de partager du contenu de manière décentralisée : pour que les données restent celles des utilisateurs	0	Absence de base de données pour stocker ces données
a.3	Le système ne devra pas permettre à une tierce personne de savoir ce qui est échangé par qui	1	Aucun suivit côté serveur de liaison
a.4	Répondre aux critères Android en temps de réponse	0	
a.5	Le système doit garantir une utilisation agréable et simple	1	Composants IHM + Respecter une charte visuelle
a.6	Le système doit être ouvert à d'éventuelles évolutions quant aux environnements des clients participant aux échanges	1	Standards adoptés en terme de serveurs
a.7	Le système doit garantir une communication possible	1	Redondance de composants de communicatons
a.8	Le système doit garantir la sécurité des données des utilisateurs	1	Présence module accès + absence de données hebergées
a.9	Le systèle doit se protéger contre les échanges illégaux	0	Vocabulaire. Charte de moralité
a.10	Le système doit être utilisable par la majorité des appareils	0	cf liste des release Android, et dév pour la majorité + langue + taille des écrans : adapter aux écrans
a.11	Le système doit garantir aux utilisateurs qu'ils réaliseront chaque action en un temps donné	0	Charte des temps
a.12	Le système doit permettre à un certain nombre de pairs d'être impliqué dans l'échange d'un fichier simultanément. (multisource ou non)	1	à définir au niveau des systèmes particuliers
a.13	Le système doit être adapté au nombre de personnes avec lesquelles on envisage d'échanger (cercle restreint, élargi, ou sans limite à priori)	1	à définir au niveau systèmes particuliers
a.14	Le système devra être économe en terme d'énergie sur les systèmes Android	1	Format des données d'échange
a.15	Le système devra s'adapter en fonction de la charge	1	Milite pour l'utilisation de plateforme PAAS notamment
a.16	Le système doit être disponible	1	Milite pour une plateforme type google engine + critère à définir pour chaque système particulier, notamment le temps de disponibilité devra être mesuré
a.17	Le système doit être tolérant aux pannes	1	Le choix de l'architecture matérielle et du support de l'application sera influencée. Milite pour l'adoption d'une plateforme PAAS.
a.18	Le système doit être récupérable, relançable	1	Milite pour l'adoption d'une plateforme PAAS.
a.19	Le système devra fournir une documentation explicative quant à son utilisation.	0	
a.20	Le système devra fournir une log des pannes	1	Ajouter un composant de journalisation
a.21	Le système devra être modifiable sans effet de bord	1	Milite pour l'utilisation de plateforme PAAS notamment
a.22	Le système devra pouvoir être mis à jour de manière transparente	1	Milite pour l'adoption d'une plateforme PAAS + Utilisation de standards
b.1	Le système doit permettre de lister et choisir les fichiers à partager	0	
b.2	Le système doit permettre de lister et administrer les utilisateurs autorisés à lister les fichiers disponibles	1	Base de données
b.3	Le système doit permettre aux utilisateurs d'exposer ses fichiers aux tiers	1	Composant serveur
b.4	Le système doit permettre aux utilisateurs de limiter l'accès aux seuls tiers autorisés/contrôler l'accès aux fichiers	1	Module accès + base de données
b.5	Le système doit permettre d'échanger ses informations de connexion par QR code	1	Composant QR Code
b.6	Le système doit permettre de lister et récupérer les fichiers disponibles d'un pair	1	Client compatible avec le serveur d'un tiers
b.7	Le système doit permettre aux utilisateurs de savoir qui accède à leurs fichiers et quand	1	Prise en charge par un composant dédié

Tableau 37 – Nouvelle liste des besoins après étude de la norme ISO 25010

Caractèristiques / Sous caractèristiques ISO 25010	Besoin	priorité assignée	Attribut de qualité mesuré	objectif valué
2.a	a.4, a.11	2	Temps de réponse	moins d'une seconde, à définir pour chaque produit selon une charte des temps
2.b	a.1, a.14	2	nombre d'échanges, de demandes de traitements et volumes de données échangées et hébergées	à définir selon chaque produit de la famille
	a.2	1	Absence de données sauvegardées	attribut binaire : 1
2.c	a.12, a.13	2	Nombre de personnes	à définir selon chaque produit de la famille
3.a	a.4	2	Temps de réponse	moins d'une seconde
4.a	a.19	4	Existence d'une documentation explicative quant à l'usage	attribut binaire : 1
4.c	a.11	4	Temps pris par l'utilisateur pour lancer une tâche.	à définir pour chaque produit selon une charte des temps
4.e	a.5	5	Existence d'une charte visuelle respectant les bonnes pratiques en matière d'IHM	attribut binaire : 1
5.b	a.7	2	Disponibilité de multiple composants de communication	attribut binaire : 1
	a.16	1	Coté serveur, moyens techniques disponibles pour assurer la disponibilité du système	attribut binaire : 1
5.c	a.17	1	Coté serveur, moyens techniques disponibles pour assurer la tolérance aux pannes	attribut binaire : 1
5.d	a.18	1	Coté serveur, moyens techniques disponibles pour assurer la reprise après plantage de l'application	attribut binaire : 1
6.a	a.3	2	Coté serveur, absence de moyens techniques pour garder une telle trace, coté client sécurisation des fluxs de data	attribut binaire : 1, 1
	a.8	2	Présence d'un composant de restriction d'accès	attribut binaire : 1
6.b	a.8	2	Présence d'un composant de restriction d'accès	attribut binaire : 1
7.b	a.6	4	Utilisation de format de données standard	attribut binaire : 1
7.c	a.20	2	Stockage des log d'erreurs	attribut binaire : 1
7.d	a.21	2	Présence de procédure de modification, de documentation du système, et choix de plateforme facilitant le test de modificaiton	attribut binaire : 1, 1, 1
8.a	a.10	5	Adapation des tamplates Android aux differents formats d'écrans	attribut binaire : 1
	a.15	2	Présence de moyen technique pour adapter les ressources disponibles à l'échelle des demandes	attribut binaire : 1
8.c	a.22	3	Utilisation de format de données standard	attribut binaire : 1
		2	Présence de moyen techniques facilitant le passage à une nouvelle version	attribut binaire : 1
	ajout			

Tableau 38 – Nouveau modèle de qualité

Comme on peut le voir, toutes les caractéristiques de qualité énoncées par la norme ISO ne sont pas reportées ici. En pratique, la norme rappelle que l'ensemble des caractéristiques ne sont pas applicables à tout produit logiciel. Par ailleurs, le modèle de qualité ne doit reporter que les qualités importantes.

III.2.3 Le modèle de qualité fonctionnel pour le noyau : FQM

III.2.3.1 **Raffinement fonctionnel**

N. Levy et *al.* (5), définissent cette activité. « L'activité de raffinement fonctionnel (*Functional Refinement*) analyse les besoins utilisateur en termes fonctionnels, et les formalise à l'aide d'un ensemble de fonctionnalités. […] Les fonctions annotées comme

implicites prennent en compte une exigence perçue en première analyse, au niveau utilisateur, comme non fonctionnelle. » Nous nous penchons ici sur les besoins communs à tous les systèmes : le tableau suivant reprend les principales fonctionnalités communes, regroupées par classes.

classe de fonctionnalité	fonctionnalité	Utilisateur	Commentaire
Gestion de droits sur les fichiers	Lister les fichiers	Partageur	
	Attribuer des droits	Partageur	
	Consulter l'historique d'accès aux fichiers	Partageur	
Gestion des utilisateurs	Lister les utilisateurs	Partageur	
	Modifier leurs droits	Partageur	
	Ajouter un nouvel utilisateur	Partageur	
	Emettre un QR code pour le partage	Partageur / tiers	
	Soumettre un QR code	Partageur / tiers	
Récupération de fichiers	Lister les fichiers chez un partageur	Tiers	
	Récupérer un fichier sélectionné	Tiers	
Authentification	Saisie de mot de passe	Tiers	Fonction implicite
	Enregistrement du mot de passe	Tiers	Fonction implicite

Tableau 39 – Raffinement fonctionnel du noyau.

III.2.3.2 **FQM**

Ce modèle est encore une fois clairement défini dans (5) : «Le modèle de qualité fonctionnelle (FQM) permet de relier les exigences fonctionnelles et les propriétés de qualité qui peuvent être requises pour bien accomplir chaque fonctionnalité. […] Le modèle de qualité fonctionnelle synthétise l'ensemble des exigences de qualités afférant aux fonctionnalités du domaine. Ce modèle précise, pour chaque fonctionnalité identifiée au niveau du domaine, la ou les exigences non fonctionnelles attendues modulant celle-ci (telles que performance, sécurité, etc.). Les exigences de niveau domaine peuvent être, pour certaines, définies de manière parfaitement précise dès le niveau domaine et s'appliquer à tout système du domaine, parce qu'intimement liées aux pratiques professionnelles considérées : ce sont les exigences spécifiques du niveau domaine. D'autres sont définies de manière générique et nécessitent la définition de la valeur d'un paramètre propre à un système particulier. C'est le cas où une certaine exigence doit être présente dans tout système du domaine, avec cependant une certaine variation concernant soit le niveau de l'exigence, soit ses modalités précises. Ce sont les exigences génériques (telles que des exigences de disponibilité). »

Il s'agit donc de relier des caractéristiques de qualité aux exigences fonctionnelles que nous avons établies. En reprenant le tableau obtenu après raffinement fonctionnel, il suffit de lister les fonctionnalités attendues et de leur associer une ou plusieurs caractéristiques de qualité de la norme ISO 25010. C'est ce que fait le tableau suivant. Il indique aussi la manière d'évaluer ces caractéristiques (une mesure) et un objectif valué quand cela est applicable.

fonction	Caractérisitique / Sous caractéristique	Attribut de qualité mesuré [paramètre]	Paramètre
Lister les fichiers	1.c, 4.d,4.c	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 3 s
Attribuer des droits	6.a, 6.b	Accès aux fichiers restreints aux seuls fichiers partagés [liste des fichiers renvoyée filtrée]	_
Consultation Historique d'accès aux fichiers	6.d	Possibilité de savoir qui accède à quoi [Présence d'un composant dédié]	_
Lister les utilisateurs	1.c, 4.d, 4.c	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 3 s
Modifier leurs droits	6.a, 6.b, 6.e	Accès aux fichiers restreint aux seules personnes autorisées [présence d'un composant de restriction d'accès]	_
Ajouter un nouvel utilisateur	4.d	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	
Emettre un QR code pour le partage	1.c, 4.d	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	
Soumettre un QR code	1.c, 4.d	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	
lister les fichiers chez un partageur	1.c, 4.d	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 10 s
Récupérer un fichier sélectionné	1.c	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	
Saisie de mot de passe	6.a, 6.b, 6.e	Authentification des tiers[méthode d'authentification]	
Enregistrement du mot de passe	1.c, 4.d	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	

Tableau 40 – Modèle de qualité fonctionnel du noyau.

III.2.4 L'architecture noyau préliminaire

« Partant des exigences utilisateur pertinentes, l'activité de définition de l'architecture préliminaire […] opère un choix de style architectural, et propose ainsi une première ébauche architecturale de système, base à des adaptations ultérieures […], laquelle ne tient pas encore compte, des exigences de qualité […]. Elle se traduit par le choix d'un style architectural et l'identification des composants de base, avec traçabilité fonction vers composants. Cette activité peut être commencée en parallèle de l'activité de raffinement fonctionnel, dans la

mesure où le style architectural choisi ne dépend pas de manière directe du détail des fonctionnalités attendues. » (5)

La Figure 22 ne représente que les éléments induits par les besoins fonctionnels et permet d'établir un lien de traçabilité entre les fonctions attendues et les composants les réalisant. Cette architecture préliminaire se distingue du noyau obtenu fin section II.4 notamment par l'absence d'une architecture REST (elle sera justifiée plus tard par des exigences non fonctionnelles) et l'ajout d'un composant « logger » (en réponse au besoin b.7). Certaines améliorations peuvent déjà être apportées à cette étape.

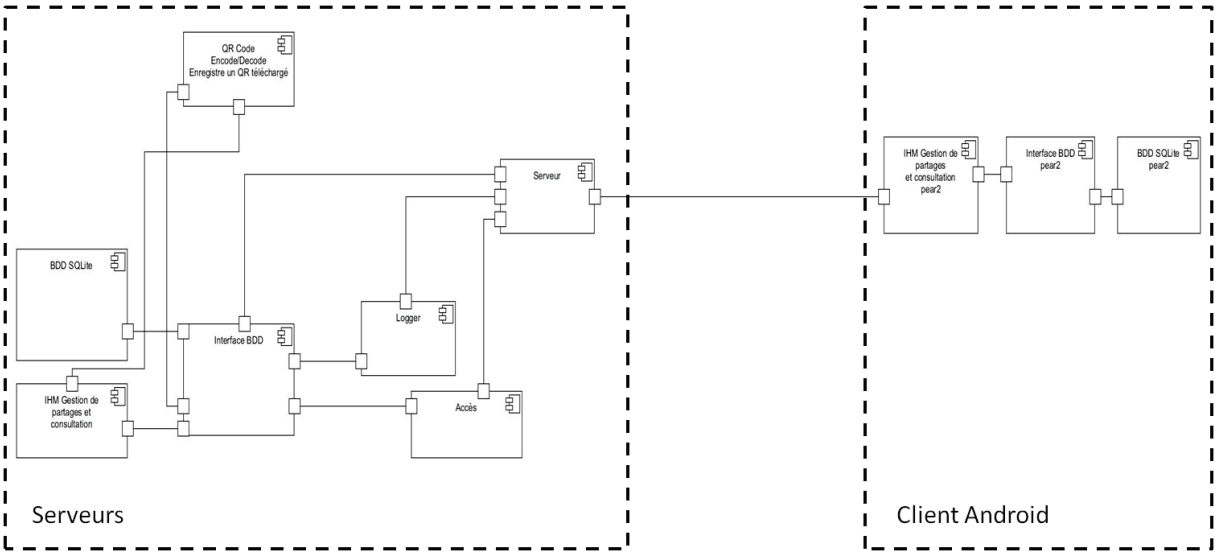


Figure 22 – Architecture préliminaire du noyau

Ci-dessous, la même représentation, indiquant cette fois-ci les exigences fonctionnelles à l'origine de la présence des composants.

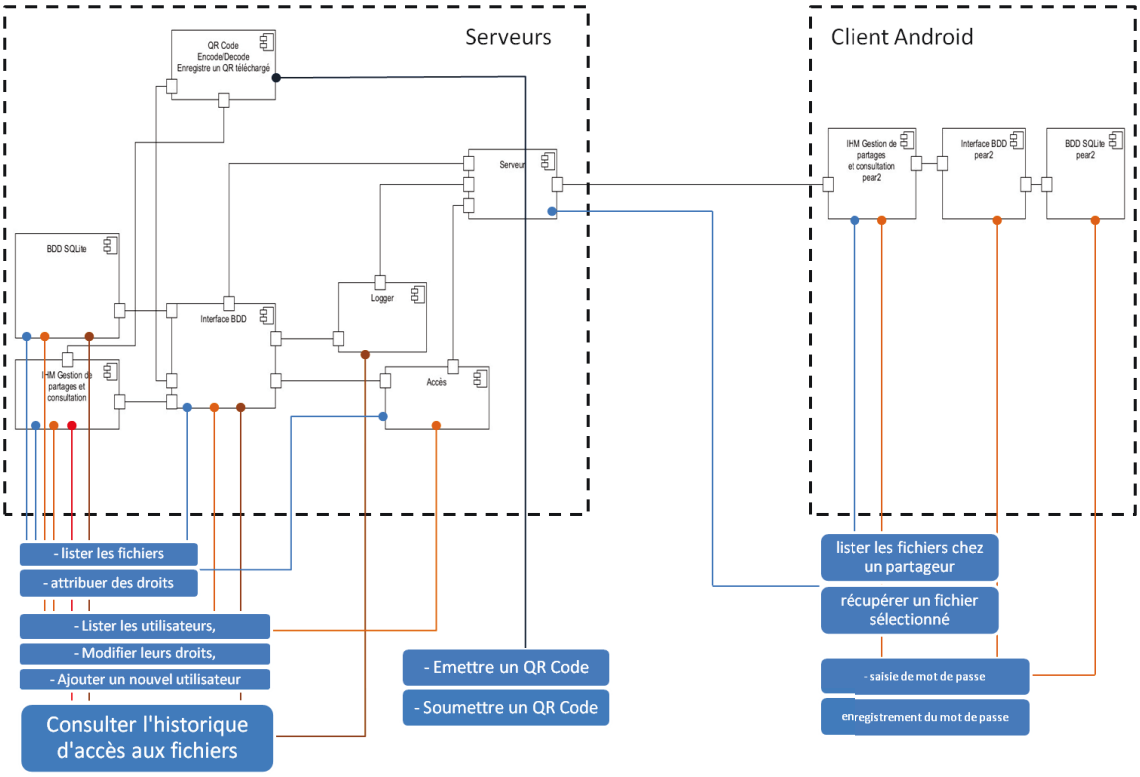


Figure 23 - Architecture préliminaire du noyau avec lien de traçabilité fonctionnel.

III.2.5 Le modèle de qualité architecturale du noyau: AQM

Ici les exigences non fonctionnelles, non attachées à des fonctions particulières ayant un impact sur l'architecture sont prises en compte. Ce sont celles exprimées dans le modèle de qualité du système. Leur impact peut être de nature à modifier l'architecture préliminaire par l'ajout de composants par exemple. L'objectif est de montrer comment les différents éléments de l'architecture prennent en compte les besoins exprimés dans le modèle de qualité du système. Le lien de traçabilité entre une exigence de qualité exprimée lors de l'analyse des besoins et les composants la prenant en charge est établi afin de permettre de vérifier qu'aucune d'entre elles ne sera oubliée lors du développement du système.

Le modèle de qualité d'une architecture (AQM) sera présenté sous forme de tableau indiquant comment chaque sous-caractéristique est prise en compte par l'architecture. La mesure appliquée et la valeur dérivée de celle mentionnée dans le modèle du système peuvent être indiquées avec plus de précision.

Besoins non fonctionnels liés à l'architecture	Caractérisitique / Sous caractéristique	Attribut de qualité mesuré [paramètre]	Domaine du paramètre	Solution architecturale
a.1	2.b	Absence de données sauvegardées	–	Absence de table pour ces données
		Nombre d'échanges, de demandes de traitements et volume de données échangées et hébergées	–	Nettoyeur de base de données
a.2	2.b	Absence de données sauvegardées	–	Absence de table pour ces données
a.3	6.a	Côté serveur, absence de moyens techniques pour garder une telle trace.	–	Absence de table pour ces données
		Coté terminaux Android, chiffrement des communicaitons entre pairs	–	–
a.4	2.a, 3.a	Temps de réponse	< 2s	–
a.5	4.e	Existence d'une charte visuelle respectant les bonnes pratiques en matière d'IHM	–	–
a.6	7.b	Utilisation de format de données standard.	–	données json.
		Utilisation de protocole de communication largement répendu.	–	Uitilisation d'HTTP et de serveur WEB/REST.
a.7	5.b	Disponibilité de multiple composants de communication		–
a.8	6.a	Présence d'un composant de restriction d'accès	–	composant accès
		absence de données hébergées	–	absence de table
a.10	8.a	Adapation des tamplates Android aux differents formats d'écrans	–	–
a.11	2.a, 4.c	Conformité à une charte des temps	–	–
a.14	2.b,	Format des données d'échange	–	échange json
a.15	5.b	Présence de moyens techniques pour adapater les ressources disponibles aux demandes	–	Milite pour l'utilisation de plateform PAAS.. Notamment
a.16	5.b	Garantie d'accessibilité [taux]	98%	Milite pour google engine
a.17	5.c	Garantie d'un RTO [temps]	1h	Milite pour l'utilisation de plateform PAAS.
		Garantie d'un RPO [temps]	selon les tables	
a.18	5.d	Garantie d'un RTO [temps]	1h	Milite pour l'utilisation de plateform PAAS
a.19	4.a	Présence d'une documentation consultable.	–	–
a.20	7.c	Présence d'une telle log	_	Logger
a.21	7.d	Moyens techniques le permettant	–	milite pour l'utilisation de plateform PAAS.. Notamment
a.22	8.c	Coté serveur, présence de moyen technique facilitant	–	milite pour l'adoption d'une plateforme PAAS

Tableau 41 – Modèle de qualité architecturale du noyau

III.3 L'architecture noyau de qualité

III.3.1 Le modèle de qualité global du noyau

« Le modèle de qualité global est obtenu en fusionnant les exigences issues des modèles FQM et AQM ordonnées par l'architecte. » (5)

Source (FQM| AQM)	Fonction (si FQM) / Besoin non foncitonnel (si AQM)	Caractéristique / Sous caractéristique	attribut de qualité mesuré [paramètre]	Domaine du paramètre
FQM	Lister les fichiers	1.c, 4.d,4.c	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 3 s
	Attribuer des droits	6.a, 6.b	Accès aux fichiers restreints aux seuls fichiers partagés [liste des fichiers renvoyée filtrée]	_
	Consultation Historique d'accès aux fichiers	6.d	Possibilité de savoir qui accède à quoi [Présence d'un composant dédié]	_
	Lister les utilisateurs	1.c, 4.d, 4.c	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 3 s
	Modifier leurs droits	6.a, 6.b, 6.e	Accès aux fichiers restreint aux seules personnes autorisées [présence d'un composant de restriction d'accès]	_
	Ajouter un nouvel utilisateur	4.d	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	
	Emettre un QR code pour le partage	1.c, 4.d	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	
	Soumettre un QR code	1.c, 4.d	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	
	Lister les fichiers chez un partageur	1.c, 4.d	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 10 s
	Récupérer un fichier sélectionné	1.c	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	
	Saisie de mot de passe	6.a, 6.b, 6.e	Authentification des tiers[méthode d'authentification]	
	Enregistrement du mot de passe	1.c, 4.d	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	
AQM	a.1	2.b	Absence de données sauvegardées	_
			Nombre d'échanges, de demandes de traitements et volume de données échangées et hébergées	_
	a.2	2.b	Absence de données sauvegardées	_
	a.3	6.a	Côté serveur, absence de moyens techniques pour garder une telle trace.	_
			Côté terminaux Android, chiffrement des communicaitons entre pairs	_
	a.4	2.a, 3.a	temps de réponse	< 2s
	a.5	4.e	Existance d'une charte visuelle respectant les bonnes pratiques en matière d'IHM	_
	a.6	7.b	Utilisation de format de données standard.	_
			Utilisation de protocole de communication largement répendu.	_
	a.7	5.b	Disponibilité de multiple composants de communication	
	a.8	6.a	Présence d'un composant de restriction d'accès	_
			Absence de données hébergées	_
	a.10	8.a	Adapation des tamplates Android aux differents formats d'écrans	_
	a.11	2.a, 4.c	Conformité à une charte des temps	_
	a.14	2.b,	Format des données d'échange	_
	a.15	5.b	Présence de moyens techniques pour adapater les ressources disponibles aux demandes	_
	a.16	5.b	Garantie d'accessibilité [taux]	98%
	a.17	5.c	Garantie d'un RTO [temps]	1h
			Garantie d'un RPO [temps]	selon les tables
	a.18	5.d	Garantie d'un RTO [temps]	1h
	a.19	4.a	Présence d'une documentation consultable.	_
	a.20	7.c	Présence d'une telle log	_
	a.21	7.d	Moyens techniques le permettant	_
	a.22	8.c	Côté serveur, présence de moyen technique facilitant le travail de mise à jour sans impact	_

Tableau 42 – Modèle de qualité global du noyau.

III.3.2 L'architecture de qualité :

Il s'agit d'intégrer toutes les exigences issues du modèle global pour obtenir « l'architecture qui satisfait l'ensemble des exigences fonctionnelles et non fonctionnelles » (5). Des modifications sont apportées à l'architecture préliminaire. Elles sont induites par les exigences issues de l'AQM tel que décrit précédemment.

N. Levy *et al.* proposent la démarche suivante : « - partir de l'architecture « initiale » répondant aux exigences fonctionnelles, […] justifiée que par les exigences fonctionnelles et le(s) style(s) associé(s) au domaine. […]

- intégrer progressivement les différentes exigences de qualité

a) en recherchant à chaque étape les différentes modalités possibles de prise en compte de l'exigence considérée dans l'architecture proposée à l'étape actuelle de la démarche,

b) en choisissant l'une d'entre elles et en modifiant conséquemment l'architecture,

c) en vérifiant que l'architecture ainsi modifiée satisfait toujours l'ensemble des exigences de qualité considérées lors des précédentes étapes.

Les modifications de l'architecture introduites à chaque étape peuvent être :

– des ajouts de nouveaux composants. Typiquement, un tel composant ajouté sera porteur d'une exigence fonctionnelle ou d'une exigence de qualité qui garantira l'obtention de la propriété de qualité recherchée au niveau système ;

– des ajouts d'exigences fonctionnelles ou non fonctionnelles posées sur un composant déjà identifié et présent dans l'architecture ;

– des ajouts d'exigences concernant l'infrastructure d'exécution ;

– des modifications de la topologie de l'architecture (connexions entre composants) impliquant un retour arrière pour vérifier la validité des choix préalables ;

– des combinaisons de ces différentes possibilités. » (5)

Certaines exigences restent génériques au niveau domaine. Il faudra préciser les modalités de leur implémentation au niveau des systèmes particuliers. Les composants les implémentant sont indiqués dans l'architecture de domaine et les choix quant aux modalités de leur implémentation restent à faire au niveau système particulier. N.Levy *et al.* parlent d'impact architectural différé.

Exigence	a.1- Economiser les ressources serveurs.
Caractérisitque de qualité	2.b Performance efficiency / Ressource utilisation
Exigence de qualité	Nombre d'échanges, de demandes de traitements et volume de données échangées et hébergées
Mode de prise en compte/ décision de conception	Niveau architectural : - Dans la logique applicative : limiter les échanges au maximum (être le moins verbeux possible) - Stocker le moins de données possible. Décision: Ajout de composant: Nettoyeur BDD. Report des critères de suppression des données liées à l'application au niveau système.
Impact architecturale	Ajout d'un composant : Nettoyeur avec l'exigence fonctionnelle "supprimer les données [critères]"

Tableau 43 – Impact architectural a.1

Exigence	a.3 - Confidentialité des échanges
Caractérisitque de qualité	6.a Security / Confidentiality.
Exigence de qualité	Chiffrement des communications
Mode de prise en compte/ décision de conception	- Adoption d'une méthode de chiffrement des communications. Décision: Ajout d'éxigence fonctionnelle "chiffrement des communication [méthode]" sur les composants client REST et Web Mobile
Impact architecturale	Immédiat : Ajout d'exigence foncitonnelle "chiffrement des communication [méthode]" sur les composants client REST et Web Mobile Diféré: Implémentation de la méthode de chiffrement.

Tableau 44 - Impact architectural a.3

Exigence	a.6 - Ouverture à d'éventuelles évolutions quant aux environnements clients participants
Caractérisitque de qualité	7.b Maintainability / Reusability
Exigence de qualité	Utilisation de format de données et de protocole de communication largement répendus
Mode de prise en compte/ décision de conception	- Adoption d'un format d'échange de données répendu et simple à exploiter. - Adotpion d'un protocole d'échange répendu et largement utilisé. Décision: Ajout de composants: Serveur REST, clients REST et serveur Web.
Impact architecturale	Ajout de composants: Serveur REST, clients REST et serveur Web. Modification des interfaces des composants présents.(Logger, interface BDD et IHM pear2)

Tableau 45 - Impact architectural a.6

Exigence	a.8 - Sécurité des données utilisateurs
Caractérisitque de qualité	6.a Security/ Confidentiality
Exigence de qualité	Présence d'un composant de restriction d'accès restreignant l'accès aux fichiers en fonction de droits
Mode de prise en compte/ décision de conception	Niveau fonctionnel : - Authentification par identifiant/mot de passe. Niveau Architectural : - Evaluation des droits selon la politique d'accès définie: A chaque requête sur les fichiers. Lors de l'accès, avec gestion de sessions utilisateur.
Impact architecturale	Immédiat : Ajout d'exigence fonctionnelle " Gérer les droits[politique]" sur le composant Accès. Ajout d'exigence fonctionnelle sur le composant BDD SQLite Gestion d'entités "pairs" et "droits d'un pairs sur un fichier" Ajout d'exigence fonctionnelle "fournir un login/mot de passe" sur le composant IHM pear 2. Différé: Implémentation de la politique d'accès spécifiée au niveau système.

Tableau 46 – Impact architectural a.8

Exigence	a.20- Journalisation des pannes.
Caractérisitque de qualité	7.c Maintainability / Analysability
Exigence de qualité	Stocker les messages relatifs aux erreurs/pannes.
Mode de prise en compte/ décision de conception	Décision: - Ajout de l'exigence foncitonnel "Stocker les messages d'erreurs relatifs aux pannes [critères]" sur le composant logger. - Ajout d'une table correspondante dans la base.
Impact architecturale	Immédiat : - Ajout de l'exigence fonctionnel "Stocker les messages d'erreurs relatifs aux pannes [critères]" sur le composant logger. - Ajout d'une table correspondante dans la base. Différé : - Implémentation des critères de choix des messages d'erreurs à stocker au niveau système particulier.

Tableau 47 - Impact architectural a.20

La figure suivante montre l'architecture de qualité que l'on obtient suite à l'intégration de toutes les exigences issues du modèle global, en prenant pour base l'architecture initiale.

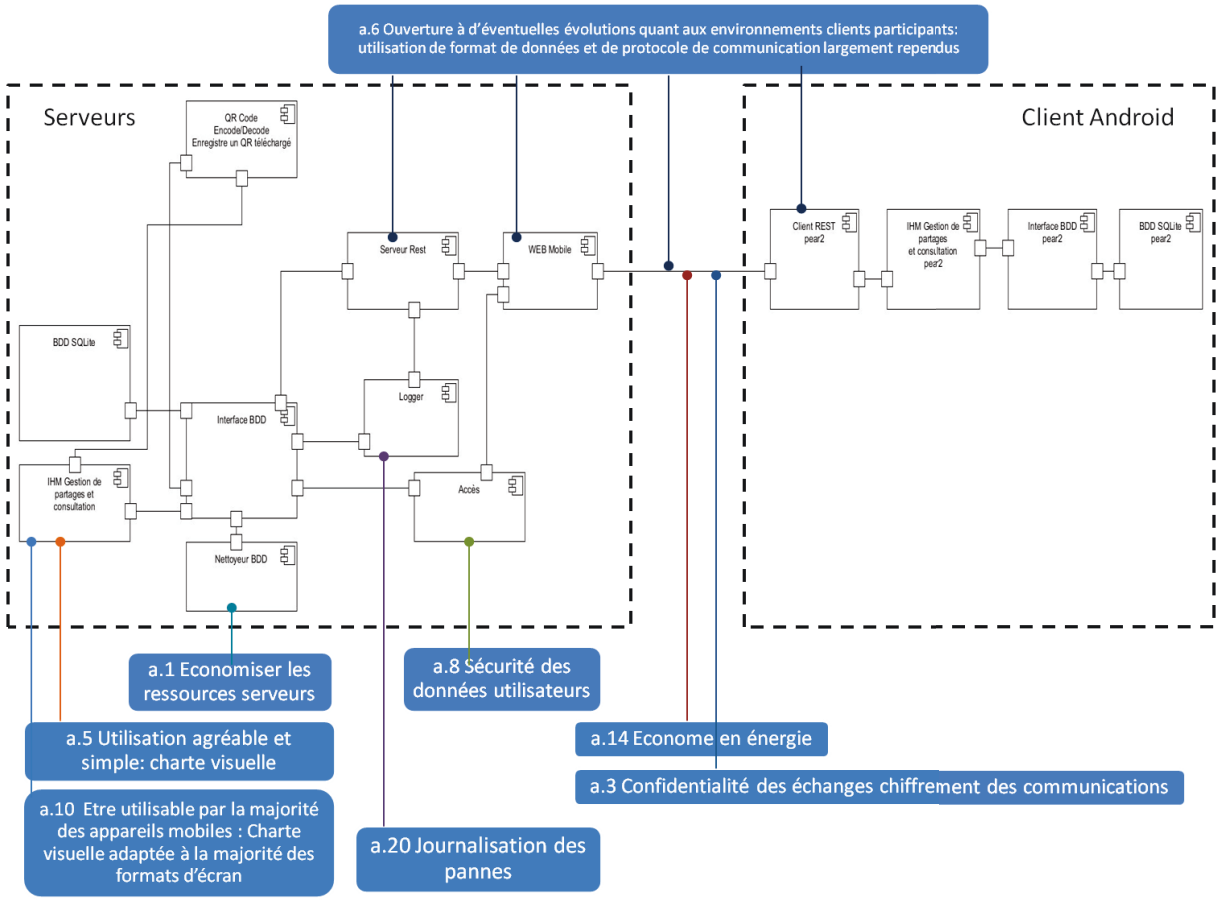


Figure 24 – Architecture de qualité

Pour résumer ce qui a été fait, voici un schéma de l'ensemble des tâches à effectuer pour obtenir une architecture de qualité :

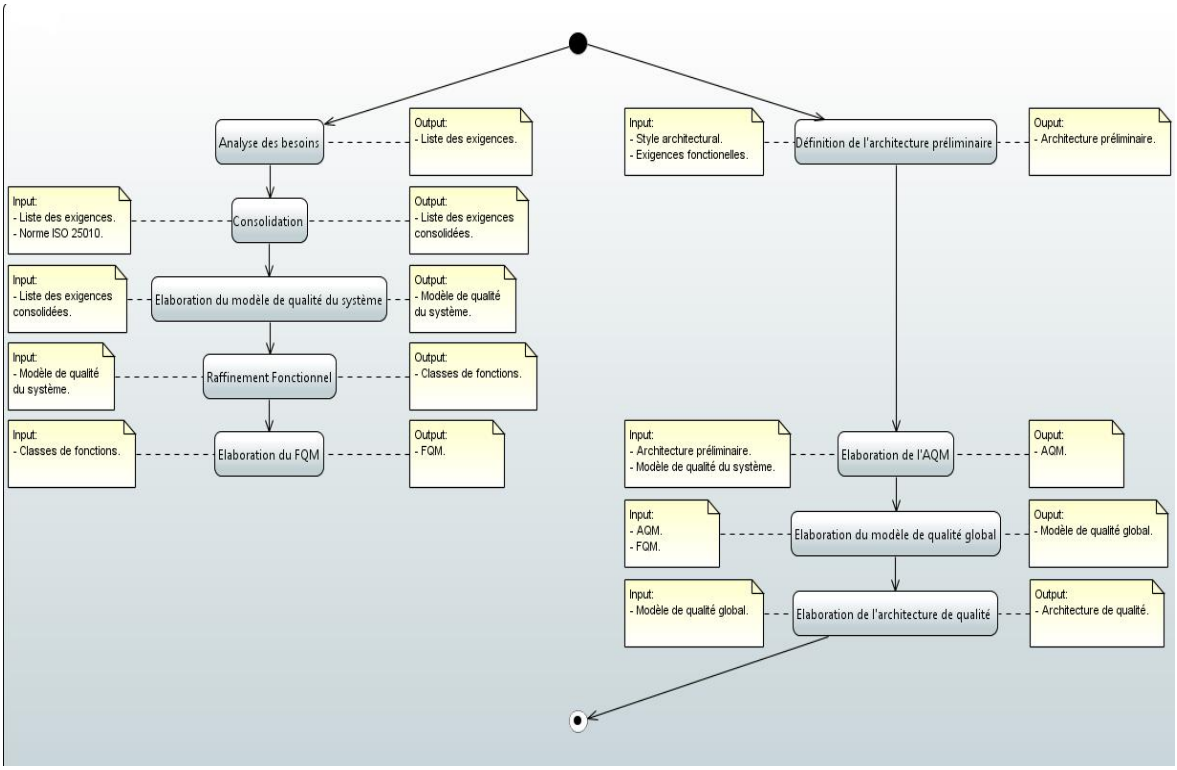


Figure 25 – Elaboration d'une architecture de qualité.

La méthode donnée définit clairement le contenu de chaque modèle. Ils ne sont pas difficiles à remplir, bien que le travail soit fastidieux. Toutefois ils apportent une documentation précieuse permettant facilement d'établir un lien de traçabilité entre les besoins, les fonctionnalités et les composants. Ainsi, aucun attribut de qualité souhaitable ne peut être omis.

III.4 Qualité des variants

Les futurs produits de la famille seront une composition faite à partir du noyau de qualité précédemment défini et des variants tels que décrits précédemment (cf. II.3). Afin de garantir la qualité des futurs produits dérivés, il faut donc s'assurer de la qualité des variants. La même méthode peut être reprise mais elle nécessitera des adaptations au contexte des variants : elle sera appliquée à leurs éléments en les considérant comme un tout cohérant. Les besoins fonctionnels et non fonctionnels doivent toujours être pris en compte d'un point de vue « domaine » au sens de (5)[5]. Il s'agit ici d'étudier les besoins spécifiques au variant et à la fonction qu'il remplit.

[5] Voir note 2

Nous allons étudier les besoins fonctionnels et non fonctionnels spécifiques à chaque variant, établir un modèle de qualité de chaque variant, un modèle de qualité fonctionnel (FQM) et en déduire une architecture préliminaire. Etablir un modèle de qualité de l'architecture et appliquer ses exigences à l'architecture du variant. A cette étape nous établirons une hiérarchie des exigences : celles propres au variant et celles héritées des exigences du domaine.

III.4.1 Modèle de qualité des variants

Il s'agit d'énoncer les besoins fonctionnels et non fonctionnels propres au variant considéré. Pour une meilleure compréhension, dans ce qui suit, l'analyse des besoins suit deux étapes :

- un premier tableau reprend la liste des exigences dressées au niveau domaine. Certaines ne sont plus pertinentes (en bleu) au niveau variant. Certaines exigences ne sont pas introduites spécifiquement par la fonction du variant mais sont à appliquer ici, et potentiellement à d'autres variants. Elles sont transversales. Ce sont les exigences « héritées » indiquées en vert. En orange, les exigences non impactantes car déjà garanties au niveau domaine. Enfin, y sont ajoutées les exigences spécifiques au variant. Ce tableau est le résultat de l'analyse des besoins à l'aune de la norme ISO 25010.

- un second tableau n'indique que les exigences encore applicables au variant, une fois éliminées les exigences non pertinentes ou sans incidences directes au niveau variant.

Certaines exigences déjà prises en compte et mise en place au niveau domaine seront assurées au niveau variant sans impact, sans aucune action nécessaire. Le variant « hérite » alors simplement de la caractéristique de qualité prise en compte au niveau domaine, sans avoir à intégrer l'exigence qui l'introduit au niveau domaine. Parfois une exigence énoncée au domaine va devoir être reprise au niveau variant, elle aura une influence sur l'implémentation du variant : ajout d'une fonctionnalité, impact architectural, ou moins visiblement, dans la manière d'implémenter le variant. On observe alors que l'ensemble des exigences d'un variant se compose des exigences strictement liées à la fonction du variant, auxquelles peuvent s'ajouter des exigences de qualité définies pour l'ensemble des produits de la famille. L'ensemble des exigences variant hérite en quelque sorte de certaines exigences de qualité du niveau noyau, lorsqu'elles sont transversales et qu'elles sont aussi à prendre en compte et à appliquer au niveau variant.

A la suite de l'analyse des exigences, un dernier tableau donne le modèle de qualité des variants à partir des besoins énoncés.

III.4.1.1 **Variant connexionLocale**

Concernant le variant connexionLocale, il s'agit de permettre une connexion entre appareils mobiles à courte distance, dans un réseau local sans fils.

indice	**Exigence** Le système devra pemettre …	**Impact sur l'architecture du variant**	**Commentaire**
a.1	…de partager des données en économisant les éventuelles ressources serveurs	0	N/A dans ce cas. Se rapporte d'avantage à des ressources serveurs traditionnelles.
a.2	…de partager du contenu de manière décentralisée : pour que les données restent celles des utilisateurs	0	La nature même du partage local le garantit
a.3	…d'interdire à une tierce personne de savoir ce qui est échangé par qui	0	Logique logicielle : Communications chiffrées,
a.4	…de répondre aux critères de temps de réponse Android	0	
a.5	…de garantir une utilisation agréable et simple	0	Exigence déjà prise en compte au niveau noyau
a.6	…d'être ouvert à d'éventuelles évolutions quant aux environnements des clients participant aux échanges	0	N/A au niveau variant. Standard déjà introduits au niveau noyau. Qualité héritée.
a.7	…de garantir une communication possible	1	redondance de composants de communicatons : hotspot et Wifi P2P
a.8	…de garantir la sécurité des données des utilisateurs	0	Exigence déjà prise en compte au niveau noyau
a.9	…de ne pas encourager les échanges illégaux	0	Exigence déjà prise en compte au niveau noyau
a.10	…d'être utilisable par la majorité des appareils	0	Exigence déjà prise en compte au niveau noyau
a.11	…de garantir aux utilisateurs qu'ils réaliseront chaque action en un temps maximum établi	0	Exigence déjà prise en compte au niveau noyau
a.12	…un certain nombre de pairs d'être impliqué dans l'échange d'un fichier simultanément. (multisource ou non)	0	Un seul, pas de composant permettant le multi-source. Wifi P2P acceptable.
a.13	Le système doit être adapté au nombre de personnes avec lesquelles on envisage d'échanger (cercle restreint, élargi, ou sans limite à priori)	0	Peu de pairs, cercle restreint. Connaissances directes. Pas de serveur de liaison nécessaire.
a.14	…de minimiser les besoins en énergie sur les systèmes Android	0	Soin apporté à l'optimisation du code pour limiter le temps de traitement
a.15	…de s'adapter en fonction de la charge	0	N/A dans ce cas. Se rapporte d'avantage à des ressources serveurs traditionnelles.

a.16	...d'être disponible	0	N/A dans ce cas, considération prise en compte par le système ANDROID sous-jacent
a.17	...d'être tolérant aux pannes	0	N/A dans ce cas, considération prise en compte par le système ANDROID sous-jacent
a.18	...d'être récupérable,relançable	0	N/A dans ce cas. Se rapporte d'avantage à des ressources serveurs traditionnelles.
a.19	Une documentation explicative quant à l'usage devra être fournie	0	N/A à ce niveau
a.20	...de fournir une log des pannes devra être disponible à l'analyse	0	Exigence déjà prise en compte au niveau noyau / une faute de connexion locale se repercutera sur le serveur REST déjà monitoré
a.21	Faire en sorte que le produit soit modifiable sans effets de bords	0	N/A dans ce cas, considération prise en compte par le système ANDROID sous-jacent
a.22	...d'être mis à jour de manière transparente	0	N/A dans ce cas, considération prise en compte par le système ANDROID sous-jacent
b.1	...de lister et choisir les fichiers à partager.	0	Fonctionnalité déjà prise en compte au niveau noyau
b.2	...de lister et administrer les utilisateurs autorisés à lister les fichiers disponibles.	0	Fonctionnalité déjà prise en compte au niveau noyau
b.3	...d'exposer ses fichiers aux tiers.	0	Fonctionnalité déjà prise en compte au niveau noyau
b.4	...de limiter l'accès aux seuls tiers autorisés/contrôler l'accès aux fichiers.	0	Fonctionnalité déjà prise en compte au niveau noyau
b.5	...d'échanger ses informations de connexion par QR code	0	Fonctionnalité déjà prise en compte au niveau noyau
b.6	...de lister et récupérer les fichiers disponibles d'un pair.	0	Foncitonnalité déjà prise en compte au niveau noyau
b.7	...de savoir qui accède aux fichiers et quand.	0	Fonctionnalité déjà prise en compte au niveau noyau
b.8	...d'établir une connexion locale en se servant de l'infrastructure existante	0	
b.9	...d'établir un hotspot wifi pour permettre la connexion par d'autres pairs	1	Composant HotSpot Wifi
b.10	...d'établir une connexion Wifi P2P	1	Composant Wifi P2P
	exigence héritée		
	exigence non pertinente au niveau variant.		
	exigence satisfaite au niveau noyau et/ou non applicable au variant		

Tableau 48 – Exigences du variant ConnexionLocale.

indice	Exigence Le système devra pemettre …	Impacte sur l'architecture du variant	Commentaire
a.3	...d'interdire à une tierce personne de savoir ce qui est échangé par qui	0	Logique logicielle : Chiffrement des communications
a.4	...de répondre aux critères de temps de réponse Android	0	
a.7	...de garantir une communication possible	1	Redondance de composants de communicatons : hotspot et Wifi P2P
a.11	...de garantir aux utilisateurs qu'ils réaliseront chaque action en un temps maximum établi	0	Exigence déjà prise en compte au niveau noyau
a.12	...un certain nombre de pairs d'être impliqué dans l'échange d'un fichier simultanément. (multisource ou non)	0	Un seul, pas de composant permettant le multi-source. Wifi P2P acceptable.
a.13	Le système doit être adapté au nombre de personnes avec lesquelles on envisage d'échanger (cercle restreint, élargi, ou sans limite à priori)	0	Peu de pairs, cercle restreint. Connaissances directes. Pas de serveur de liaison nécessaire.
a.14	...de minimiser les besoins en énergie sur les systèmes Android	0	Soin apporté à l'optimisation du code pour limiter le temps de traitement
b.8	…d'établir une connexion locale en se servant de l'infrastructure existante	0	
b.9	…d'établir un hotspot wifi pour permettre la connexion par d'autres pairs	1	Composant HotSpot Wifi
b.10	…d'établir une connexion Wifi P2P	1	Composant Wifi P2P
	exigence héritée		

Tableau 49 – Exigences connexionLocale impactantes.

L'exigence a.7 se retrouve aussi au niveau domaine. Dans le cadre de la ConnexionLocale elle se traduit par l'ajout d'exigences fonctionnelles et de ses composants associés. Les exigences b.9 et b10 découlent finalement de l'exigence non fonctionnelle héritée a.7. C'est pour cela qu'elles sont indiquées comme exigences héritées, bien que non clairement indiquées au niveau domaine. a.11 est prise en compte au niveau architectural par le noyau.

Les exigences du variant connexionLocale ne traitent que de la connexion Wifi entre pairs. Les exigences de sécurité liées à de telles communications sont incluses dans les exigences de sécurité du niveau domaine, de même que pour les exigences de performance, compatibilité, utilisabilité, de maintenabilité ou de fiabilité. Il n'y a rien à ajouter ici (en termes de composants par exemple).

Caractèristiques / Sous caractèristiques ISO 25010	Besoin	priorité assignée	Attribut de qualité mesuré	objectif valué
1.a	b.8	3	Possibilité offerte	attribut binaire : 1
2.b	a.14	5	Nombre d'échanges, de demandes de traitements et volumes de données échangées et hébergées	à déterminer au niveau système
2.c	a.12	3	Nombre de personnes	un seul, pas de composant permettant le multi-source. Wifi P2P acceptable
	a.13	3	Nombre de personnes	cercle restreint
2.a	a.4	5	temps de réponse	moins d'une seconde
4.c	a.11	4	Temps pris par l'utilisateur pour lancer une tâche.	à définir pour chaque produit selon une charte des temps
5.b	a.7	3	Disponibilité de multiple composants de communication	attribut binaire : 1
	b.9	5	Possibilité offerte	attribut binaire : 1
	b.10	5	Possibilité offerte	attribut binaire : 1
6.a	a.3	5	Sécurisation des fluxs de data	attribut binaire : 1

Tableau 50 – Modèle de qualité connexionLocale.

III.4.1.2 **Variant utilisationNomade**

Le variant utilisationNomade, doit permettre une connexion entre appareils mobiles hors réseaux local. Par anticipation sur les compositions noyau/variants possibles, l'indice des exigences continue après celles déjà utilisées pour le variant connexionLocale.

indice	Exigence Le système devra pemettre …	Impacte sur l'architecture	Commentaire
a.1	...de partager des données en économisant les éventuelles ressources serveurs	1	Nettoyeur de bases de données selon critères
a.2	...de partager du contenu de manière décentralisée : pour que les données restent celles des utilisateurs	0	Absence de base de données pour stocker ces données
a.3	...d'interdire à une tierce personne de savoir ce qui est échangé par qui	1	Chiffrement/Déchiffrement ID IP
a.4	...de répondre aux critères de temps de réponse Android	0	
a.5	...de garantir une utilisation agréable et simple	1	Upnp et intégration à IHM
a.6	...d'être ouvert à d'éventuelles évolutions quant aux environnements des clients participant aux échanges	1	standard adoptés en terme de serveurs coté LINK : WEB serveur et REST serveur
a.7	...de garantir une communication possible	1	milite pour une plateforme type PAAS
a.8	...de garantir la sécurité des données des utilisateurs	0	absence de données hebergées
a.9	...de ne pas encourager les échanges illégaux	0	Exigence déjà prise en compte au niveau noyau
a.10	...d'être utilisable par la majorité des appareils	0	Exigence déjà prise en compte au niveau noyau
a.11	...de garantir aux utilisateurs qu'ils réaliseront chaque action en un temps maximum établi	0	charte des temps, coté serveur LINK
a.12	...un certain nombre de pairs d'être impliqué dans l'échange d'un fichier simultanément. (multisource ou non)	0	Un seul, pas de composant permettant le multi-source.
a.13	Le système doit être adapté au nombre de personnes avec lesquelles on envisage d'échanger (cercle restreint, élargi, ou sans limite à priori)	0	Cercle de moyenne taille
a.14	...de minimiser les besoins en énergie sur les systèmes Android	0	format des données d'échange avec le serveur LINK
a.15	...de s'adapter en fonction de la charge	1	milite pour l'utilisation de plateforme PAAS notamment
a.16	...d'être disponible	1	milite pour une plateforme type PAAS + critère à définir pour chaque système particulier, notamment le temps de disponibilité devra être mesuré
a.17	...d'être tolérant aux pannes	1	Le choix de l'architecture matérielle et du support de l'application sera influencée. Milite pour l'adoption d'une plateforme PAAS.
a.18	...d'être récupérable,relançable	1	Milite pour l'adoption d'une plateforme PAAS
a.19	Une documentation explicative quant à l'usage devra être fournie	0	N/A à ce niveau
a.20	...de fournir une log des pannes devra être disponible à l'analyse	1	Ajouter un composant de journalisation au serveur LINK
a.21	Faire en sorte que le produit soit modifiable sans effets de bords	1	milite pour l'utilisation de plateforme PAAS notamment
a.22	...d'être mis à jour de manière transparente	1	Milite pour l'adoption d'une plateforme PAAS + Utilisation de standards

b.1	…de lister et choisir les fichiers à partager.	0	Fonctionnalité déjà prise en compte au niveau noyau (bien qu' à adatper)
b.2	…de lister et administrer les utilisateurs autorisés à lister les fichiers disponibles.	0	Fonctionnalité déjà prise en compte au niveau noyau
b.3	…d'exposer ses fichiers aux tiers.	0	Fonctionnalité déjà prise en compte au niveau noyau
b.4	…de limiter l'accès aux seuls tiers autorisés/Contrôler l'accès aux fichiers.	0	Fonctionnalité déjà prise en compte au niveau noyau
b.5	…d'échanger ses informations de connexion par QR code	0	Fonctionnalité déjà prise en compte au niveau noyau
b.6	…de lister et récupérer les fichiers disponibles d'un pair.	0	Fonctionnalité déjà prise en compte au niveau noyau (bien qu' à adatper)
b.7	…de savoir qui accède aux fichiers et quand.	0	Fonctionnalité déjà prise en compte au niveau noyau
b.11	Initier une communication avec un contact hors réseau local	1	Serveur Link
	exigence héritée		
	exigence non pertinente au niveau variant.		
	exigence satisfaite au niveau noyau et/ou non applicable au variant		

Tableau 51 – Exigences utilisationNomade.

indice	Exigence	Impacte sur l'architecture	Commentaire
a.1	Le système devra permettre de partager des données en économisant les éventuelles ressources serveurs	1	Nettoyeur BDD
a.3	Il ne devra pas permettre à une tierce personne de savoir ce qui est échangé par qui	1	Chiffrement/Déchiffrement ID IP
a.5	Garantir une utilisation agréable et simple	1	configuration simple de la redirection de port sur l'éventuel routeur - module UPNP
a.6	être ouvert à d'éventuelles évolutions quant aux environnements des clients participant aux échanges	1	standards adoptés en terme de serveurs coté LINK : WEB serveur et REST serveur
a.7	garantir une communication possible	1	Utilisation d'un PAAS
a.11	le temps pris pour effectuer chaque tâches ne doit pas dépasser un temps raisonnable	0	charte des temps, coté serveur LINK
a.12	Nombre de pair impliqué dans l'échange d'un fichier simultanément	0	Un seul, pas de composant permettant le multi-source.
a.13	Nombre de personnes avec lesquelles on envisage d'échanger	0	Cercle de moyenne taille
a.14	Le système devra être le moins gourmand possible en terme d'énergie sur les systèmes Android	0	format des données d'échange avec le serveur LINK
a.15	Adapter le système en fonction de la charge	1	milite pour l'utilisation de plateform PAAS.. Notamment
a.16	Le système devrait être disponible	1	milite pour google engine + critère à définir pour chaque système particulier, notamment le temps de disponibilité devra être mesuré
a.17	Tolérant aux pannes	1	Le choix de l'architecture matérielle et du support de l'application sera influencée milite pour google engine
a.18	Récupérable,relançable	1	milite pour google engine
a.20	Une log des pannes devra être disponible à l'analyse	1	ajouter un module de log coté serveur LINK
a.21	Faire en sorte que le produit soit modifiable sans effets de bords	1	milite pour l'utilisation de plateform PAAS.. Notamment
a.22	Le produit devra pouvoir être mis à jour de manière transparente	1	Milite pour l'adoption d'une plateforme PAAS + Utilisation de standard
b.11	Initier une communication avec un contact hors réseau local	1	Serveur Link
	exigence héritée		
	exigence non pertinente au niveau variant.		
	exigence satisfaite au niveau noyau et/ou non applicable au variant		

Tableau 52 – Exigences utilisationNomade impactantes.

Les exigences du variant utilisationNomade ne traitent que de la connexion Wifi entre pairs. Les exigences de sécurité liées à de telles communications sont incluses dans les exigences de sécurité au niveau noyau. Certaines exigences étaient déjà présentes au niveau noyau mais ont un sens ou une application particulière au niveau variant.

Caractèristiques / Sous caractèristiques ISO 25010	Besoin	priorité assignée	Attribut de qualité mesuré	objectif valué
1.c	b.11	5	possibilité offerte	attribut binaire : 1
2.a	a.11	5	temps de réponse	à définir.
2.b	a.1	5	nombre d'échanges, de demandes de traitements et volumes de données échangées et hébergées	à définir selon chaque produit de la famille
2.b	a.14	5	nombre d'échanges, de demandes de traitements et volumes de données échangées et hébergées	à déterminer au niveau système
2.c	a.12	3	Nombre de personnes	1
	a.13	3	Nombre de personnes	cercle de moyenne taille
	a.15	5	Adoption de moyens techniques garantissant une montée en charge.	attribut binaire : 1
4.e	a.5	3	Existence d'une charte visuelle respectant les bonnes pratiques en matière d'IHM	attribut binaire : 1
5.b	a.7	3	Adoption de moyens techniques garantissant une disponibilité du service.	attribut binaire : 1
5.b	a.16	3	Adoption de moyens techniques garantissant une disponibilité du service.	attribut binaire : 1
5.c	a.17	3	Adoption de moyens techniques adaptés	attribut binaire : 1
5.d	a.18	3	Adoption de moyens techniques garantissant une tolérence aux pannes.	attribut binaire : 1
6.a	a.3	5	coté serveur, absence de moyens techniques pour garder une telle trace, coté client sécurisation des fluxs de data	attribut binaire : 1, 1
7.b	a.6	4	Utilision de protocoles d'échanges standard	attribut binaire : 1
7.c	a.20	3	Présence d'un composant de journalisation.	attribut binaire : 1
7.d	a.21	3	Adoption de moyens techniques adaptés	attribut binaire : 1
8.c	a.22	3	Adoption de moyens techniques adaptés	attribut binaire : 1

Tableau 53 – Modèle de qualité utilisationNomade.

III.4.1.3 **Variant partageGrandNombre**

Le variant partageGrandNombre permet un partage de fichier entre un grand nombre d'utilisateurs, éventuellement en accédant à un même fichier depuis plusieurs sources.

indice	Exigence Le système devra pemettre …	Impacte sur l'architecture	Commentaire
a.1	...de partager des données en économisant les éventuelles ressources serveurs	0	
a.2	...de partager du contenu de manière décentralisée : pour que les données restent celles des utilisateurs	0	Justifie aussi l'utilisation d'une API pour client P2P

a.3	...d'interdire à une tierce personne de savoir ce qui est échangé par qui	0	
a.4	...de répondre aux critères de temps de réponse Android	0	
a.5	...de garantir une utilisation agréable et simple	0	composant IHM devra intégrer les nouvelles foncitonnalités
a.6	...d'être ouvert à d'éventuelles évolutions quant aux environnements des clients participant aux échanges	0	prise en compte par ailleurs
a.7	...de garantir une communication possible	0	N/A dans ce cas, prise en compte par un autre variant
a.8	...de garantir la sécurité des données des utilisateurs	0	N/A dans ce cas, prise en compte par ailleurs
a.9	...de ne pas encourager les échanges illégaux	0	Exigence déjà prise en compte au niveau noyau
a.10	...d'être utilisable par la majorité des appareils	0	Exigence déjà prise en compte au niveau noyau
a.11	...de garantir aux utilisateurs qu'ils réaliseront chaque action en un temps maximum établi	0	charte des temps variant PartageEnGrandNombre
a.12	...un certain nombre de pairs d'être impliqué dans l'échange d'un fichier simultanément. (multisource ou non)	1	Plusieurs, composant BitTorrent Sync API mode
a.13	Le système doit être adapté au nombre de personnes avec lesquelles on envisage d'échanger (cercle restreint, élargi, ou sans limite à priori)	1	Grand nombre possible, composant BitTorrent Sync API mode
a.14	...de minimiser les besoins en énergie sur les systèmes Android	0	Exigence transversale, soin apporté au code
a.15	...de s'adapter en fonction de la charge	0	N/A dans ce cas
a.16	...d'être disponible	0	N/A dans ce cas, considération prise en compte par le système Android sous-jacent
a.17	...d'être tolérant aux pannes	0	N/A dans ce cas, considération prise en compte par le système Android sous-jacent
a.18	...d'être récupérable,relançable	1	N/A dans ce cas, se rapporte davantage à des ressources serveurs
a.19	Une documentation explicative quant à l'usage devra être fournie	0	N/A à ce niveau
a.20	...de fournir une log des pannes devra être disponible à l'analyse	0	Exigence déjà prise en compte au niveau domaine
a.21	Faire en sorte que le produit soit modifiable sans effets de bords	0	N/A dans ce cas, prise en compte par un autre variant et par le système android sous-jacent
a.22	...d'être mis à jour de manière transparente	0	N/A dans ce cas, prise en compte par un autre variant et par le système android sous-jacent
b.1	…de lister et choisir les fichiers à partager.	0	Fonctionnalité déjà prise en compte au niveau noyau (bien qu'à adapter pour ce variant)
b.2	…de lister et administrer les utilisateurs autorisés à lister les fichiers disponibles.	0	Fonctionnalité déjà prise en compte au niveau noyau

b.3	…d'exposer ses fichiers aux tiers.	0	Fonctionnalité déjà prise en compte au niveau noyau (bien qu'à adapter pour ce variant)
b.4	…de limiter l'accès aux seuls tiers autorisés/Contrôler l'accès aux fichiers.	0	Foncitonnalité déjà prise en compte au niveau noyau
b.5	…d'échanger ses informations de connexion par QR code	0	Fonctionnalité déjà prise en compte en partie au niveau noyau. Ajout de fonctionnalité pour y inclure une clé d'un fichier/répertoire
b.6	…de lister et récupérer les fichiers disponibles d'un pair.	0	Fonctionnalité déjà prise en compte en partie au niveau noyau. Ajout de fonctionnalité de récupération de clés.
b.7	…de savoir qui accède aux fichiers et quand.	0	Fonctionnalité déjà prise en compte au niveau domaine
b.12	… le partage de fichiers multi-source	1	Composant BitTorrent Sync API mode

exigence héritée
exigence non pertinente au niveau variant.
exigence satisfaite au niveau noyau et/ou non applicable au variant

Tableau 54 – Exigences partageGrandNombre.

indice	Exigence Le système devra pemettre …	Impacte sur l'architecture	Commentaire
a.2	...de partager du contenu de manière décentralisée : pour que les données restent celles des utilisateurs	0	Justifie aussi l'utilisation d'une API pour client P2P
a.4	...de répondre aux critères de temps de réponse Android	0	
a.11	...de garantir aux utilisateurs qu'ils réaliseront chaque action en un temps maximum établi	0	charte des temps variant PartageEnGrandNombre
a.12	...un certain nombre de pairs d'être impliqué dans l'échange d'un fichier simultanément. (multisource ou non)	1	Plusieurs, composant BitTorrent Sync API mode
a.13	Le système doit être adapté au nombre de personnes avec lesquelles on envisage d'échanger (cercle restreint, élargi, ou sans limite à priori)	1	Grand nombre possible, composant BitTorrent Sync API mode
a.14	...de minimiser les besoins en énergie sur les systèmes Android	0	Exigence transversale,
b.12	… le partage de fichiers multi-source	1	composant BitTorrent Sync API mode

exigence héritée

Tableau 55 – Exigences impactantes partageGrandNombre.

Caractèristiques / Sous caractèristiques ISO 25010	Besoin	priorité assignée	Attribut de qualité mesuré	objectif valué
1.c	a.2	5	Adoption de moyens tehcniques le permettant.	attribut binaire : 1
	a.12	5	Nombre de personnes	>1
	a.13	3	Nombre de personnes	à définir.
	b.12	5	Nombre de personnes	>1
2.a	a.4	5	temps de réponse	<1s
2.a	a.11	5	temps de réponse	à définir.
2.b	a.14	5	nombre d'échanges, de demandes de traitements et volumes de données échangées et hébergées	à déterminer au niveau système

Tableau 56 – Modèle de qualité partageGrandNombre.

III.4.2 FQM

Comme précédemment pour le noyau nous allons procéder au raffinement fonctionnel. Cependant nous nous attacherons à ne considérer que les exigences strictement liées au variant pour les regrouper en ensembles de fonctionnalités. Ce sont ces ensembles qui justifient l'existence du variant.

Je commence par effectuer un raffinement fonctionnel, où je liste toutes les fonctionnalités attendues. Certaines d'entre elles sont déjà prises en compte au niveau domaine. Elles sont indiquées en orange. Pour le tableau FQM, elles sont encore indiquées, (mais elles ne se reflèteront pas plus tard sur l'architecture préliminaire, ni sur l'architecture de qualité des variants, les composant les prenant en compte ayant déjà été introduits au niveau domaine).

III.4.2.1 **Variant connexionLocale**

classe de fonctionnalité	fonctionnalité	Utilisateur	Commentaire
Connexion locale	lister les réseaux Sans fils disponibles	partageur/pair	
	Sélectionner l'un d'entre eux pour s'y connecter	partageur/pair	
	Créer un réseau Wifi (HotSpot)	partageur/pair	b.9 découle de a.7
	Créer un réseau Wifi P2P	partageur/pair	b.10 découle de a.7
	Afficher un QR code contenant les informations du réseau créé	partageur/pair	QR code introduit au niveau domaine
	Scanner un QR code affiché par un pair	partageur/pair	
	fonction induite par une exigence héritée		
	Fonction remplie par un composant déjà introduit au niveau système		

Tableau 57 –Raffinement fonctionnel connexionLocale.

fonction	Caractérisitique / Sous caractéristique	Attribut de qualité mesuré [paramètre]	Domaine du paramètre
lister les réseaux Sans fils disponibles	1.c, 4.c,4.d	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 3 s
Sélectionner l'un d'entre eux pour s'y connecter	1.c, 4.c,4.d	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 2 s
Créer un réseau Wifi (HotSpot)	1.c, 4.c,4.d,5.b	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 5 s
Créer un réseau Wifi P2P	1.c, 4.c,4.d,5.b	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 5 s
Afficher un QR code contenant les informations du réseau créé	1.c, 4.d, 4.c	Accès aux fichiers restreint aux seules personnes autorisées [présence d'un composant de restriction d'accès]	≤ 5 s
Scanner un QR code affiché par un pair	1.c, 4.d, 4.c	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 5 s
	fonction induite par une exigence héritée		
	Fonction remplie par un composant déjà introduit au niveau système		

Tableau 58 – FQM connexionLocale.

III.4.2.2 **Variant utilisationNomade**

classe de fonctionnalité	fonctionnalité	Utilisateur	Commentaire
Utilisation nomade	Obtenir un ID unique	partageur/pair	
	S'inscrir en ligne	partageur/pair	
	Récupérer l'IP:port d'un contact	partageur/pair	
	Choix du port de l'application	Partageur	UPnP introduit par a.5
	Créer un QR code contenant des informations utiles	partageur/pair	QR code introduit au niveau domaine
	Traiter un QR code	partageur/pair	
	fonction induite par une exigence héritée		
	Fonction remplie par un composant déjà introduit au niveau système		

Tableau 59 - Raffinement fonctionnel utilisationNomade.

fonction	Caractérisitique / Sous caractéristique	Attribut de qualité mesuré [paramètre]	Domaine du paramètre
Obtenir un ID unique	1.c, 4.c,4.d	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 2 s
S'inscrir en ligne	1.c, 4.c,4.d	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 5 s
Récupérer l'IP:port d'un contact	1.c, 4.c,4.d	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 5 s
Choix du port de l'application	4.c	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 5 s
Créer un QR code contenant des informations utiles	1.c, 4.d, 4.c	Accès aux fichiers restreint aux seules personnes autorisées [présence d'un composant de restriction d'accès]	≤ 5 s
Traiter un QR code	1.c, 4.d, 4.c	Temps pris par l'utilisateur pour lancer une tâche.[valeur de délai maximum]	≤ 5 s
	fonction induite par une exigence héritée		
	Fonction remplie par un composant déjà introduit au niveau système		

Tableau 60 - FQM utilisationNomade.

III.4.2.3 **Variant partageGrandNombre**

classe de fonctionnalité	fonctionnalité	Utilisateur	Commentaire
PartageGrandNombre	Créer un identifiant unique (clé) par fichier	Hôte	
	Créer un identifiant unique par répertoire	Hôte	
	Récupérer la liste des hôtes d'un fichier	pair	
	Choix des sources pour une clé	pair	fonction héritée
	Récupérer la liste des clés disponibles chez un hôte choisi	pair	
	Créer un QR code contenant des informations utiles	hôte/pair	fonction héritée
	Traiter un QR code	hôte/pair	fonction héritée
	fonction induite par une exigence héritée		

Tableau 61 - Raffinement fonctionnel partageGrandNombre.

fonction	Caractérisitique / Sous caractéristique	Attribut de qualité mesuré [paramètre]	Domaine du paramètre
Créer un identifiant unique (clé) par fichier	1.c		
Créer un identifiant unique par répertoire	1.c		
Récupérer la liste des hôtes d'un fichier	1.c		
Choix des sources pour une clé	1.c		
Récupérer la liste des clés disponibles chez un hôte choisi	1.c		
	fonction induite par une exigence héritée		

Tableau 62 – FQM partageGrandNombre.

III.4.3 Architecture préliminaire du variant

Les composants grisés indiquent ceux ne faisant pas partie du variant, indiqués pour remettre les composants du variant dans le contexte d'une architecture utilisant le variant. Les composants verts sont les composants du variant dont l'existence sera justifiée par un élément du FQM, hors les composants déjà introduits au niveau domaine.

III.4.3.1 Variant connexionLocale

Si l'on se base sur les seules fonctionnalités propres au variant, il ne s'agit que de permettre à l'utilisateur de se connecter à un réseau.

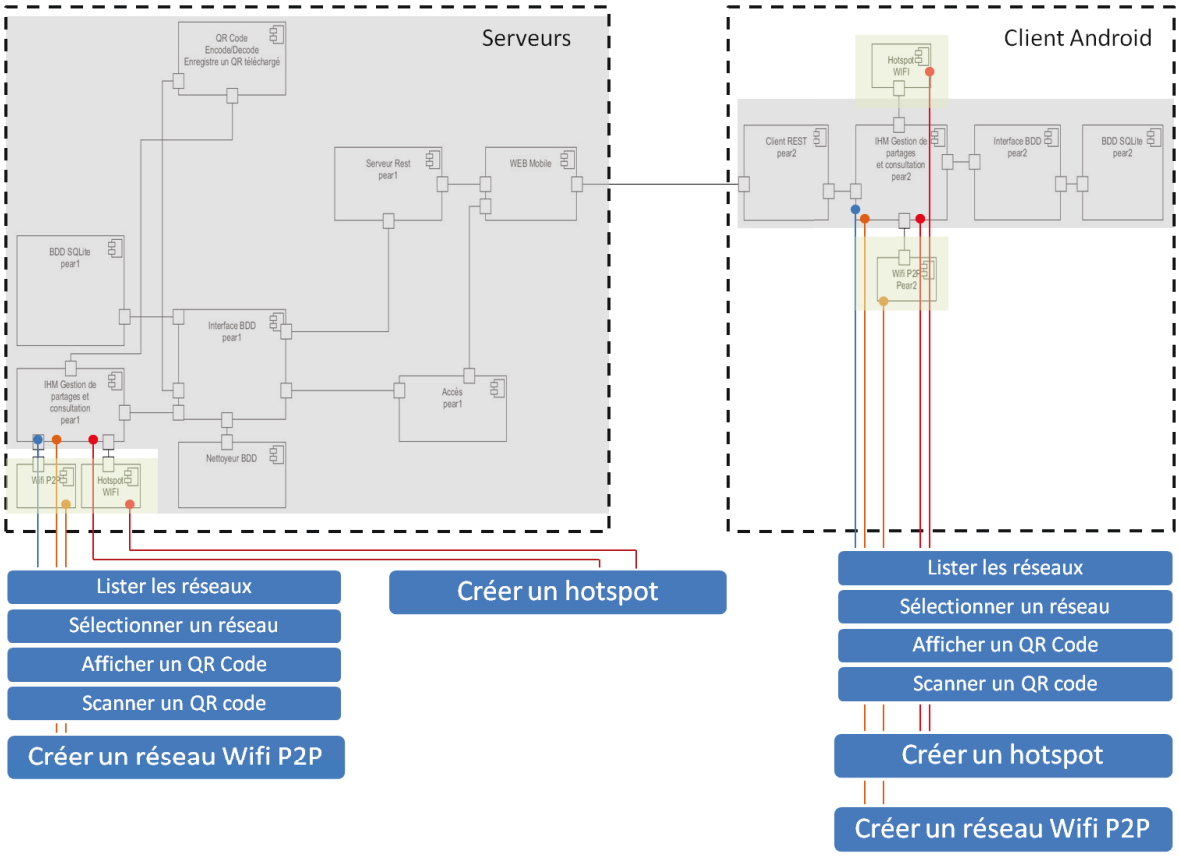


Figure 26 – Architecture préliminaire – connexionLocale.

III.4.3.2 Variant utilisationNomade

L'architecture préliminaire ne tenant pas en compte les exigences de qualité, seules les exigences fonctionnelles influencent cette architecture. On n'y inclut donc pas encore les composants permettant l'UPnP, le chiffrement des IP, la purge de la base de données

régulière. Le composant REST côté serveur LINK est déjà présent puisque nous devons interfacer avec des clients REST.

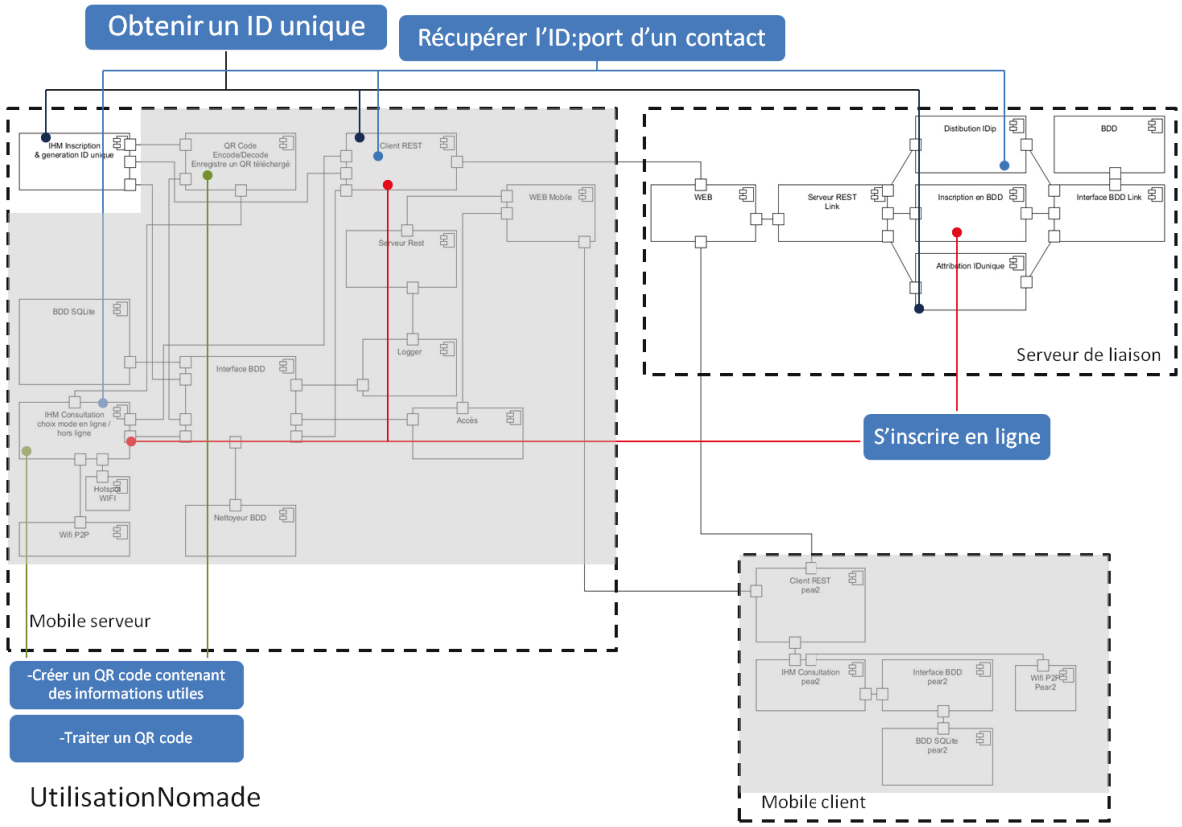


Figure 27 - Architecture préliminaire utilisationNomade.

III.4.3.3 **Variant partageGrandNombre**

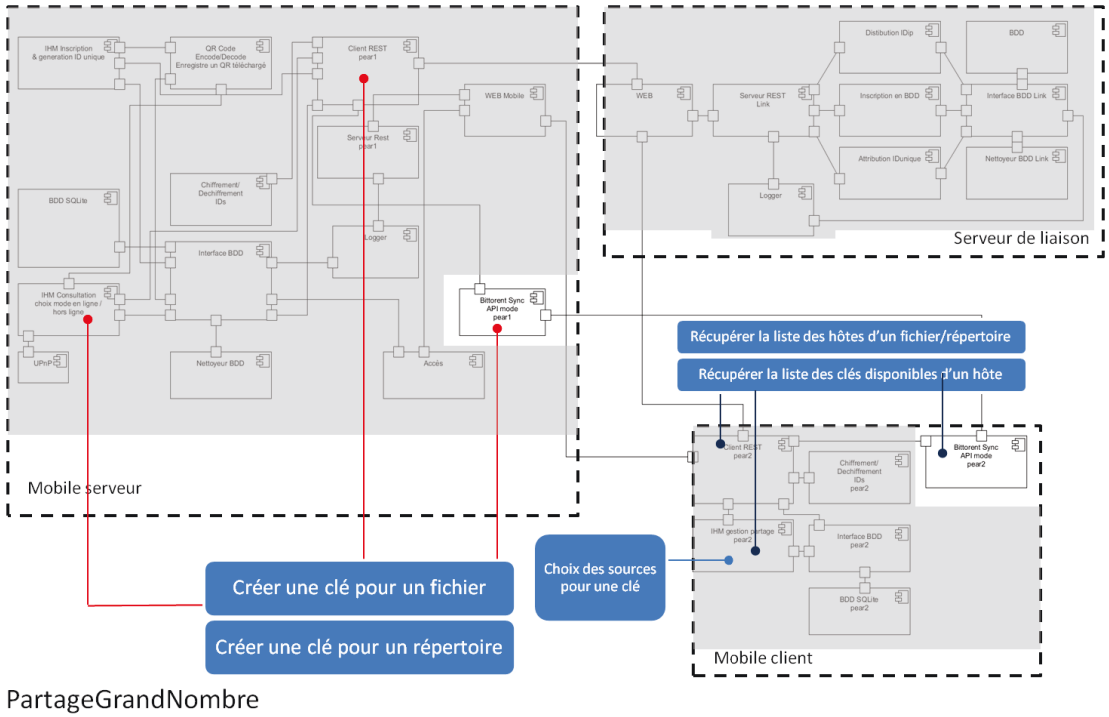


Figure 28 - Architecture préliminaire partageGrandNombre.

III.4.4 Le modèle de qualité de l'architecture du variant

L'AQM reprend toutes les exigences non fonctionnelles non attachées à des fonctions. Cependant, elles peuvent aussi impacter les architectures précédentes. Ici, nous nous situons dans le cadre plus restreint du variant. Il faut essayer de retrouver les exigences de qualité associées directement à celui-ci. Nous ne sommes toujours pas au niveau système, d'où une imprécision des paramètres et critères, qui restent toujours généraux.

III.4.4.1 **Variant connexionLocale**

Dans le cas du variant BitPool, aucune exigence de qualité strictement liée à celui-ci n'est exprimée. Toutefois, certaines exigences exprimées au niveau domaine auront une incidence. Il convient maintenant d'appliquer les exigences héritées afférant au variant connexionLocale. L'exigence non fonctionnelle a.7 (garantir une communication possible) a un impact architectural. Cette exigence déjà exprimée au niveau domaine se traduit ici par l'ajout de deux fonctionnalités b.10 et b11 (création d'un hotspot et connexion wifi P2P). Cela ajoute à la classe de fonctionnalité Connexion Locale, les fonctions « créer un réseau

Wifi HotSpot », « créer un réseau Wifi P2P ». Cette exigence nous amène encore à introduire une nouvelle exigence non fonctionnelle qu'elle rend nécessaire : l'exigence a.5. Celle-ci est sans impact architectural au niveau variant, toutefois elle amène deux fonctionnalités : « Afficher un QR code » et « scanner un QR code affiché par un pair ». Les exigences a.12 et a.13 n'ont aucune incidence sur l'architecture, elles n'entrent pas en conflit avec la solution proposée par le variant. Le variant hérite encore des exigences a.3, a.4, a.10, a.11 et a.14. Elles ont une influence sur l'implémentation du variant.

Besoins non fonctionnels liés à l'architecture	Caractérisitique / Sous caractéristique	Attribut de qualité mesuré [paramètre]	Domaine du paramètre	Solution architecturale
a.3	6.a	Chiffrement des communications entre pairs	–	–
a.4	2.a, 3.a	Temps de réponse	<1s	–
a.7	5.b	Disponibilité de multiple composants de communication	–	HotSpot Wifi et WifiP2P
a.11	2.a, 4.c	Conformité à une charte des temps ConnexionLocale	–	–
a.14	2.b,	Soin apporté à l'optimisation du code pour limiter le temps de traitement	–	–

Tableau 63 – AQM connexionLocale.

III.4.4.2 **Variant UtilisationNomade**

Besoins non fonctionnels liés à l'architecture	Caractérisitique / Sous caractéristique	Attribut de qualité mesuré [paramètre]	Domaine du paramètre	Solution architecturale
a.1	2.b	nombre d'échanges	_	Nettoyeur BDD
a.2	2.b, 6.a, 6.b	Aucune données hébergées	_	
a.3	6.a	Chiffrement des communications entre pairs	_	_
	6.a, 6.b	Aucune information stockée	_	_
		Chiffrement des IPs envoyées en réponse par le Serveur LINK	_	Chiffrement/Déchiffrement des IP
a.5	4.c	Présence de composant facilitant l'utilisation	_	UPNP
a.6	7.b	_	_	standard adoptés en terme de serveurs coté LINK : WEB serveur et REST serveur
a.7	5.b	Connexion réseaux multiples	_	Utilisation d'un PAAS
a.8	6.a, 6.b	absence de données hébergées	_	_
a.11	2.a, 4.c	Conformité à une charte des temps UtilisationNomade	_	_
a.14	2.b	Format des données échangées	_	_
a.15	5.b	Plateforme hébergeant le service scalable	_	Utilisation d'un PAAS
a.16	5.b	Plateforme hébergeant le service fiable [temps de disponibilité]	_	Utilisation d'un PAAS
a.17	5.c	Plateforme hébergeant le service redondante, Garantie d'un RTO[temps]	1h	Utilisation d'un PAAS
		Garantie d'un RPO[temps]	1h	Utilisation d'un PAAS
a.18	5.d	Garantie d'un RTO[temps]	1h	Utilisation d'un PAAS
a.20	7.c	Journalisation des erreurs [interval]	_	Module de log coté serveur LINK
a.21	7.d	Applications compartimentées	_	Utilisation d'un PAAS
a.22	8.c	Application versionnables	_	Utilisation d'un PAAS

exigence héritée et enforced au niveau variant

exigence satisfaite au niveau noyau et/ou non applicable au variant

Tableau 64 – AQM utilisationNomade.

III.4.4.3 **Variant partageGrandNombre**

Besoins non fonctionnels liés à l'architecture	Caractérisitique / Sous caractéristique	Attribut de qualité mesuré [paramètre]	Domaine du paramètre	Solution architecturale
a.2	**2.b,6.a,6.b**	_	_	Composant BitTorrent Sync API mode
a.4	2.a, 3.a	Réactivité[temps de réponse]	<1s	_
a.11	2.a, 4.c	Conformité à une charte des temps PartageGrandNombre	_	_
a.12	2.c	_	_	Composant BitTorrent Sync API mode
a.13	2.c	_	_	Composant BitTorrent Sync API mode
a.14	2.b	Soin apporté à l'optimisation du code pour limiter le temps de traitement	_	_

Tableau 65 – AQM partageGrandNombre.

III.4.5 L'architecture de qualité des variants

Nous allons représenter l'architecture prenant en compte l'ensemble des exigences de chacun des variants. En reprenant l'architecture préliminaire, en lui ajoutant les composant répondant aux exigences établies plus haut, on obtient l'architecture de variant de qualité. De nouvelles exigences ont été introduites depuis l'élaboration de l'architecture préliminaire. Elles se traduisent ici par l'ajout de composants. En reprenant la même méthode que pour le niveau domaine, voici l'architecture de variant de qualité et les exigences qui les justifient :

III.4.5.1 **Variant connexionLocale**

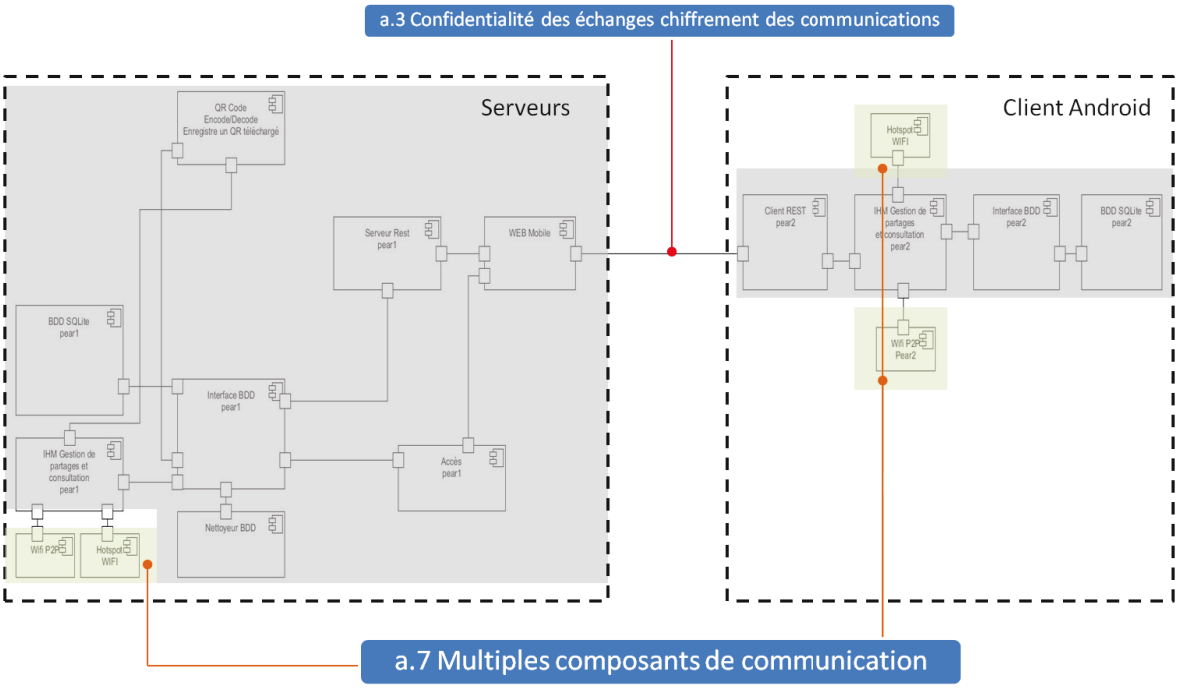


Figure 29 – Architecture de qualité connexionLocale.Variant utilisationNomade

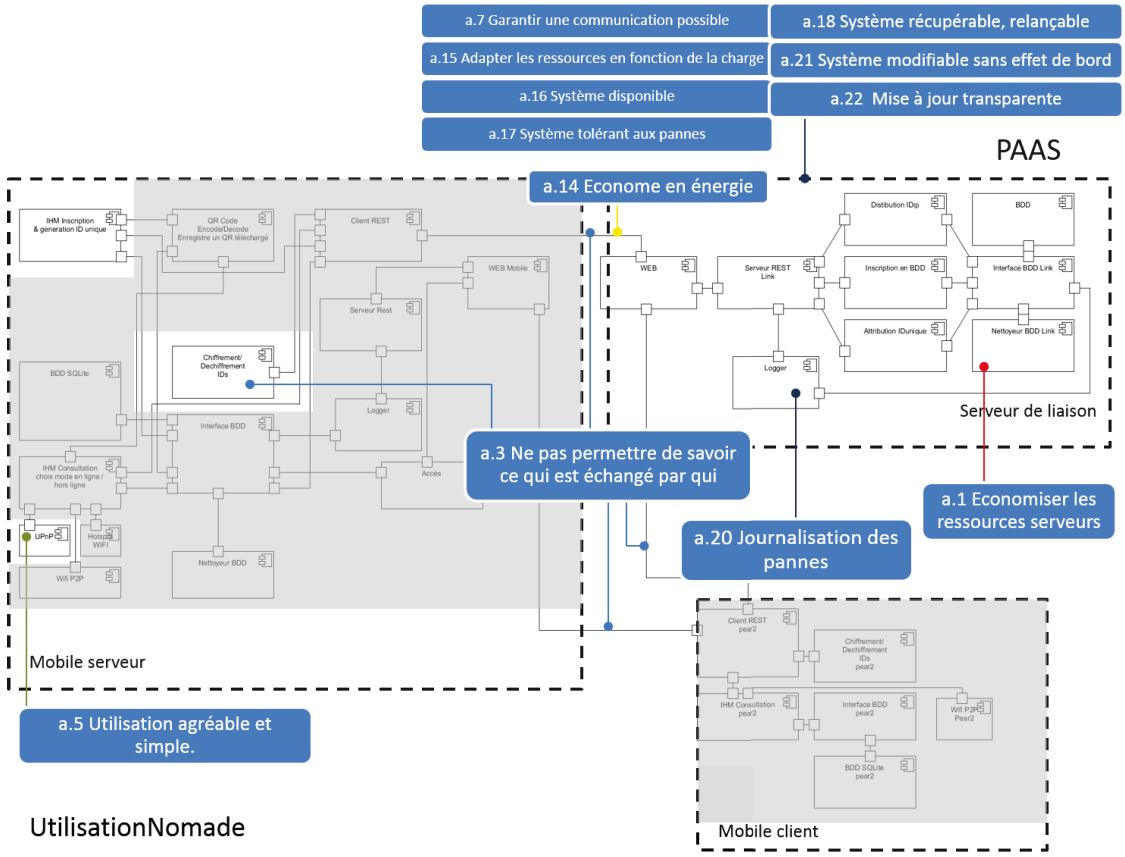


Figure 30 – Architecture de qualité utilisationNomade.

III.4.5.2 **Variant partageGrandNombre**

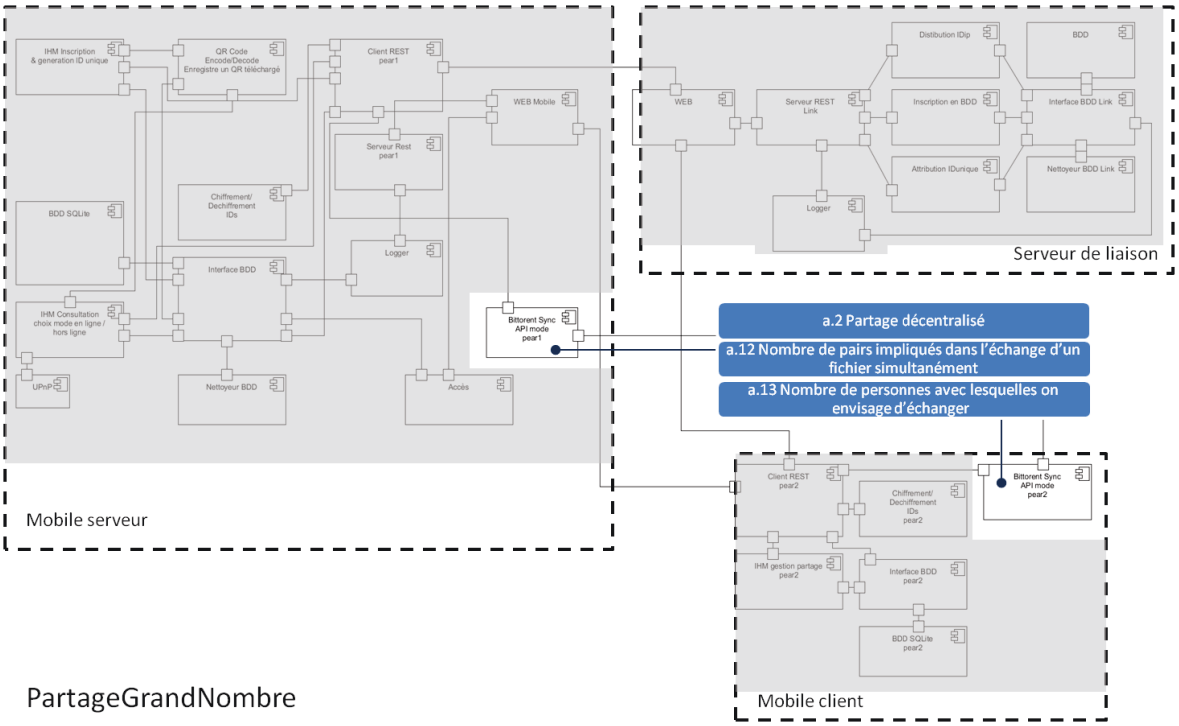


Figure 31 - Architecture de qualité partageGrandNombre.

III.5 Compromis de qualité

De nombreux ouvrages traitent de la gestion des compromis de qualité. Il ne s'agit pas ici de reprendre les différentes approches possibles. Nous n'allons que rappeler certaines questions que soulève ce sujet.

Nous avons obtenu une architecture de noyau de qualité. Il faut toutefois noter que cette architecture n'est pas tout à fait réaliste. Il faudrait encore conditionner l'adoption des composants (ou l'ajout de nouvelles exigences fonctionnelles à des composants) issus des exigences non-fonctionnelles par leur impact sur d'autres critères :

- des critères de qualité. Améliorer la maintenabilité par l'ajout d'un composant de journalisation ne pénalise-t-il pas trop la performance globale du système en ajoutant un temps de traitement ?

- des critères de productivité : le temps de développement, les ressources nécessaires au développement des composants contribuant à l'amélioration de la qualité sont-ils trop important dans le cadre du projet en cours ?

Il est difficile de répondre à ces questions au niveau domaine, et le report de la décision d'adoption ou non des composants au moment de l'implémentation d'un système particulier est possible. Il est d'autant facilité que le lien de traçabilité entre les exigences non fonctionnelles et les composants qui les garanti a été documenté précédemment. Il est toutefois possible d'éliminer certains composants au niveau domaine. Par exemple, pour ma part, je ne garderai pas l'ajout de l'exigence fonctionnelle "Stocker les messages d'erreurs relatifs aux pannes [critères]" sur le composant logger. Il n'y pas de gain réel, l'architecture noyau ne traitant que des appareils mobiles, puisqu'il est peu probable que les utilisateurs du système consultent cette log en cas de problème. Il faut davantage se concentrer sur les exigences de fiabilité du système.

Ces composants peuvent être choisis de manière optionnelle lors de la configuration du système particulier. Ceci pourra être modélisé par un feature diagram.

IV Gestion de la variabilité

La première partie du travail effectué consistait à déterminer les composants réutilisables (noyau et variants) et à les documenter, notamment en mettant en évidence le lien de traçabilité entre exigences et composants. On peut considérer la première partie comme la constitution d'une bibliothèque logicielle. Créer une application revient à implémenter les composants nécessaires de la bibliothèque pour construire l'application. L'objectif des SPL est la recherche d'économies. La réutilisation de composant logiciels en est le principal levier. Pour être efficace, cette réutilisation doit être orchestrée. C'est l'objet de ce qui suit.

Il s'agit maintenant d'établir une méthode de dérivation afin d'obtenir des produits valides : des produits qui respectent les conditions de composition que nous avons. Les recherches de Levy *et al*. utilisées ici s'arrêtent à l'établissement de l'architecture de référence. J'ai recherché une méthode permettant de dériver un produit à partir de ce que nous avons. La dérivation d'un produit passe par la gestion de la variabilité. Il s'agit d'exprimer la ligne de produits en terme de composants qui sont soit obligatoires, soit optionnels. Certains choix en impliquent ou en excluent d'autres. Ces contraintes doivent aussi être exprimées. Plusieurs écoles coexistent et aucun réel consensus n'a été trouvé quant à la meilleure méthode à employer. Il s'agira ici de choisir celle qui semble être une suite logique à la méthode donnée par N. Levy *et al.* Dans le sujet de départ il m'était demandé d'automatiser autant que possible le développement de la ligne de produits. Cela devait passer par l'utilisation de l'environnement de développement de ligne de produits logiciels SEQUOIA du CEA list et son modèle de contraintes. Ceci ne fut pas possible, je n'ai pas pu avoir accès à cet environnement.

Puisque nous partons d'une bibliothèque de composants, dont certains entreront obligatoirement dans la composition des applications dérivées (noyau), et qu'il s'agit finalement de choisir les composants voulus en fonction des exigences qu'ils satisfont, idéalement, le mode d'expression de la variabilité doit permettre de représenter les composants de notre SPL, d'indiquer les composants obligatoires, les composants optionnels et les contraintes qui existent entre ces composants.

J'ai étudié les différentes méthodes existantes et cherché celle qui se rapproche le plus de la démarche proposée par N. Levy *et al.* Ces recherches furent longues, beaucoup d'articles existent sur le sujet. (Ils sont malheureusement davantage théoriques que

pratiques.) En revanche peu d'outils existent permettant d'appliquer facilement les méthodes recommandées. Dans leur grande majorité, les outils que j'ai pu trouver sont réalisés par des équipes de chercheurs et sont trop peu documentés. Leur développement est aussi souvent interrompu et les outils ne sont donc plus utilisables dans un environnement actuel. Cette partie fut très frustrante.

N.Levy *et al.* nous amènent à exprimer l'architecture de référence en termes d'addition de matrices. Le noyau et les variants regroupent un ensemble cohérent de fonctionnalités attendues par l'utilisateur. Ceci se rapproche de la démarche feature model abordé ci-après. Un feature model permet de modéliser la variabilité d'une ligne de produit puis de définir des compositions de features valides pour former un produit de la ligne de produit.

IV.1 Démarche

IV.1.1 Expression de la variabilité

IV.1.1.1 **Feature model.**

Le concept central du feature model est celui de feature, souvent traduit par le terme « aspect » en français. Il est d'abord définit par Kang *et al.* (9) comme une qualité, une caractéristique ou un aspect distinctif d'un système perçu par l'utilisateur. La définition fut adaptée au domaine de la SPL par Hence Czarnecki et Eisenecker comme suit : une propriété d'un système pertinente pour une partie prenante utilisée pour saisir les points communs entre les membres d'une SPL ou les différencier. (10)

En s'appuyant sur le concept de feature, il est proposé d'utiliser un feature model pour représenter la variabilité dans une SPL. Un feature model est constitué d'un diagramme de feature (feature diagram) et d'informations additionnelles : des contraintes et des règles de dépendance entre features. Le feature diagram est une représentation graphique. Un arbre représentant l'organisation hiérarchique des fonctionnalités d'un produit représentées par des features dont la racine symbolise le produit complet, ou dans notre cas, la ligne de produit. On part de fonctionnalités à un haut niveau d'abstraction pour les décomposer en features plus affinées. Les relations entre les nœuds de l'arbre (les features) sont matérialisées par des arcs. Des contraintes sont indiquées sous forme d'annotations textuelles. La variabilité est décrite par différentes représentations. La présence ou l'absence d'une feature dans un produit est mise en avant par des features obligatoires (mandatory) ou

optionnelles (optional). Les features sont représentées par des rectangles tandis que leur variabilité est décrite par des éléments graphiques additionnels : souvent un cercle vide pour les features optionnelles, un cercle plein pour une feature obligatoire. Les features peuvent faire partie d'un groupe. Des opérateurs booléens sont utilisés pour indiquer les contraintes et dépendances entre features : le ou exclusif (XOR, un seul des enfants de ce groupe sera présent), inclusif (OR, au moins une des features de ce groupe sera présente), l'implication (requires, AND, si le nœud A est sélectionné, alors B aussi), l'exclusion (A AND NOT B, si le nœud A est sélectionné, B ne peut l'être). Ces contraintes guident l'utilisateur dans le choix des features d'un produit. XOR et OR seront représentés graphiquement alors que l'exclusion et l'implication sont en général représentées par des contraintes textuelles.

IV.1.1.2 **Extension du Feature Model**

IV.1.1.2.1 **Cahier des charges**

Le feature diagram est une manière concise de décrire la variabilité d'une SPL, les choix permis dans la composition des produits de la SPL. Toutefois certaines choses ne peuvent être exprimées telles quelles. Par exemple, il ne permet pas d'exprimer explicitement le lien de traçabilité entre un composant et l'exigence qui l'introduit. Le lien de traçabilité entre la fonctionnalité attendue et le composant reste malheureusement implicite, ce qui enlève l'outil de configuration des mains du client final. Il faut encore raisonner en termes de composants. A l'opposé des préoccupations, le lien entre ces composants et le code est aussi à définir. Des extensions ont été proposées par divers approches afin de rendre le feature model davantage expressif et de créer un pont entre les fonctionnalités attendues et le code. J'ai retenu ces idées :

- Il faut distinguer les features réalisées et celles qui ne le sont pas : soit parce qu'elles sont abstraites, soit parce qu'elles regroupent d'autres features (feature group). Ceci correspond notamment à la vue que nous avons de notre architecture de référence, avec un noyau et des variants composés de composants, des boîtes noires dont nous ne savons pas encore comment elles sont implémentées.

- J'organise mon feature diagram en trois niveaux d'abstraction. A un très haut niveau d'abstraction, je représente mon architecture de référence par un noyau et des variants. C'est un niveau accessible par le client où le futur utilisateur indique les exigences

qu'il a du futur produit. Ensuite, par raffinement, j'indique les composants qui les composent. C'est le niveau architectural, toujours indépendant de la plateforme technique d'implémentation qui sera choisie finalement. En termes de représentation UML cela correspondrait à un diagramme de composants. Enfin, au niveau technique, j'indique les éléments qui composent les composants. Si je décide de développer en langage orienté objet, ici ce seraient des classes. Ceci permet d'envisager de modéliser la variabilité d'un système qui pourrait être développé sur plusieurs plateformes.

- J'indique par des annotations l'exigence à l'origine des features.

Je n'ai pas trouvé d'outil adapté à ce que je voulais représenter ici. Je dois donc créer un métamodèle représentant ces notions et un éditeur graphique correspondant (un modeleur).

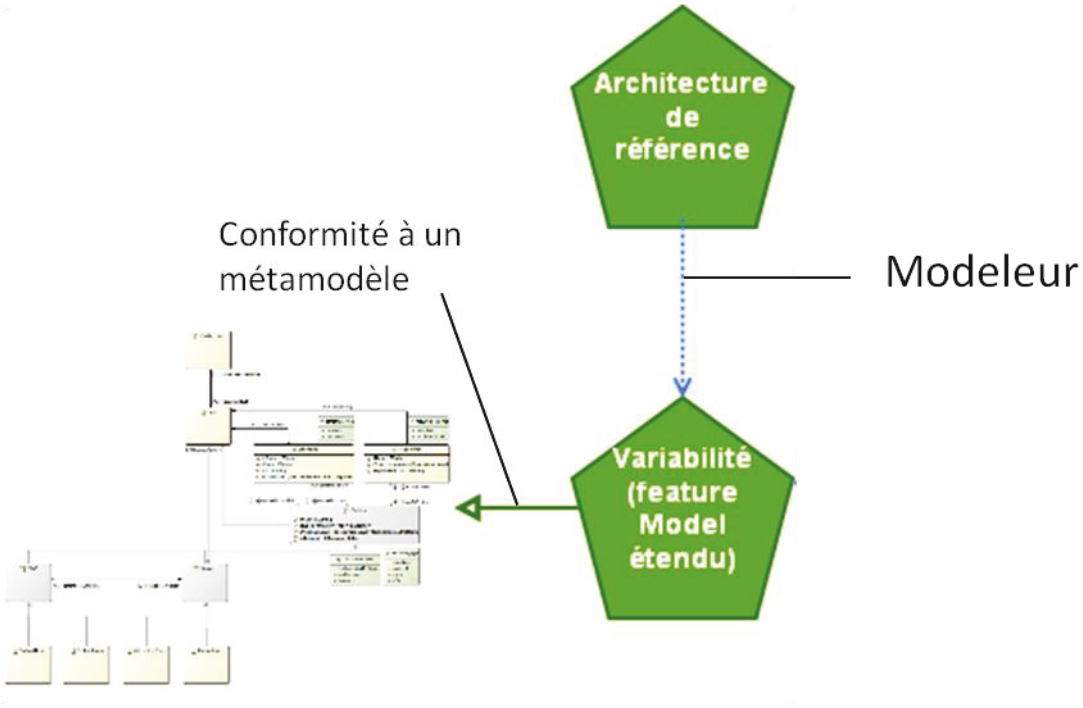


Figure 32 – Rôle du modeleur.

IV.1.2 Configuration

Après avoir exprimé la variabilité par un feature model, l'activité de configuration permet de composer un produit en sélectionnant les composants qui le constitueront conformément aux attentes que l'on a du produit final. Il me faut un outil pour sélectionner tel ou tel composant de mon feature model et répercuter automatiquement certains choix sur l'ensemble des features.

Je décide donc d'ajouter un attribut booléen isSelected à mes features de mon futur métamodèle. J'y exprimerai aussi certaines contraintes qui formeront la base de la logique de sélection. Le modeleur correspondant devra permettre d'implémenter la logique de sélection et de me fournir un modèle de pré-dérivation, qui représentera l'ensemble des features sélectionnées ou non. L'attribut isSelected sera représenté graphiquement.

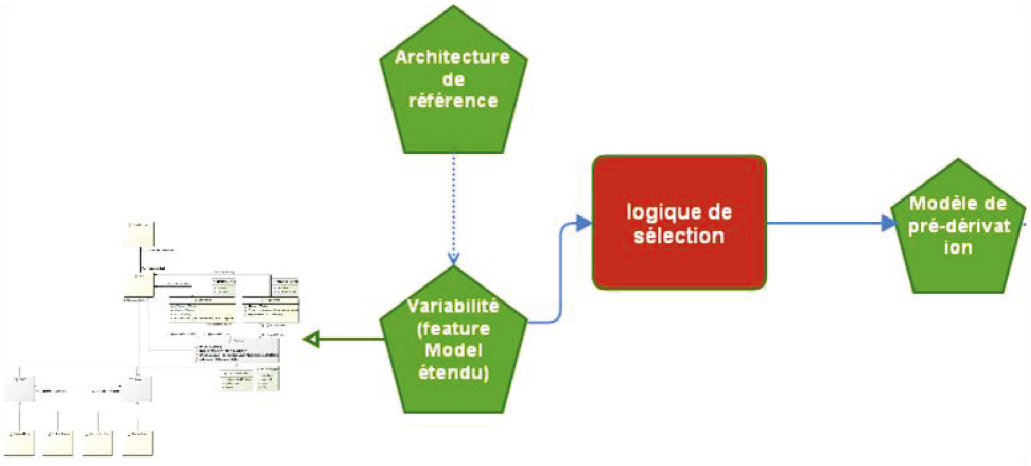


Figure 33 – Logique de sélection et modèle de pré-dérivation.

IV.1.3 Dérivation

L'activité de dérivation permet d'obtenir le modèle du produit final, ne gardant que les features sélectionnées. Le modeleur devra permettre de produire ce modèle sur la base du modèle de pré-dérivation.

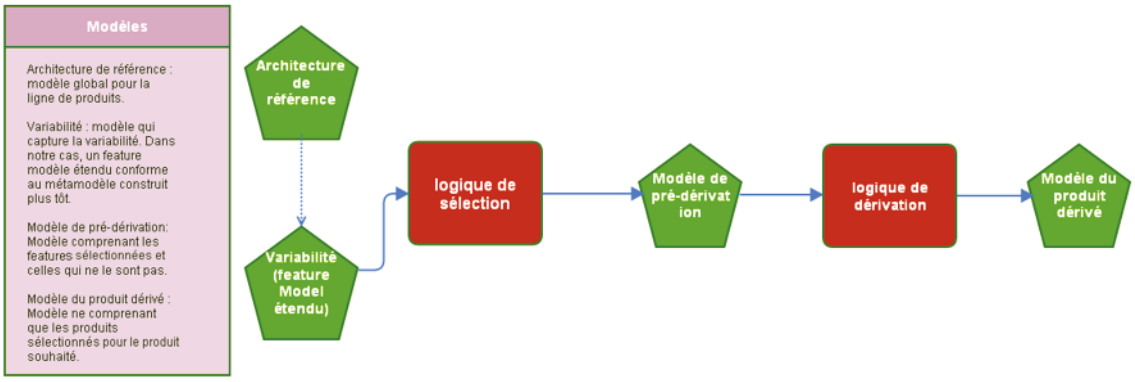


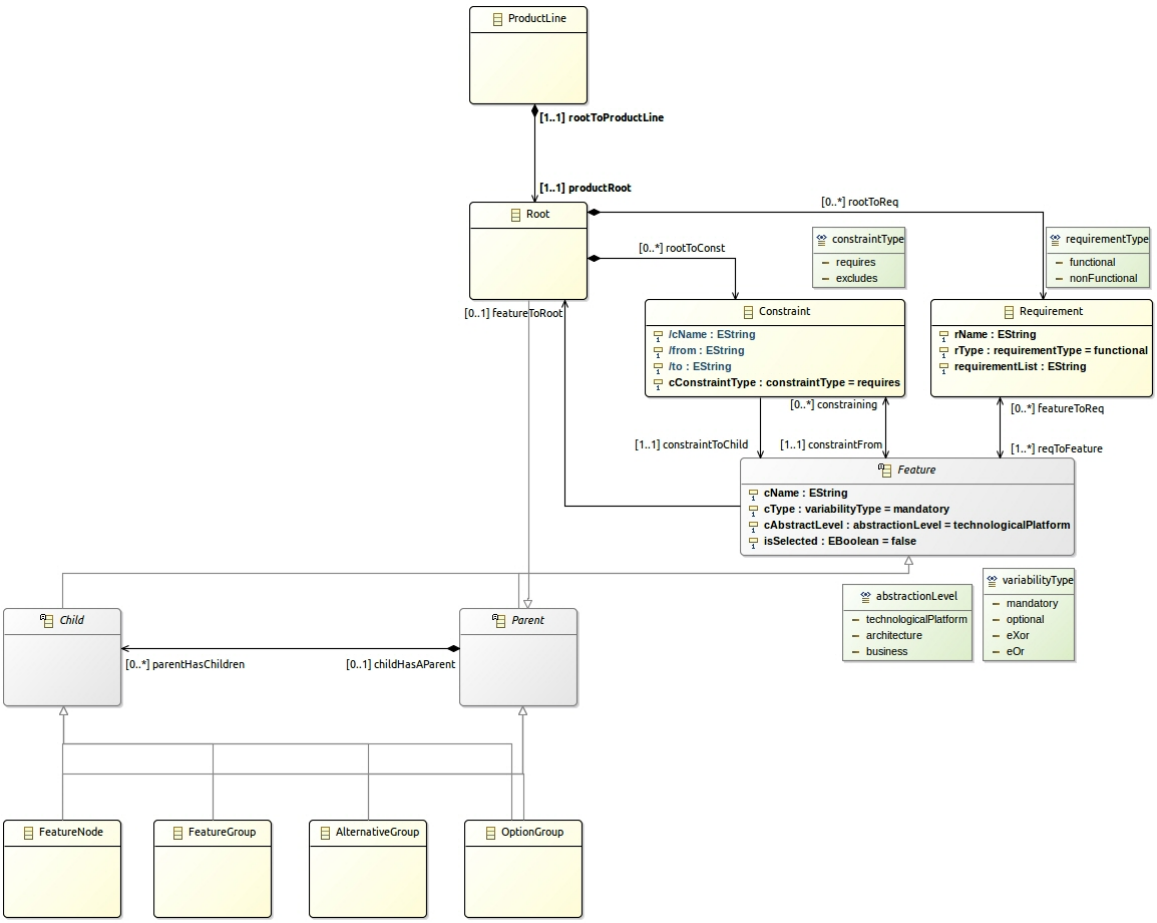
Figure 34- Logique de dérivation et modèle du produit dérivé.

IV.2 Réalisation

IV.2.1 Modélisation de notre Domain Specific Language (DSL)

Le monde de l'ingénierie logicielle a deux choix pour adopter une modélisation proche d'un domaine spécifique : les profils UML pour étendre le langage de représentation générique UML ou les Domain-Specific Modeling Languages (DSML) qui décrivent dans un métamodèle les concepts d'un domaine particulier et leurs relations. A l'aide d'un DSML il est possible d'instancier un modèle décrivant un cas de configuration possible du domaine auquel il s'applique. On obtient ici un Domain Specific Language (DSL) propre au domaine que l'on veut représenter. C'est cette dernière approche que j'ai choisie.

J'ai modélisé cette représentation sous Eclipse grâce au Modelling Framework et notamment au modeler Eclipse EcoreTools. Il m'a permis de définir un métamodèle Ecore pour mon Feature Model étendu. (Ecore est une spécification de méta-métamodèle. Par exemple, l'outil Papyrus, un modeler UML, est lui aussi conforme à ce méta-métamodèle) De manière générale, un modèle seul ne permet pas d'éviter certaines ambiguïtés. Il faut le compléter avec des annotations formelles. J'ai choisi le langage d'expression de contraintes OCL (Object Constraint Language, http://www.omg.org/spec/OCL/), notamment utilisé dans la définition du métamodèle UML. Il permet de décrire des invariants dans un modèle, sous forme de pseudo-code, des pré/post-conditions pour une opération, des expressions de navigation. Son implémentation dans l'environnement EMF (eclipse modelling framework) est possible grâce à l'outil OCLinEcore qui permet d'annoter un modèle Ecore. Le détail de cette réalisation ne serait qu'un pas à pas sans beaucoup d'intérêt. Voici une représentation du modèle final. Ce métamodèle a été obtenu après plusieurs itérations et ajustements suite à l'utilisation de ce métamodèle avec l'outil Sirius (introduit plus tard). Certains choix de modélisation et de relations ne sont là que pour faciliter la navigation dans ce modèle et son interrogation depuis l'outil Sirius.



La classe ProductLine est un exemple d'une adaptation faite à l'outil Sirius. Elle ne servira que de conteneur pour le reste. Elle ne pourra être représentée. J'ai dû ajouter cette classe pour représenter la racine du feature modèle. La ProductLine contient un et un seul Root. Cette classe, pour des raisons purement pratiques, hérite de la classe Parent pour pouvoir représenter de manière générique la relation Parent/enfant : parentHasChildren. La classe Parent hérite de Feature. Feature est une classe abstraite. Elle possède tous les attributs et relations communs à tout type de feature. J'ai indiqué quatre types de feature :

- la feature classique : FeatureNode ;

- le FeatureGroup représentera une feature dont la réalisation nécessite de multiple features, au moins deux ;

- l'AlternativeGroup est un groupe de features dont un seul des enfants peut être choisi. J'ai choisi cette représentation de la variabilité XOR pour des raisons pratiques : l'élaboration des contraintes dans le modèle puis la spécification de leurs représentations en est simplifiée ;

- l'OptionGroup représente une variabilité de type OR. Un de ses enfants au moins doit être sélectionné lors de la configuration d'un produit. Ces quatre classes héritent de Parent et Child qui héritent de Feature. Elles ne servent qu'à faciliter la navigation dans la hiérarchie de features.

Root contient deux dernières classes :

- Constraint, qui représente les contraintes de type excludes ou requires. Constraint est liée à Feature. La relation « constraining », indique qu'une feature va en contraindre une autre par le biais de la contrainte vers laquelle elle pointe. La relation « constraintToChild » indique sur quelle feature la contrainte va s'appliquer. « constraintFrom » n'est que l'opposée de « constraining ». Chaque contrainte est unique. Son identifiant est son nom, l'attribut dérivé cName, élaboré selon le schéma suivant : from_cConstraintType_to : « from » indique la feature à l'origine de la contrainte. Cet attribut de Constraint est lui-même un attribut dérivé, obtenu en interrogeant la relation « constraintFrom ». « cConstrainType » correspond au type de contrainte : soit requires (featureA AND featureB), soit excludes (featureA AND NOT featureB). L'exclusion mutuelle s'exprime par deux contraintes de type excludes en intervertissant la source et la destination de la contrainte. Enfin, « to » indique la feature sur laquelle la contrainte s'applique. Cet attribut dérivé est obtenu en interrogeant la relation « constraintToChild ».

- Requirement, qui représente les exigences à l'origine des features. Le lien de traçabilité est représenté par la relation « reqToFeature ».

La relation « featureToRoot » est redondante mais elle est très utile lors de l'interrogation du modèle, notamment depuis l'outil Sirius. Elle permet de gagner en temps de traitement lorsque l'on veut atteindre Root depuis ses enfants directs. Seuls les Features ayant cette relation sont interrogées. Sans cette relation, il aurait fallu interroger toutes les features sur leur relation « childHasAParent » et filtrer les résultats par type (type = Root. Exemple : ChildHasParent.oclIsType(Root), suite du query…)

Ce modèle, tel qu'il est ne permet pas d'exprimer l'ensemble des contraintes que je souhaite voir exister au sein de ce métamodèle. J'ai donc adjoint certaines contraintes OCL au métamodèle grâce à l'outil OCLinEcore. Les contraintes OCL sont à établir dans le contexte d'une classe. Il est ensuite possible d'interroger et de naviguer dans le modèle depuis ce point d'origine. Pour chaque contrainte il faut donc choisir le contexte le plus

approprié pour écrire des contraintes les plus simples possibles. Il s'agit principalement pour mon cas d'invariants. Voici l'ensemble des contraintes OCL. Les simples attributs et relations sont remplacés dans le code par […]. Les commentaires indiquent ce que je souhaite implémenter.

```
package extended_feature_Mmodel : extended_feature_Mmodel =
'http://www.example.org/extended_feature_Mmodel'
{
    class Root extends Parent
    {
        [...]

        invariant rootMustBeBusinessLevel:
            self.cAbstractLevel = abstractionLevel::business
        ;
        /*Je souhaite que les enfants directs de Root soient du niveau
        *d'abstraction Business (le plus haut niveau).
        */
        invariant rootsChildMustBeBusinessLevel:
            parentHasChildren->size()>0 implies
            parentHasChildren->select(cAbstractLevel =
                abstractionLevel::business)->size()
            = parentHasChildren->size()
        ;
        /* Je ne donne pas la liberté du choix du nom donné
        * à l'instance de Root.
        */
        invariant theNameIsRoot:
            self.cName = 'Root'
        ;
        /*Root hérite de feature. Toutefois, Root n'est pas
        * à sélectionner ou non. Il doit l'être et il est obligatoire.
        */
        invariant rootIsSelected:
            self.isSelected
        ;
        invariant rootIsMandatory:
            self.cType = variabilityType::mandatory
        ;
    }
    abstract class Child extends Feature
    {
        [...]
        /*Un enfant doit être d'un niveau d'abstraction égal ou inférieur
        * à celui de son parent.
        */
        invariant childLevelIsLowerOrEqualToParents_Architecture:
            not childHasAParent.oclIsUndefined()
            and childHasAParent.cAbstractLevel =
                abstractionLevel::architecture
            implies(
                cAbstractLevel = childHasAParent.cAbstractLevel
                or cAbstractLevel = abstractionLevel::technologicalPlatform
            )
        ;
        invariant childLevelIsLowerOrEqualToParents_Techno:
            not childHasAParent.oclIsUndefined()
```

```
            and childHasAParent.cAbstractLevel =
                abstractionLevel::technologicalPlatform
            implies(
                cAbstractLevel = childHasAParent.cAbstractLevel
            )
        ;
        /*Un enfant de Root, s'il est obligatoire (mandatory), doit être
        * sélectionné.
        */
        invariant mandatoryRootsChildMustBeSelected:
            childHasAParent.oclIsUndefined() /* dans ce cas c'est un enfant
                                de root… ce qui devrait être parent*/
            and cType = variabilityType::mandatory
            implies isSelected
        ;
        /*Les enfants obligatoires de features (au sens large)
        * sélectionnées doivent être sélectionnés.
        */
        invariant mandatoryChildrenOfSelectedOnesMustBeSelected:
            childHasAParent.isSelected
            and cType = variabilityType::mandatory
            implies isSelected
        ;
        /* Les enfants de features non sélectionnées ne doivent pas
        * être sélectionnés. */
        invariant childrenOfNotSelectedOnesMustNotBeSelected:
            (childHasAParent->size()=1 and (not
                childHasAParent.isSelected))
            implies isSelected = false
        ;
        /*Les enfants d'une FeatureNode ou FeatureGroup doivent être
        * optionnels ou obligatoires. Cela implique qu'ils ne peuvent
        avoir l'attribut eXor ou eOr que j'ai réservé pour les enfants des
        * AlternativeGroup et OptionGroup.*/
        invariant aFeatureOrFeatureGroupsChildrenMustBeMadatoryOrOptional:
            (childHasAParent->size()=1
                and (childHasAParent.oclIsTypeOf(FeatureGroup)
                        or childHasAParent.oclIsTypeOf(FeatureNode)
                    )
            )
            implies ( cType = variabilityType::mandatory
                        or cType = variabilityType::optional
                    )
        ;
    }
    abstract class Parent extends Feature
    {
        [...]
        /* Root hérite de Parent et donc de Feature indirectement.
        * Ceci m'oblige à ajouter certaines contraintes, comme celle-ci.
        * Root étant indirectement une feature, il pourrait être choisi
        * pour être un nouvel enfant. Ce n'est pas ce que je souhaite.*/
        invariant childrenCantBeRoot:
            parentHasChildren->selectByKind(Root)
            ->size() = 0
        ;
    }
    abstract class Feature
    {
        [...]
```

```
}
class FeatureNode extends Child,Parent;
class FeatureGroup extends Child,Parent
{
   /* Une instance de FeatureGroup doit regrouper des features. Il
   * faut donc qu'il ait au moins deux enfants. */
   invariant aFeatureGroupHasAtLeast2Children:
      parentHasChildren->size()>=2
   ;
}
class OptionGroup extends Child,Parent
{
   /* Une instance d'OptionGroup doit regrouper des features. Il
   * faut donc qu'il ait au moins deux enfants. */
   invariant anOptionGroupHasAtLeast2Children:
      parentHasChildren->size()>=2
   ;
   /* Une instance d'OptionGroup, si elle est sélectionnée, doit
   * avoir au moins un de ses enfants sélectionné.*/
   invariant atLeastOneOptionIsSelected:
      isSelected implies
      parentHasChildren->select(isSelected = true )->size() >=1
   ;
   /* L'enfant d'une instance d'OptionGroup doit avoir un type de
   * variabilité définit à eOr.*/
   invariant childrenVariabilityTypeMustBeOr:
      parentHasChildren->select(cType = variabilityType::eOr)
      ->size() = parentHasChildren->size()
   ;
}
class AlternativeGroup extends Child,Parent
{
   /* Une instance d'AlternativeGroup doit regrouper des features. Il
   * faut donc qu'il ait au moins deux enfants. */
   invariant anAlternativeGroupHasAtLeast2Children:
      parentHasChildren->size()>=2
   ;
   /* Une instance AlternativeGroup, si elle est sélectionnée, doit
   * avoir au moins un de ses enfants sélectionné,
   * et au maximum un.*/
   invariant oneButOnlyOneSelectedAlternative:
      isSelected implies
      parentHasChildren->select(isSelected = true )->size() =1
   ;
   /* L'enfant d'une instance d'AlternativeGroup doit avoir un type
   * de variabilité définit à eOr.*/
   invariant childrenVariabilityTypeMustBeXOr:
      parentHasChildren->select(cType = variabilityType::eXor)
      ->size() = parentHasChildren->size()
   ;
}
class Requirement
{
   [...]
}
enum requirementType { serializable }
{
   [...]
}
enum constraintType { serializable }
```

```
{
    [...]
}
class Constraint
{
    /* Une contrainte OCL concernant l'attribut dérivé cName. Le nom
    * doit être défini à partir d'autres attributs:
    * from cConstraintType to */
    attribute cName : String { derived id }
    {
        derivation:
            if constraintFrom->size()=1 and constraintToChild->size()=1
            then (if cConstraintType= constraintType::requires
                    then
                    self.from.concat('_').concat('requires').concat('_').
                    concat(self.to)
                    else
                    self.from.concat('_').concat('excludes').concat('_').
                    concat(self.to)
                    endif
                )
            else ''
            endif;
    }
    /* Une contrainte OCL concernant l'attribut dérivé from. Si
    * l'interrogation de la relation constraintFrom retourne un
    * résultat (il y a donc une feature qui est liée à cette
    * instance de Constraint) alors from = le nom de ce résultat.
    */
    attribute from : String { derived }
    {
        derivation:
            if constraintFrom->size()=1
            then constraintFrom.cName
            else ''
            endif;
    }
    /* Une contrainte OCL concernant l'attribut dérivé to. Si
    * l'interrogation de la
    * relation constraintToChild retourne un résultat, alors
    * to = le nom de ce résultat (de type Feature).
    */
    attribute to : String { derived }
    {
        derivation:
            if constraintToChild->size()=1
            then constraintToChild.cName
            else ''
            endif
        ;
    }
    [...]
    /* Si la feature à l'origine de la contrainte est sélectionnée,
    * et que la contrainte est de type excludes, alors la feature
    * sur laquelle s'exerce la contrainte ne doit pas être
    * sélectionnée. */
    invariant excludedFeatureMustNotBeSelected:
        constraintFrom.isSelected
        and cConstraintType = constraintType::excludes
        implies constraintToChild.isSelected = false
```

```
        ;
        /* De la même manière, mais pour la contrainte requires.
        * La feature cible doit être sélectionnée. */
        invariant requiredFeatureMustBeSelected:
            constraintFrom.isSelected
            and cConstraintType = constraintType::requires
            implies constraintToChild.isSelected = true
        ;
        /* Une Feature ne peut s'exclure ou exiger sa propre présence.*/
        invariant aFeatureCantConstraintItSelf:
            (constraintFrom = constraintToChild) = false
        ;
    }
    enum variabilityType { serializable }
    {
        [...]
    }
    enum abstractionLevel { serializable }
    {
        [...]
    }
    class ProductLine
    {
        [...]
    }
}
```

A ce stade, et tout au long de l'élaboration des contraintes, il est possible de les tester en créant une instance dynamique de notre modèle que l'on peut peupler et valider en même temps que l'on élabore les contraintes OCL.

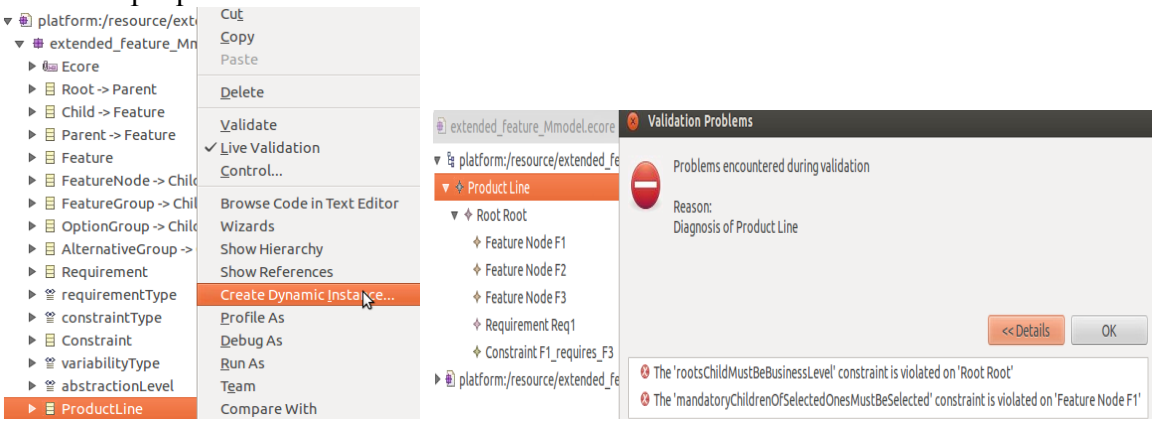


Figure 35 – Validation des contraintes OCL – 1/2.

Dans cet exemple, au moins un des enfants de root possède un attribut cAbstractLevel différent de « business », ce qui viole la contrainte rootsChildMustBeBusinessLevel.

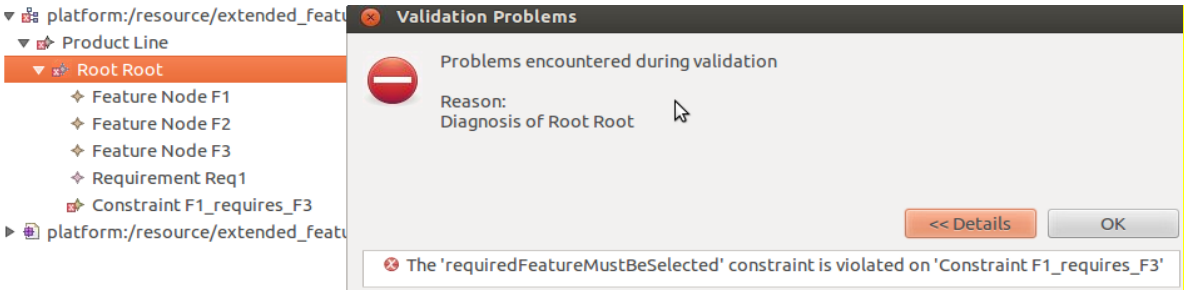


Figure 36 – Validation des contraintes OCL – 2/2.

Dans cet exemple, F1 requière F3. F1 est sélectionné (attribut isSelected = true), mais F3 ne l'est pas. La contrainte requiredFeatureMustBeSelected est violée. Une console OCL peut aussi être ouverte pour tester des expressions OCL et obtenir un résultat.

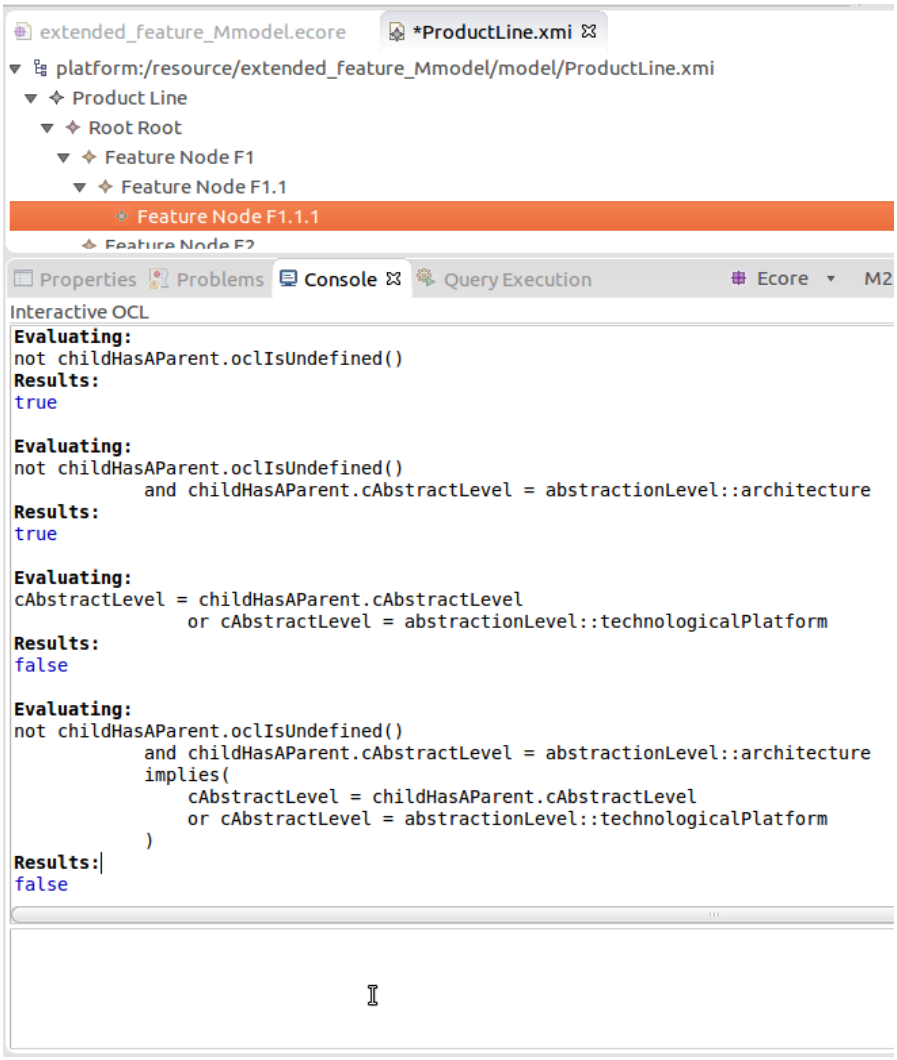


Figure 37 – Console OCL.

En sélectionnant F1.1.1, on se place dans son contexte. L'expression saisie sera évaluée depuis F1.1.1. Dans cet exemple, on voit l'élaboration étape par étape d'un futur invariant. Je teste différentes parties de ce futur invariant avant de les combiner. Ici F1.1.1 est d'abstractLevel = business, alors que son parent est au niveau architecture. Le résultat est donc false. La contrainte saisie est violée, c'est ce que l'on attendait. Une fois cet invariant adjoint au modèle, le système de validation remontera une erreur à chaque fois que cet invariant n'est pas vérifié :

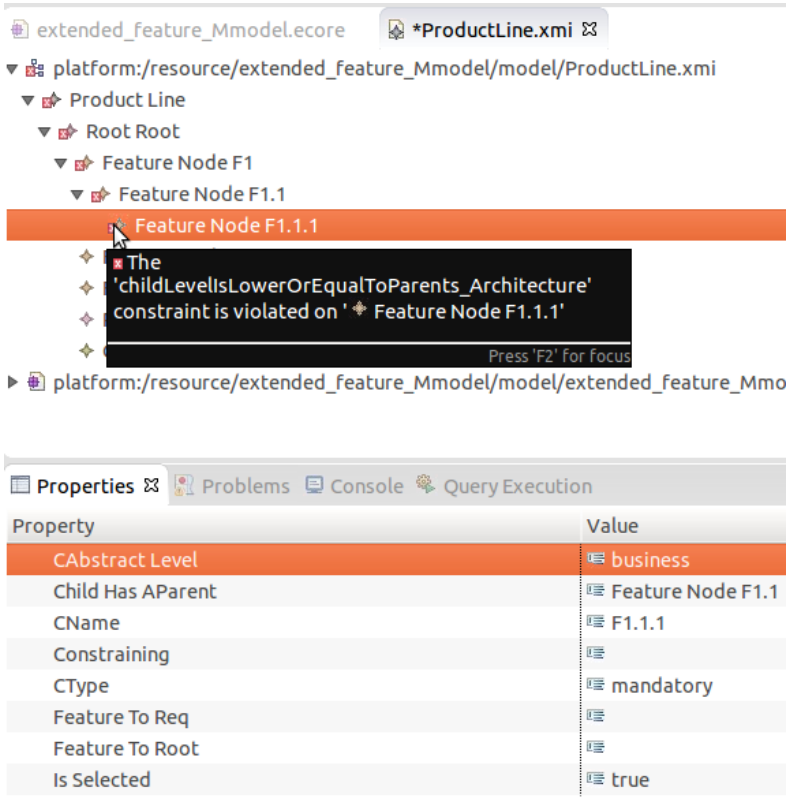


Figure 38 – Système de validation temps réel.

On remarque qu'une croix rouge est placée dans l'explorateur de l'instance (l'arborescence à gauche) au niveau de l'instance de la classe dans laquelle est définie la contrainte et sur ses parents.

IV.2.2 Création du modeleur

On commence donc par générer des plugins eclipse depuis notre métamodèle. L'outil EcoreTools le fait automatiquement à la demande. Une fois les plugins générés, on lance une instance d'eclipse qui embarque les plug-ins générés plus tôt. Ceux-ci nous permettent de créer un « exemple », une instance de notre modèle (que nous utiliserons comme base pour la spécification de notre éditeur dans l'outil Sirius).

Ensuite on créé une instance d'exemple de notre métamodèle dans la nouvelle instance Eclipse. Elle devra contenir au moins un FeatureGroup, OptionGroup, FeatureNode, AlternativeGroup, et ce à chaque niveau d'abstraction. Puis, deux Requirement (un par type de requirement, fonctionnel et non fonctionnel), deux Constraint (un requires et un excludes) pour pouvoir spécifier la représentation de tous les concepts et de toutes les relations entre eux.

Une fois le travail précédent fini, nous pourrions nous contenter de cet outil pour la définition de la ligne de produit et la configuration d'un produit grâce à l'outil de validation. Je peux maintenant instancier mon métamodèle en un feature model étendu, conforme à mes spécifications, et qui exprime la variabilité de ma SPL. L'outil EcoreTools permet de le faire et de valider la conformité d'une instanciation à son modèle. Toutefois l'aide serait minimale, l'élaboration de la ligne de produit et la configuration de produits fastidieuse, bien que facilité par les contraintes OCL et le système de validation qui nous informe des incohérences que l'on pourrait laisser dans le modèle en cours d'élaboration. En effet l'interface d'EMF n'est pas très conviviale et ne permet pas de travailler rapidement. Il faut aller plus loin pour automatiser ce qui peut l'être dans la propagation des contraintes lorsque l'utilisateur élaborera sa ligne de produit et fera des choix lors de la configuration d'un produit. Par exemple :

- Lors de l'ajout d'une feature (au sens large), elle sera enfant de Root, et elle pourra directement être de niveau business, sélectionnée si définie à mandatory.

- Lors de la sélection d'un parent, si ses enfants sont mandatory, ils pourront automatiquement être sélectionnés.

- Lors de la sélection d'une feature qui en exclut une autre, si elle est sélectionnée, l'autre sera désélectionnée.

- Lors de la création d'un lien parent/enfant entre deux features, la feature enfant, si elle a un niveau d'abstraction supérieure prendra un niveau égal à celui de son nouveau parent.

-etc.

C'est pour ceci que j'ai continué et cherché à créer un éditeur graphique reposant sur le métamodèle que j'ai spécifié. Je me suis servi de l'outil Eclipse Sirius, qui permet de spécifier des représentations pour les concepts mis en place dans un modèle compatible emf (Eclipse Modelling Framework). Il me permet donc de créer un modeleur qui sera composé de trois parties :

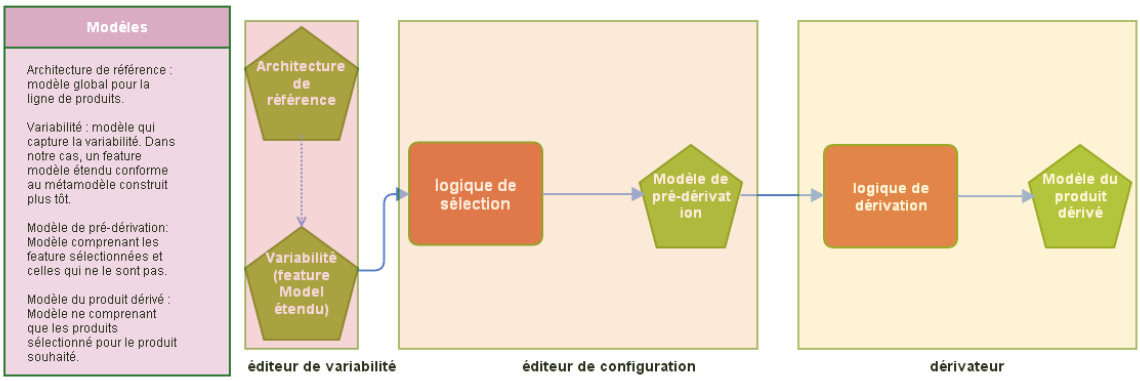


Figure 39 – Composantes du modeleur.

- un éditeur de variabilité, qui est utilisé pour transposer notre architecture de référence en feature model étendu, le modèle de variabilité.

- un éditeur de configuration. J'ai établi une logique de sélection au travers de contraintes OCL que je vais implémenter dans notre modeleur. Un outil permettra de choisir une feature et la logique de sélection répercutera les choix faits automatiquement sur les features impactées (prise en comptes des contraintes « requires », « excludes » etc.). On obtient ainsi le modèle de pré-dérivation qui indique l'ensemble des features, qu'elles soient sélectionnées ou non. Cette caractéristique est indiquée graphiquement.

- un dérivateur : fournit le modèle du produit dérivé en supprimant les features non sélectionnées du modèle de pré-dérivation.

IV.2.2.1 **Spécification de l'éditeur de variabilité avec Sirius**

IV.2.2.1.1 **Instanciation du modèle d'exemple**

Sirius demande une instance de notre métamodèle sur laquelle s'appuyer pour y trouver les concepts à représenter. Il permet ensuite de définir la manière de représenter chacun des concepts indiqués dans le modèle et chacune des relations. Le résultat peut être observé en temps réel, ce qui permet de gagner beaucoup de temps.

On commence par instancier notre métamodèle et le peupler. (L'objectif ici est simplement de directement pouvoir observer le rendu d'un concept dès qu'il a été spécifié dans Sirius.)

EMF fournit des générateurs de code. Ils permettent de générer les classes Java de notre modèle Ecore. Cette génération est paramétrée par un modèle EMF ayant une extension .genmodel. (11) L'éditeur du .genmodel permet aussi de générer le code Java de notre modèle adapté pour permettre l'affichage des classes (texte, icônes et descriptions par défaut sont fournies), notamment au sein d'un éditeur EMF simple intégré à Eclipse. On peut ainsi instancier notre métamodèle et l'éditer au sein d'une nouvelle instance d'Eclipse. Par défaut, sont notamment générées :

- les interfaces correspondants aux concepts de notre métamodèle (dans mon cas, dans le package extended_feature_Mmodel.src.extended_feature_Mmodel).

- Les classes d'implémentation de ces interfaces (qui sont présentes dans le package extended_feature_Mmodel.src.extended_feature_Mmodel.impl).

- Des classes utilitaires dans le dernier package (dans mon cas, dans le package extended_feature_Mmodel.src.extended_feature_Mmodel.util).

De nombreux mécanismes sont implémentés, notamment des méthodes de manipulation des classes de manière réflective : les méthodes eGet, eSet et eContainer() seront utilisées dans le cadre de l'outil Sirius.

La première étape consiste à créer les plug-ins de l'éditeur de modèle. Ils fournissent les outils pour créer de nouvelles instances de notre modèle ainsi qu'un éditeur pour saisir les informations de l'instance sur laquelle on travaille. EcoreTools crée par défaut un fichier .genmodel pour notre modèle dans le modelling project qui contient notre spécification de métamodèle (d'autres outils ne le font pas, il faut alors le créer manuellement). En ouvrant ce fichier, on va pouvoir générer les plugins edit et editor par un simple Clic droit. Ici je choisis « generate all » pour aller au plus vite.

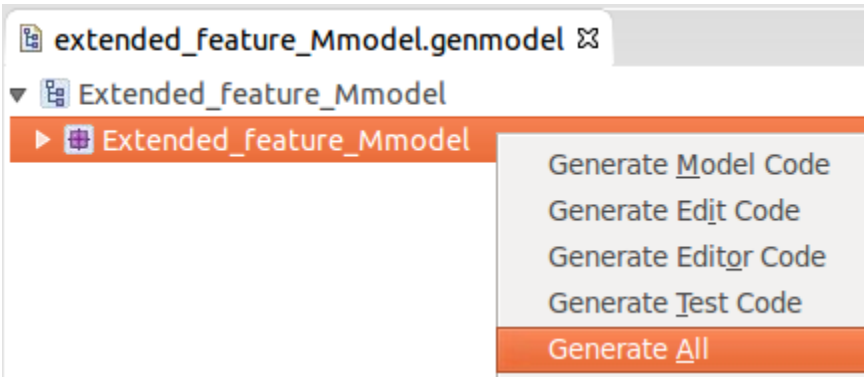


Figure 40 – Instanciation de modèle 1/6.

Les deux plug-ins Eclipse qui nous intéressent se retrouvent dans l'explorateur.

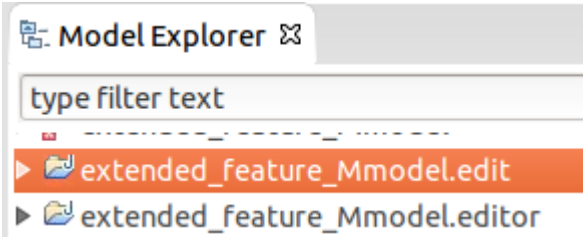


Figure 41– Instanciation de modèle 2/6.

Il faut ensuite lancer les plug-ins en sélectionnant le plug-in *.editor et en démarrant une nouvelle instance Eclipse. Cela se fait par un simple clic-droit.

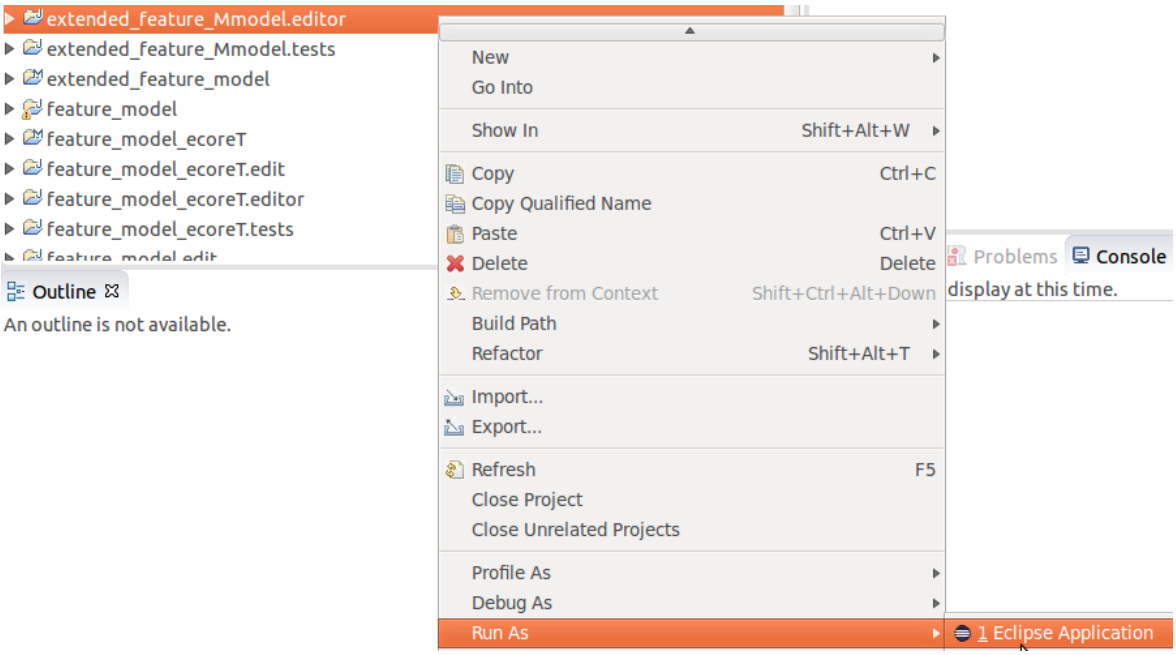


Figure 42 – Instanciation de modèle 3/6.

Nous ne travaillerons plus que dans la nouvelle instance d'Eclipse. Après avoir sélectionné la vue Sirius, on peut créer un « modelling project » qui contiendra l'instance exemple de notre métamodèle : extended_feature_Mmodel.example.model. L'instance se crée en un clic droit sur le modelling project > new > other> Exemple EMF Model Creation Wizards. Il suffit de choisir le métamodèle qu'on a spécifié avec EcoreTools :

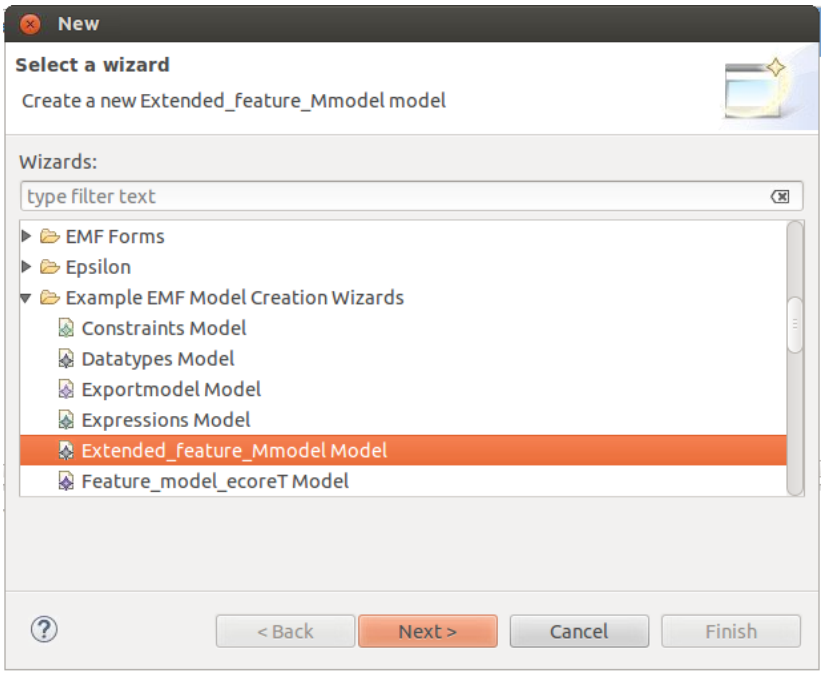


Figure 43 – Instanciation de modèle 4/6.

Puis on choisit un nom pour l'instance de featuremodel sans changer l'extension (elle correspond au nom du modèle spécifié avec ecoretools en minuscule).

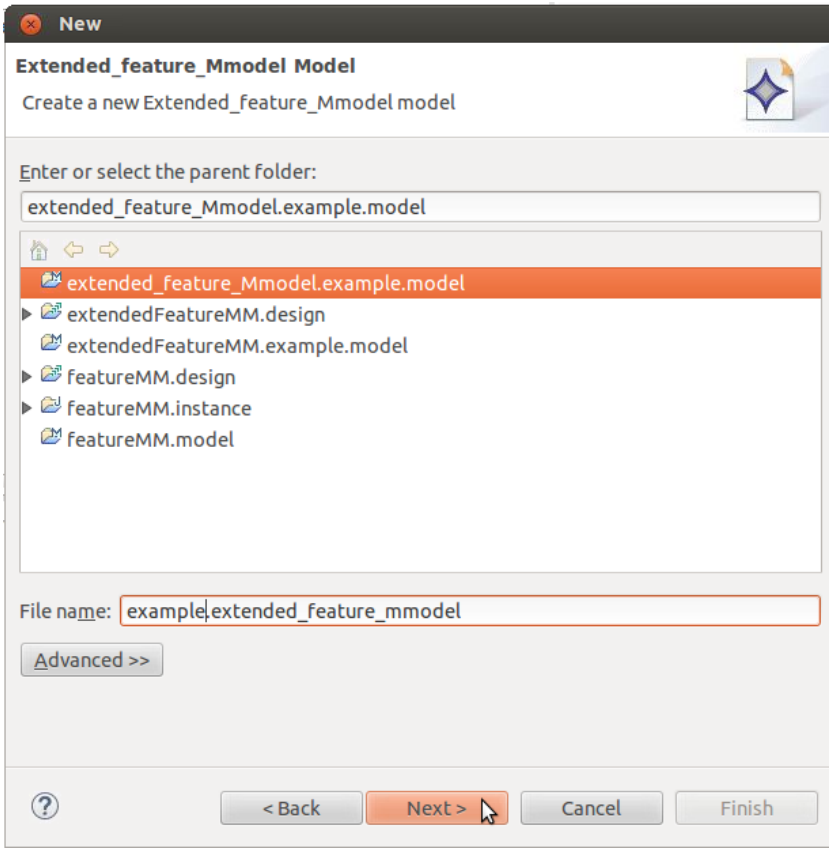


Figure 44 – Instanciation de modèle 5/6.

Il faut ensuite choisir l'objet racine (Model Object) du modèle : pour notre cas c'est ProductLine. Nous avons notre instance de modèle qu'il faut peupler.

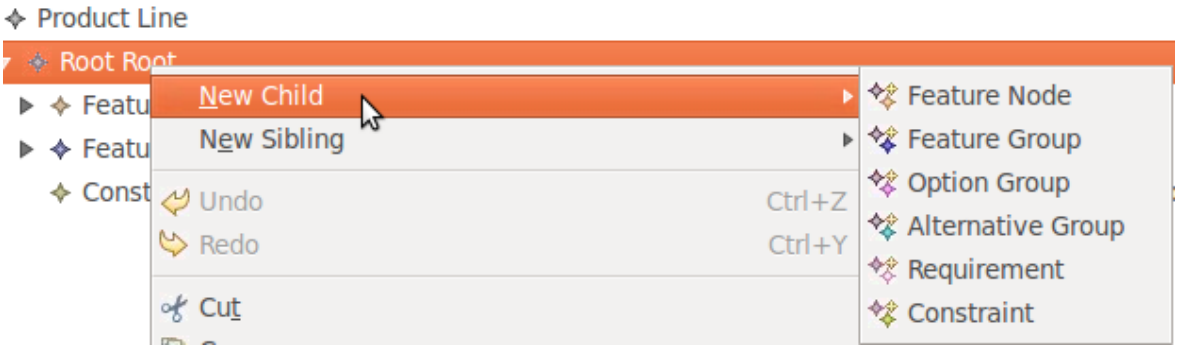


Figure 45 - – Instanciation de modèle 6/6.

Il faut se rappeler que l'objectif est d'utiliser cette instance pour pouvoir spécifier la représentation de chacun des concepts de notre métamodèle dans l'outil Sirius. Peu importe donc le détail des noms et des attributs, ni que ce modèle ne représente pas une ligne de produit aboutie. Il suffit que cette instanciation contienne l'ensemble des concepts et des relations possibles indiquées dans notre métamodèle. Elle devra contenir au moins un FeatureGroup, OptionGroup, FeatureNode, AlternativeGroup, et ce à chaque niveau d'abstraction. Puis, deux Requirements (un par type de requirement, fonctionnel et non fonctionnel), deux Constraints (un requires et un excludes) pour pouvoir spécifier la représentation de tous les concepts et de toutes les relations entre eux.

IV.2.2.1.2 **Spécification de la représentation sous Eclipse Sirius**

Maintenant que notre instance d'exemple est prête, il nous faut définir la représentation de ses concepts.

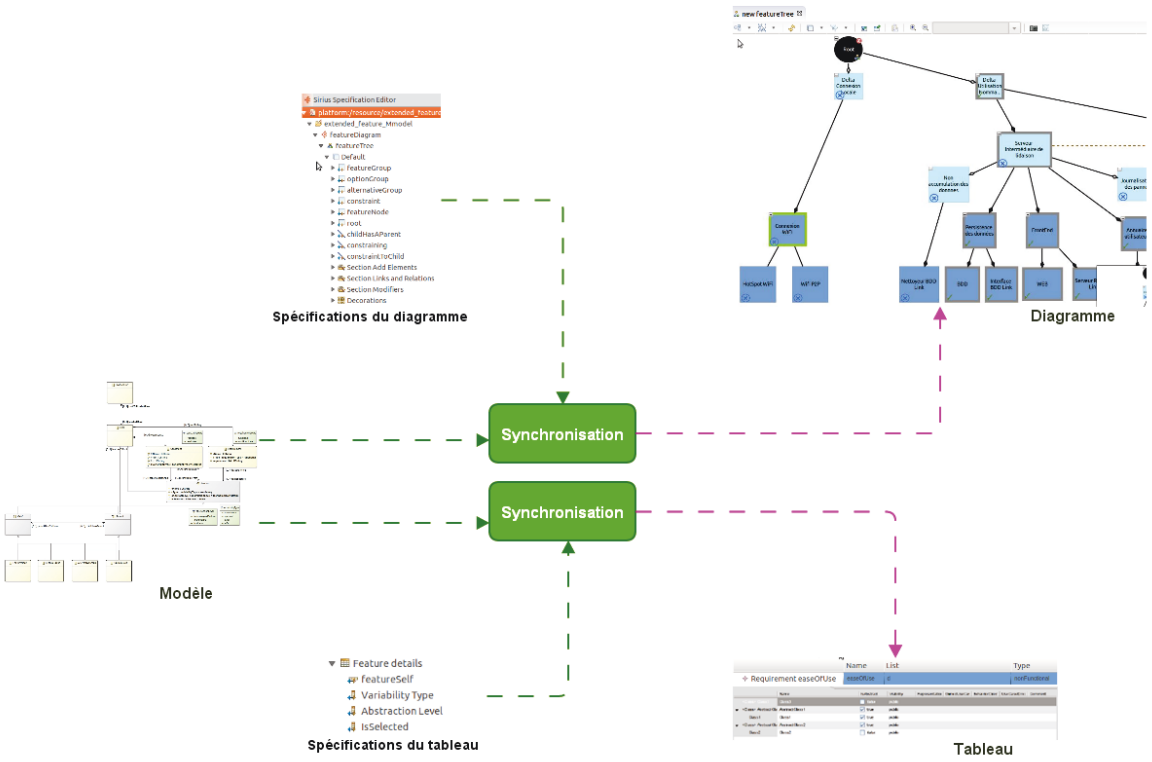


Figure 46 – Représentations Sirius.

Ceci se fait à l'aide de l'outil Sirius. On peut spécifier la représentation des concepts de notre modèle pour obtenir un diagramme ou un tableau et observer le résultat. Le diagramme ou le tableau est synchronisé avec la spécification. Cette spécification est stockée dans template de paramétrisation, un fichier *.odesign, qui nous permettra de construire un modeler personnalisé. Dès que les modifications apportées à la spécification sont enregistrées, le résultat s'affiche sur le diagramme ou le tableau (d'autres représentation sont possibles, toutefois je n'ai spécifié que des diagrammes ou des tableaux pour mes besoins dans le cadre de ce mémoire). Sirius repose sur le Graphical Modeling Framework d'Eclipse (GMF), notamment sur l'Eclipse Modeling Framework (EMF) et le Graphical Editing Framework (GEF), qui fournissent tous les éléments nécessaires à la construction d'un modeleur. L'avantage de Sirius est de simplifier la création d'un modeleur comparativement à ce que permettent GEF et GMF. Toutefois, l'utilisation de Sirius exige que certains concepts soient compris :

- Sirius permet de manipuler les éléments du modèle d'exemple qu'on lui fournit ainsi que les éléments de la représentation qu'il produit. On peut donc se placer selon l'un de ces deux contextes. Le vocabulaire Sirius parle d'éléments sémantiques lorsque l'on fait référence à

un élément du modèle et d'élément graphique lorsque l'on fait référence à un élément du diagramme ou du tableau que l'on spécifie.

- Les viewpoints : la notion de viewpoint est une abstraction qui permet de fournir les spécifications d'un système en se concentrant sur une problématique particulière. Cela permet une séparation des problèmes ou des rôles dans la spécification de la représentation d'un système et permet de guider les utilisateurs du futur modeleur en cours d'élaboration dans la modélisation du type de système pour lequel le modeleur est spécifié. Par exemple, dans notre cas, nous pourrions séparer la spécification d'une ligne de produit en laissant l'architecte utiliser le futur modeleur pour ajouter des composants en fonction des fonctionnalités attendues, et laisser le soin à l'analyste des besoins de spécifier les besoins à l'origine des composants présents dans le diagramme. L'objectif est d'effectuer une séparation des préoccupations et de permettre une plus grande expressivité : un viewpoint fournira un ensemble de représentations dédiées à une préoccupation.

- Les représentations : au sein d'un viewpoint, une représentation est utilisée pour voir ou éditer un ensemble de concepts sémantiques. Une représentation se présente sous la forme d'un diagramme, d'un tableau ou d'une matrice.

- Les layers : au sein d'une représentation, différentes « couches » peuvent exister. Ces couches regroupent l'ensemble des éléments graphiques que l'on souhaite. Ces couches peuvent se superposer. Cela permet de laisser le choix à l'utilisateur d'afficher ou non certains éléments de notre modèle pour plus clarté et de le laisser se concentrer sur certaines représentations seulement.

- Les éléments graphiques : on peut associer une représentation que l'on spécifie à un concept de notre modèle.

- Les requêtes : afin de spécifier les éléments graphiques ou leur comportement, le besoin de naviguer au travers du modèle d'exemple ou de sa représentation (diagramme, tableau…) est important. Sirius permet de le faire en utilisant le langage OCL ou Acceleo.

Nous allons spécifier pour chaque élément de notre modèle d'exemple une représentation sous Sirius. Il faut pour cela créer « viewpoint Spécification Project » puis donner un nom au fichier .odesign qui contiendra la spécification du modeleur. Le fichier s'ouvre en affichant un unique élément. Il faut créer un « viewpoint ». Je choisis de créer un viewpoint pour la définition du feature model.

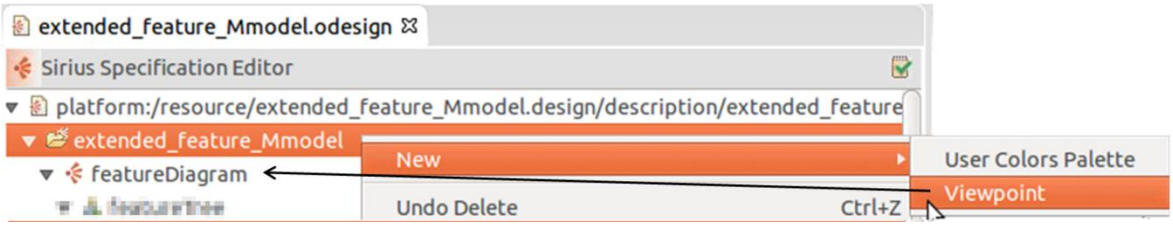


Figure 47 – Création d'un ViewPoint.

Une fois créé, il faut renseigner certaines propriétés. La plus importante est le champ « Model File Extension » où il faut renseigner l'extension du fichier de modèle qu'on souhaite associer à notre viewpoint (extended_feature_mmodel dans notre cas).

Il faut ensuite créer une représentation, pour notre cas, un diagramme qui représentera nos features et leurs relations.

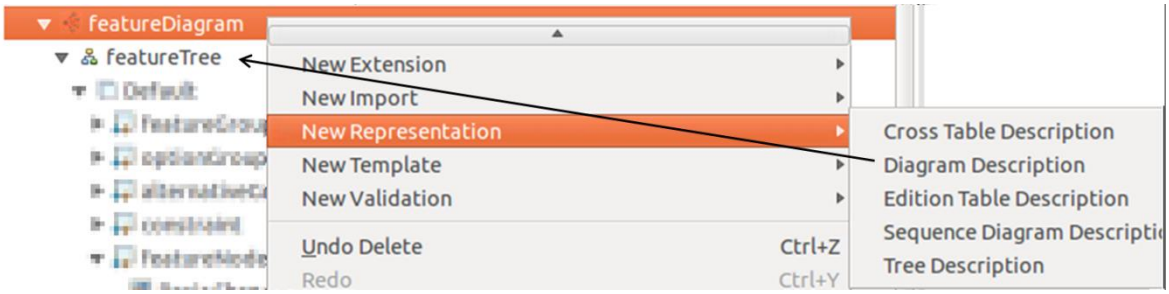


Figure 48 – Création d'une représentation.

Il faut renseigner certaines propriétés. La plus importante est celle du champ « Domain Class ». Il faut indiquer le type (la classe) de l'élément racine de notre diagramme. Pour nous il s'agit de ProductLine qui dans notre métamodèle introduit les autres éléments.

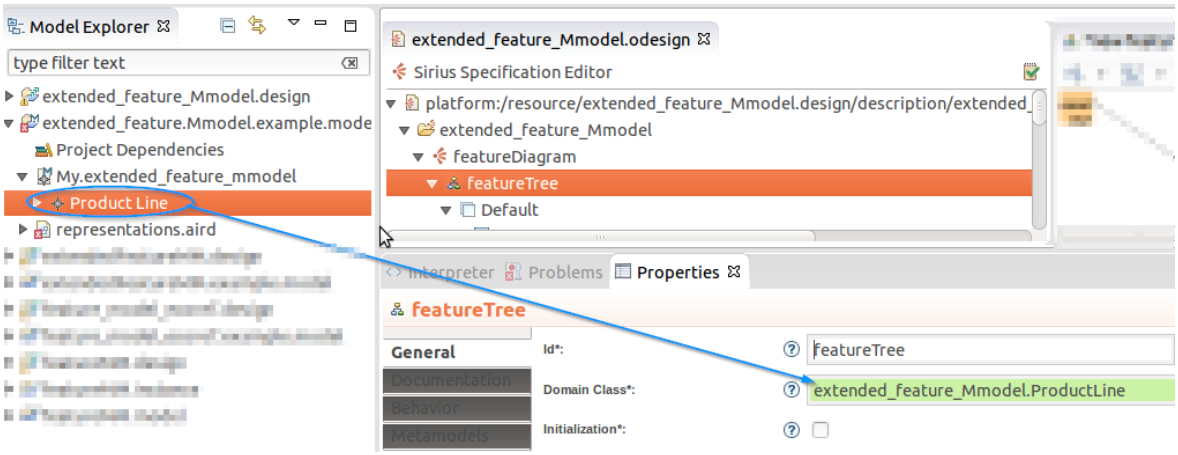


Figure 49 – Domain Class de la représentation.

Dans l'onglet Metamodels, il est possible d'ajouter le chemin du fichier ecore contenant la description de notre métamodèle. Ceci est optionnel mais permet à Sirius d'être plus précis dans l'évaluation des requêtes que s'il se reposait sur la seule instance d'exemple de notre métamodèle.

On peut maintenant visualiser le modeleur en temps réel et observer les répercussions de toutes les modifications apportées au fichier odesign par un clic droit sur le projet qui contient le modèle d'exemple et la sélection de Viewpoints Selection > le viewpoint que l'on vient de créer. Ensuite, dans ce même projet, si on sélectionne l'élément racine de notre modèle d'exemple (comme indiqué plus haut, ProductLine) on peut créer une nouvelle instance de notre représentation :

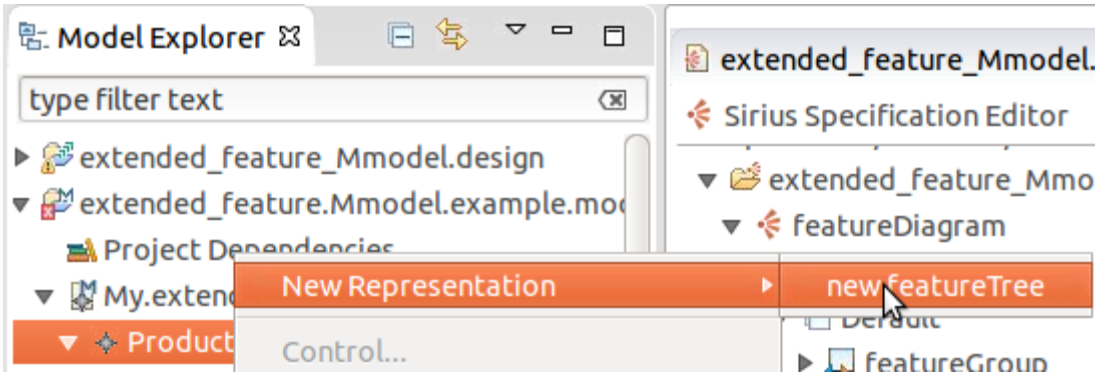


Figure 50 – Création d'une instance de représentation.

On peut maintenant travailler sur la spécification tout en observant le résultat en organisant les fenêtres Eclipse ainsi :

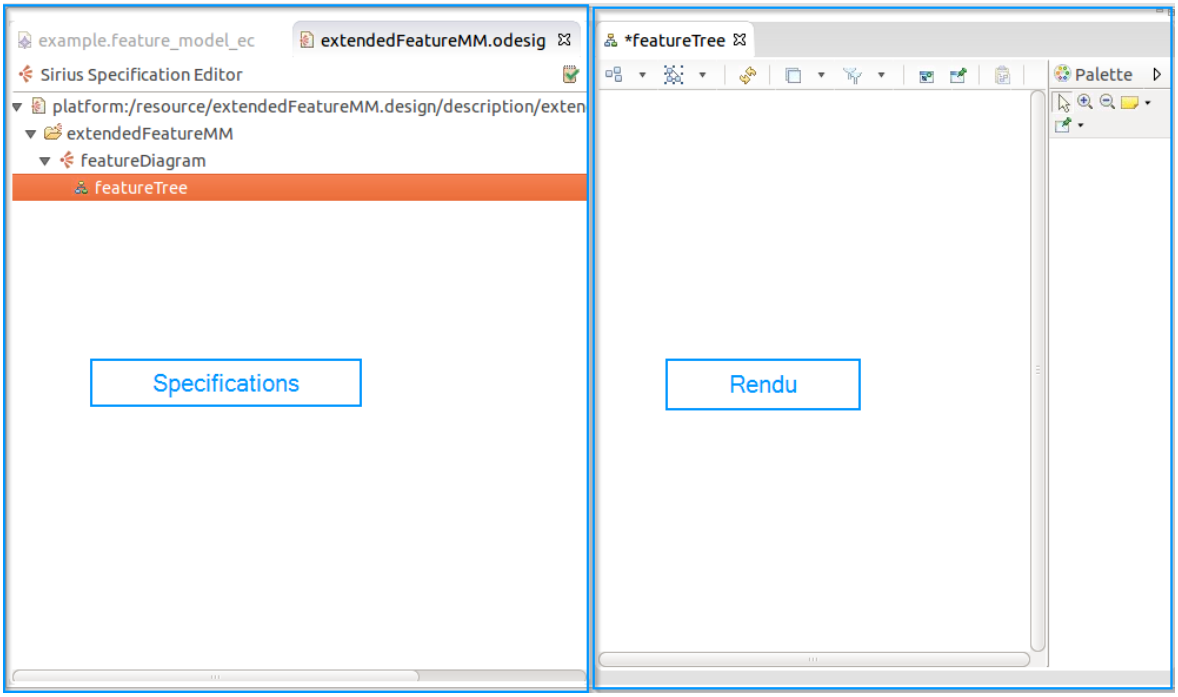


Figure 51 – Rendu en temps réel.

Je crée un layer, une couche par défaut qui affichera nos features. Un autre sera créé pour afficher les requirements à la demande.

Figure 52 – Layers.

Dans ce layer je définis la représentation des différents types de features par des éléments graphiques (ou diagram Element) et les liens entre elles :

Figure 53 – Eléments graphiques.

Je me suis servi de deux types d'éléments graphiques :

- Les Nodes : elles permettent d'associer à un élément sémantique une représentation graphique que l'on spécifie. Il faut donc indiquer quelle classe l'élément graphique va représenter. Par exemple, pour FeatureNode :

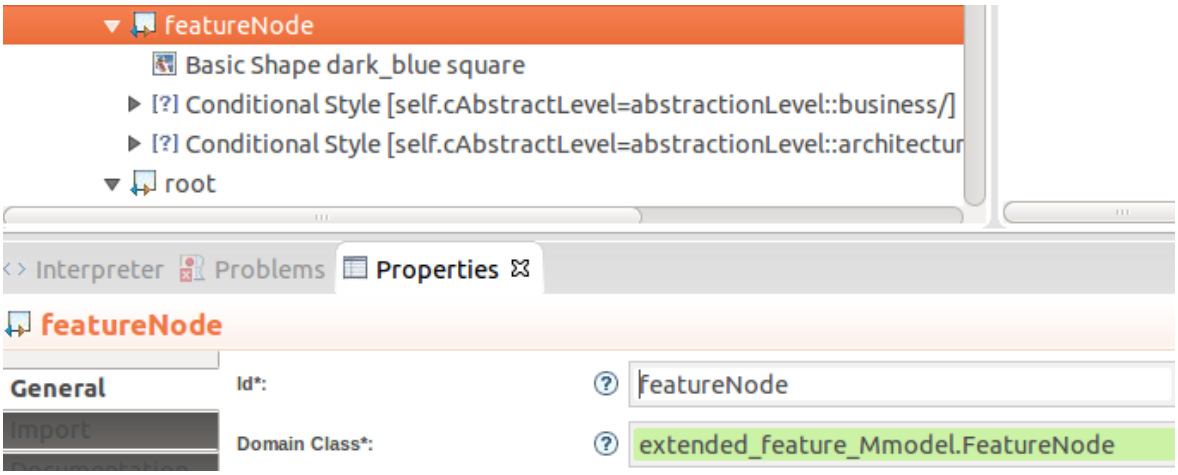


Figure 54 – Association élément graphique – élément sémantique.

Il faut ensuite indiquer son « style », sa représentation. J'ai choisi une représentation par défaut et deux styles conditionnels, en fonction de son niveau d'abstraction. Les styles conditionnels se définissent grâce à une « predicate expression ». Par exemple, [self.cAbstractLevel=abstractionLevel::business/] teste si l'élément du modèle d'exemple en cours de représentation est un featureNode au niveau d'abstraction business. Si c'est le cas, le style associé à cette condition est appliqué : la couleur, la forme et l'étiquette peuvent être définies. L'étiquette peut être obtenue par interrogation du modèle. Je choisis d'y faire apparaître l'attribut cName des FeatureNode. J'ai défini selon la même méthode les autres types de feature. Le style et la couleur de la bordure change pour chacun. La couleur de remplissage indique le niveau d'abstraction. Enfin, des « decorations » peuvent être définies. Ce sont des éléments graphiques additionnels qui viennent se greffer à nos représentations. J'en ai défini deux pour indiquer si la feature est sélectionnée ou non. Il suffit d'indiquer la condition d'application de la décoration (precondition expression), son emplacement, et les éléments graphiques sur lesquelles elle s'applique.

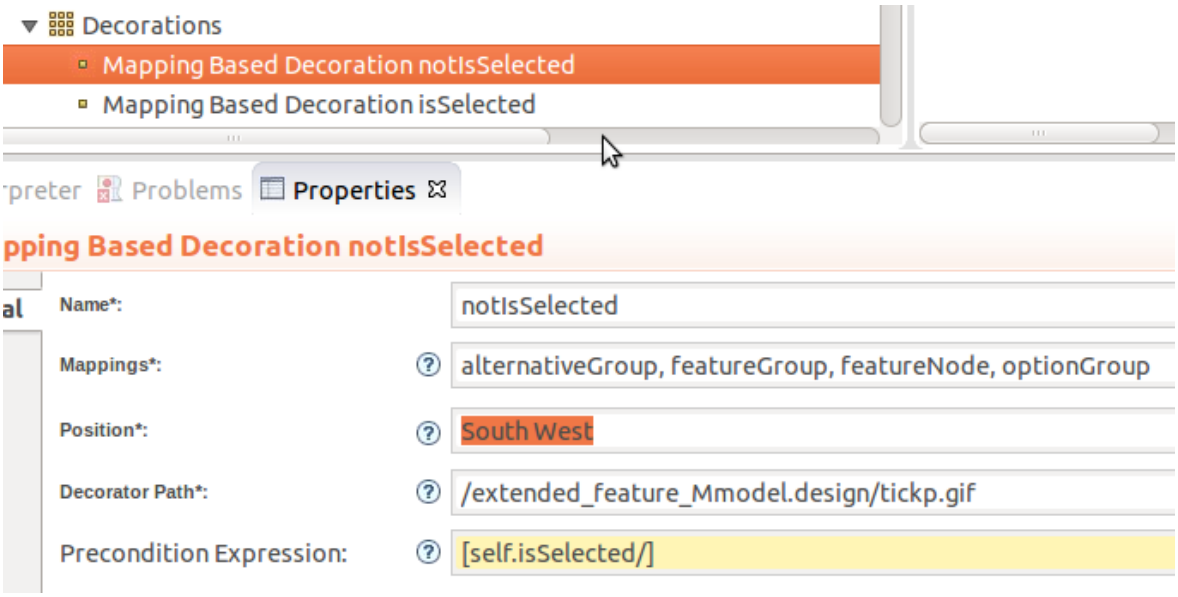


Figure 55 – Spécification d'une décoration.

Dans mon cas j'ai interrogé l'attribut isSelected pour conditionner l'application de l'un ou l'autre decorator. Lorsque isSelected = true, une icône que j'ai créé (un tick) est ajouté, sinon une croix rouge. Par exemple :

Figure 56 – Décorations NotIsSelected et isSelected

- Les Relation based Edges : j'ai représenté les liens entre features en fonction de la relation présente entre elles dans notre modèle. Le lien parent/enfants est la relation childHasAParent. Le lien entre une feature qui en exclu ou requière une autre et un Constraint est la relation constraining. Enfin le lien entre un Constraint et une feature exigée ou exclue est la relation constraintToChild. Par exemple, le lien parent/enfant se spécifie ainsi :

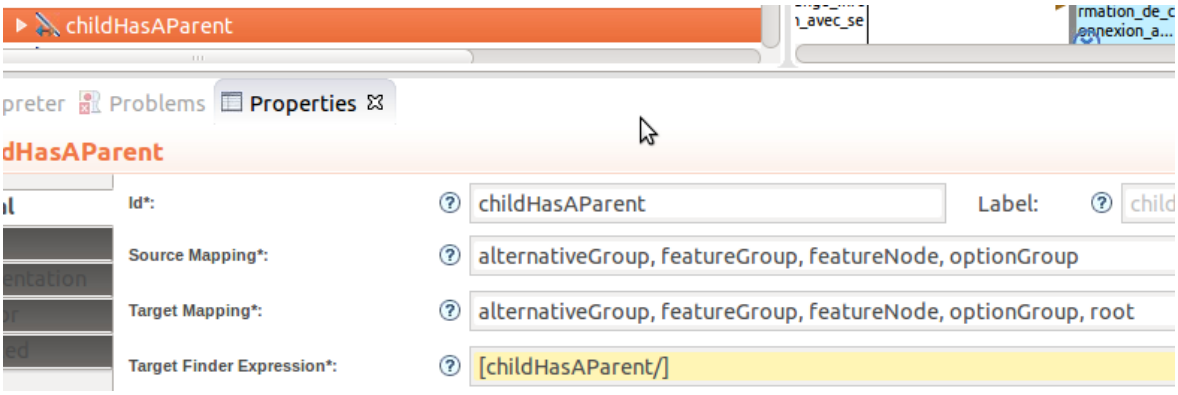


Figure 57 – Spécification de la relation childHasAParent.

On indique la source potentielle du lien et sa destination potentielle dans le contexte graphique (pas sémantique). Enfin on indique le « Target Finder Expression » par son rôle, sa relation présente dans le modèle. Ici je pars d'un enfant vers son parent. C'est ainsi qu'il faudra procéder lorsque l'on voudra ajouter un tel lien entre deux features dans le modeleur une fois que l'outil le permettant aura été spécifié.

En l'état, le modeleur ne permet que l'affichage des éléments déjà présents dans notre modèle d'exemple. Il faut ajouter des outils pour que l'utilisateur puisse :

- créer de nouveaux éléments (features et contraintes) et leurs relations,

- les modifier,

- Les supprimer.

Ces actions seront aussi enregistrées au sein du modèle d'exemple. Ceci se fait par la spécification d'outils. Ils seront ajoutés au modeleur au sein de la palette. La figure suivante montre la relation entre les outils spécifiés et leurs dispositions dans la palette du modeleur.

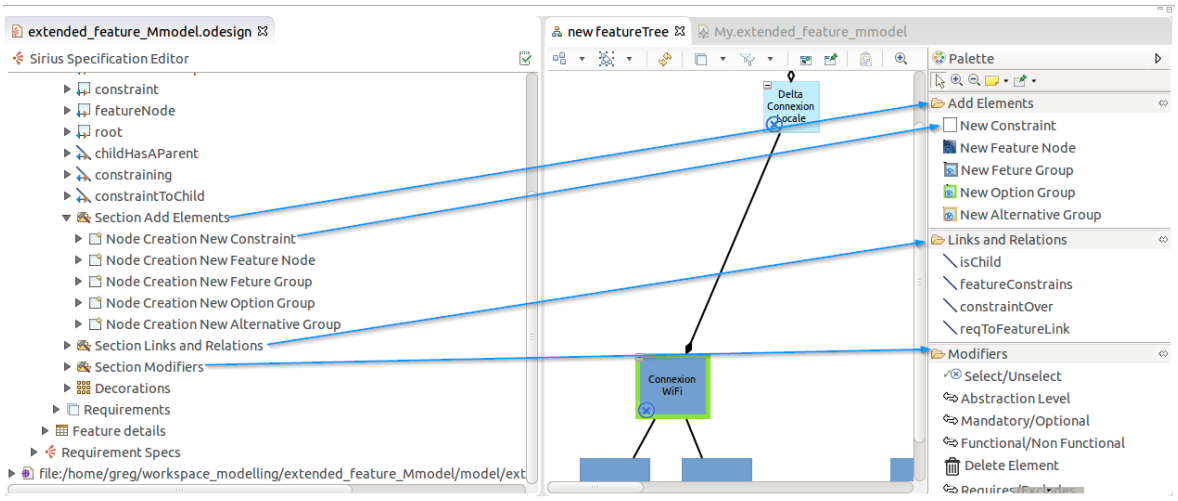


Figure 58 – Outils et palette.

Il faut commencer par créer des sections. Elles permettent d'organiser les outils s'il le faut. Dans notre cas, en ce qui concerne les features, j'ai créé trois sections : Ajout d'éléments, Liens et relations, Modification. La dernière permet de modifier certains attributs (le niveau d'abstraction par exemple) par simple clic sur la feature.

Les outils d'ajout sont tous spécifiés de la même manière :

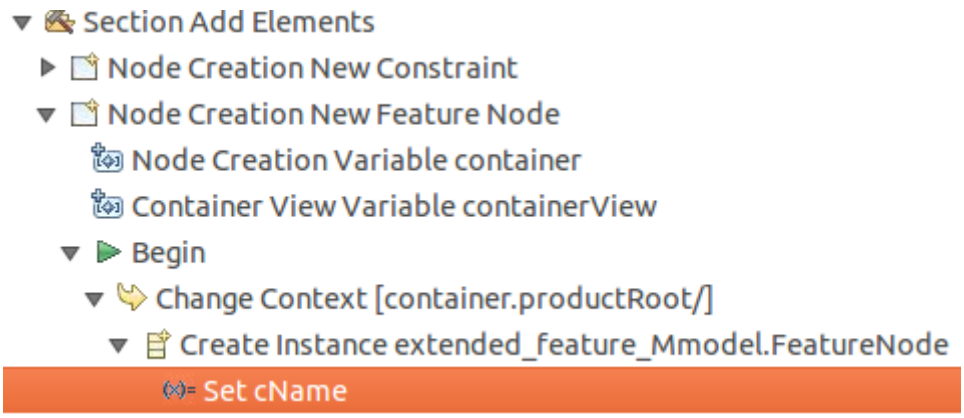


Figure 59 – Exemple de spécification d'outils d'ajout.

Sirius nous fournit par défaut deux variables : container, qui pointe vers le conteneur base des autres classes (le point d'entrée dans le métamodèle dans notre cas) et containerView, son équivalent dans le contexte graphique. Il nous faut spécifier les actions que l'on souhaite voir accomplies par l'outil. Cela commence à la balise Begin. Dans notre cas, je me place dans le contexte du conteneur et je navigue vers Root. Je souhaite qu'il soit le conteneur par défaut de toute nouvelle feature. Ceci se fait par l'opération « Change

Context » et son « Browse Expression » définie à [container.productRoot/] qui me mène à Root. L'utilisateur pourra ensuite en faire l'enfant d'une autre feature grâce à un outil qu'il trouvera dans la palette sous la section Links and Relations. Ensuite une opération « Create Instance » doit être spécifiée, en y indiquant la classe qui sera instanciée par l'outil grâce à la propriété « Type Name ». On y indique aussi la propriété « Reference Name », la relation faisant référence à cette instance nouvellement créée depuis son conteneur. Pour notre cas, le conteneur sera Root. Root fait référence aux features par la relation parentHasChildren d'après notre modèle. Enfin il est possible de définir certains attributs dès la création de l'instance. Ceci se fait par la spécification d'une opération « Set » où il faut indiquer l'attribut à initialiser et la valeur à lui donner.

Concernant les relations, le principale lien est celui établissant la relation Parent/enfant. Il faut spécifier un outil « Edge Creation ».

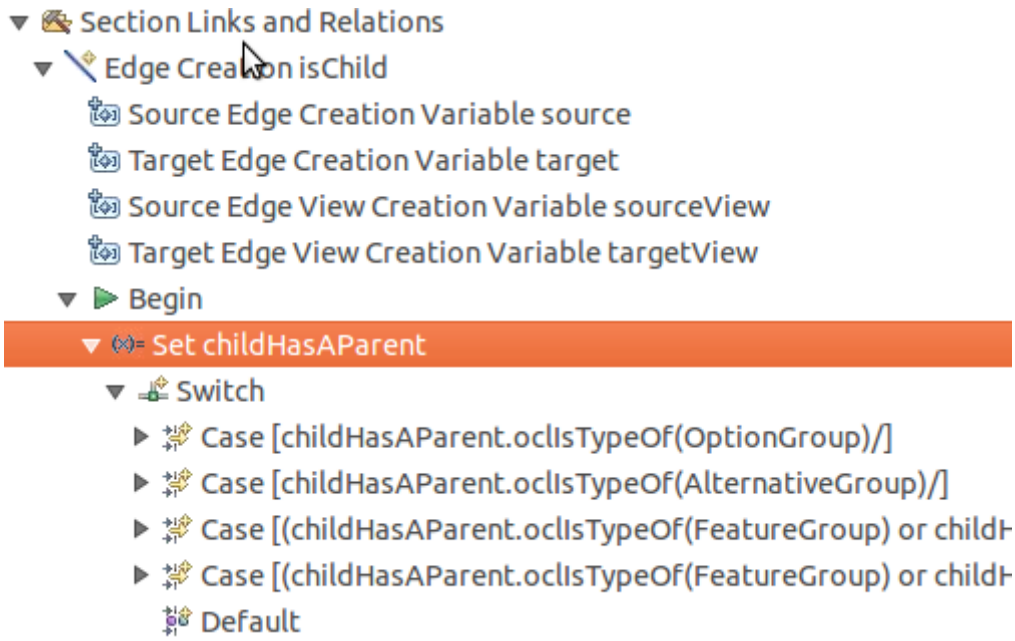


Figure 60 – Exemple d'une spécification d'une relation.

Encore une fois Sirius nous fournit des variables dans le contexte sémantique et graphique. Les variables source et target qui pointent vers la source et la cible du lien que l'utilisateur est en train de « dessiner » dans le modeleur. Ensuite, il faut indiquer les actions que l'outil doit effectuer. Je décide que l'utilisateur devra tracer le lien en partant de l'enfant vers le parent pour établir le lien parent/enfant. Je commence par définir ce vers quoi pointe

childHasParent. Pour chaque feature, childHasAparent renverra la cible, ce vers quoi pointe le lien établi par l'utilisateur grâce à la variable target fournie par Sirius :

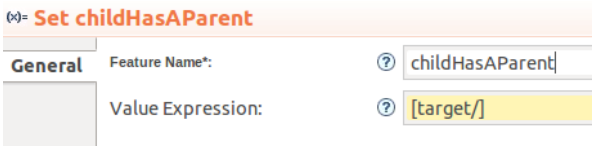


Figure 61 – Définition de la valeur de la variable ChildAsAParent à l'aide de la variable Sirius « target ».

Ensuite je définis d'autres variables selon certaines « Condition Expression » au sein d'une construction Switch. Si le parent est de type «OptionGroup » l'enfant a forcément une variabilité de type eOr. De même pour un parent de type AlternativeGroup, cet attribut sera eXor.

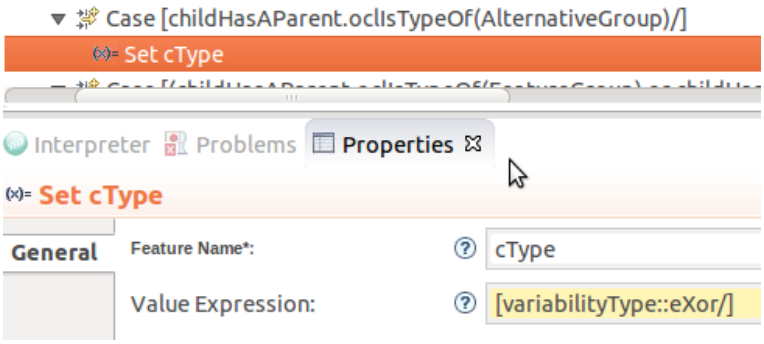


Figure 62 – Exemple de spécification d'une variable conditionnelle 1/2.

Si le parent est de type FeatureNode ou OptionGroup et qu'il est sélectionné je définis l'attribut de l'enfant isSelected à true par défaut. C'est un choix que je fais. A l'inverse si le parent ne l'est pas, l'enfant ne l'est pas non plus. Cette fois-ci, cette action est obligatoire pour garder un modèle cohérent.

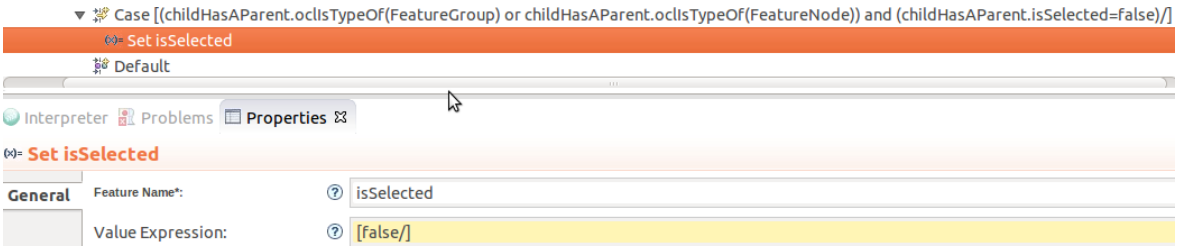


Figure 63 - Exemple de spécification d'une variable conditionnelle 2/2.

Les outils de modification sont aussi en grande partie définis de la même manière. Par exemple l'outil de modification du niveau d'abstraction consiste en la spécification d'un

outil basique, non dédié à une tâche spécifique comme c'était le cas pour la création de nœuds ou d'arcs (selon la terminologie Sirius). Sirius nous fournit toutefois deux variables, chacune dans leur contexte (sémantique et graphique), dont la variable « element » qui pointe vers l'élément du modèle auquel l'outil en cours de spécification se rapporte. La suite d'action pour cet outil est la suivante : changement de contexte, on se place directement sur l'élément du modèle concerné (que l'on cliquera) grâce à la variable « element ». Ensuite, je place une construction Switch et deux « case » : le premier définit l'attribut de la feature sélectionnée à business si celle-ci était technologicalPlatform. Le second, à architecture si elle était à business. Le cas Default qui s'applique si aucune des deux autres ne respecte la condition établie, définit le même attribut à technologicalPlatform. Le résultat est qu'à chaque fois que l'on clique sur une feature à l'aide de cet outil, celle-ci change d'abstractionLevel en itérant dans l'énumération abstractionLevel de manière cyclique.

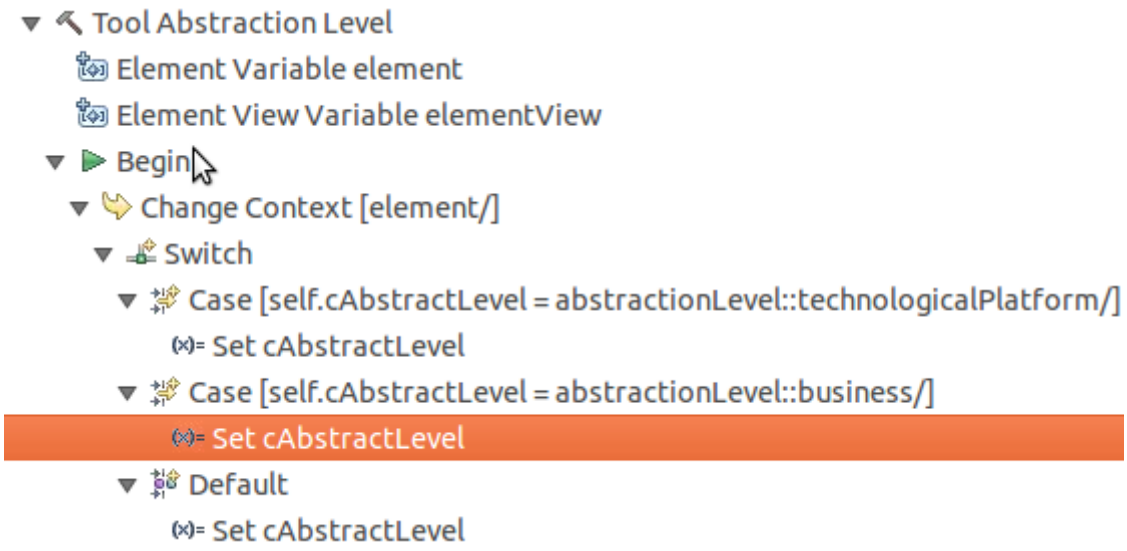


Figure 64 – Exemple de spécification d'outils de modification.

Pour chacun des outils j'ai défini une icône pour plus facilement les identifier.

Concernant les exigences fonctionnelles ou non fonctionnelles (requirements), je souhaite les voir s'afficher sur le diagramme, si l'utilisateur le souhaite. J'ai donc créé une couche additionnelle en définissant la représentation des requirements, sa relation et un outil d'ajout des requirements. Pour les afficher il faut sélectionner ce layer dans le modeleur :

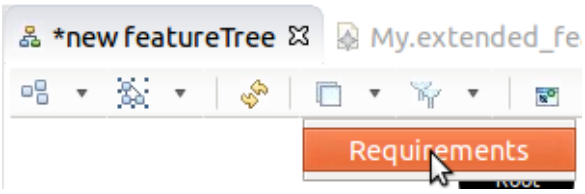


Figure 65 – Sélection de layer depuis le modeleur.

Certaines informations étant plus facilement consultables sous forme de tableau, j'ai aussi spécifié un tableau pour afficher tous les attributs d'une feature. La représentation des requirements dans la vue featureDiagram et son outil d'ajout n'est qu'un plus. Ils ne permettent pas de gérer les requirements aussi bien que le permettrait un tableau. Pour une meilleure gestion des requirements, j'ai créé une vue « Requirement Specs » dédiée à l'analyste des besoins. J'y ai spécifié plusieurs représentations sous forme de tableaux, notamment :

-un tableau dédié à chaque feature permettant de lister les besoins qui la justifient. Une coloration des cellules conditionnelle est spécifiée selon le caractère fonctionnel ou non fonctionnel du besoin qui justifie la feature. Voici un exemple pour la feature IHM-GestionDePartageEtConsultation

	List	Type
GestionDesUtilisateurs	b.2;b.5	functional
GestionDesDroitsSurLesFichiers	b.1;b.3;b.4;b.7	functional
RécupérationDeFichiers	b.6	functional
Authentification	b.4	functional

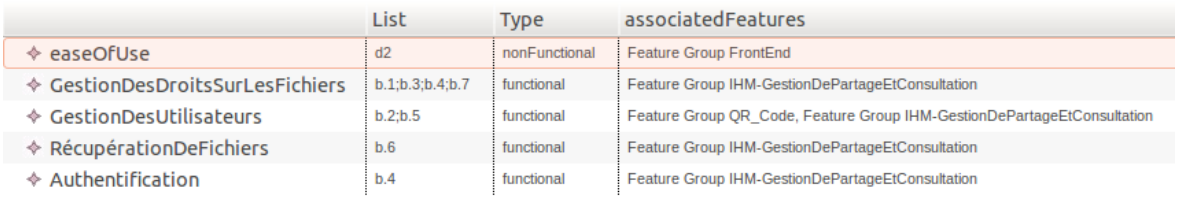
Figure 66 – Exemple de tableau dédié à la relation feature/besoin(s)

- un tableau pour chaque requirement permettant d'afficher ses attributs et listant les features qu'il justifie.

	Name	List	Type	associatedFeatures
Requirement GestionDesUtilisateurs	GestionDesUtilisateurs	b.2;b.5	functional	Feature Group QR_Code, Feature Group IHM-GestionDePartageEtConsultation

Figure 67 – Exemple de tableau requirement/feature associée

- un tableau permettant de lister tous les requirements de la ligne de produit, indiquant leurs attributs et les features qu'ils justifient :



	List	Type	associatedFeatures
easeOfUse	d2	nonFunctional	Feature Group FrontEnd
GestionDesDroitsSurLesFichiers	b.1;b.3;b.4;b.7	functional	Feature Group IHM-GestionDePartageEtConsultation
GestionDesUtilisateurs	b.2;b.5	functional	Feature Group QR_Code, Feature Group IHM-GestionDePartageEtConsultation
RécupérationDeFichiers	b.6	functional	Feature Group IHM-GestionDePartageEtConsultation
Authentification	b.4	functional	Feature Group IHM-GestionDePartageEtConsultation

Figure 68 – Ensemble des requirements et leurs features associées.

Pour obtenir ce tableau il faut spécifier le « domain class », de la même manière que pour notre diagramme spécifié plus haut. Ici j'ai choisi extended_feature_Mmodel.Root.

Il faut ensuite spécifier ce que chaque ligne représentera. Ici ce seront tous les requirements. Selon la même logique que pour la représentation des nodes dans notre diagramme, je renseigne un champ « Domain Class » (extended_feature_Mmodel.Requirement) et « Semantic Candidate Expression » (j'indique [rootToReq/]. Ce champ est optionnel et ce que j'indique là ne réduit pas le nombre de candidats à l'affichage). Ensuite je spécifie trois colonnes. Deux affichant les attributs de chaque requirement (son type et la liste des qualités du modèle de qualité satisfaites par ce requirement) et une dernière qui affiche la liste des features concernées par ce requirement. Il suffit d'indiquer le « Feature Name » de l'instance que représente la ligne. Ici se sont les instances de Requirement, j'indique donc le nom de ses attributs.

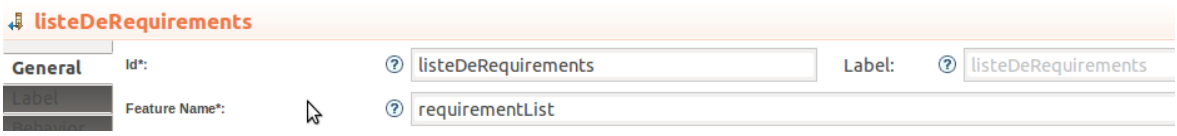


Figure 69- Spécification de la liste de requirements.

Pour le cas de la liste de features, j'indique la relation reqToFeature, que Sirius interrogera pour obtenir la liste de toutes les features vers laquelle cette relation pointe depuis l'instance de Requirement que la ligne courante représente.

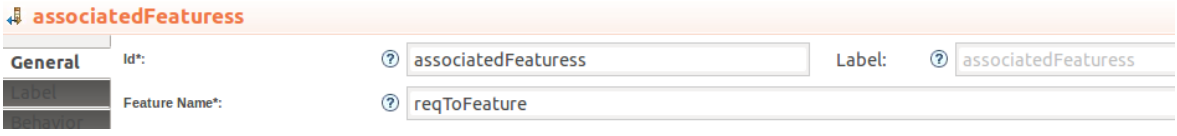


Figure 70 – Spécification de la liste de features.

Le tableau obtenu est éditable : la liste des features associées peut être modifiée, et les attributs du requirement aussi. L'utilisateur pourra ajouter un nouveau requirement grâce à la spécification d'un outil « Create Line tool » qui se spécifie de la même manière qu'un outil de création de nouveau nœud dans notre diagramme (soit l'ajout d'une nouvelle feature).

Nous en avons fini avec la spécification de l'éditeur de variabilité. Pour ne pas alourdir inutilement ce mémoire, je n'ai pas décris de manière exhaustive l'ensemble des paramètres à renseigner, mais les principaux mécanismes sont présentés. Les tableaux peuvent ainsi être créés par l'utilisateur par un clic-droit sur l'instance de Root dans le modèle d'exemple.

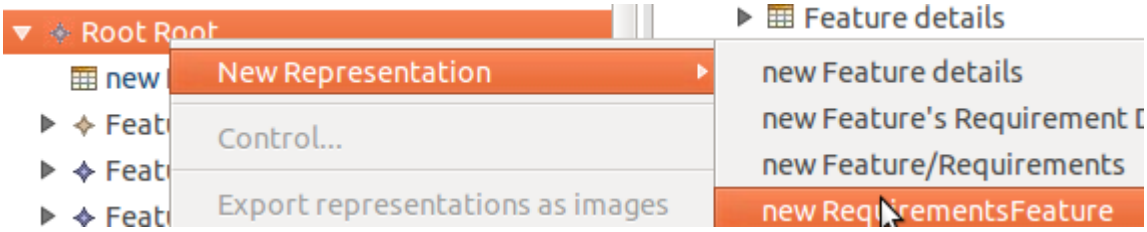


Figure 71 – Création d'une instance du tableau.

Toutes les représentations disponibles depuis Root sont indiquées. Ce sont ici les tableaux de la vue Specification Specs.

IV.2.2.2 **Spécification de l'éditeur de configuration**

IV.2.2.2.1 **Implémentation du système de contraintes**

L'objectif est d'assister l'utilisateur dans l'activité de configuration. L'impact des différentes contraintes introduites dans le modèle peut être difficile à évaluer. L'outil devra empêcher l'utilisateur de faire des choix irréguliers ou à défaut lui indiquer lorsqu'il en a fait un. Cela passe par la mise en place de contraintes OCL et la création d'un outil de sélection de features. Le premier point a déjà été implémenté, il nous reste à aborder la création de l'outil de sélection dans Sirius. Cet outil devra aussi permettre d'appliquer les répercussions d'une sélection de features sur les autres. Par exemple, choisir une feature devra entrainer la sélection automatique de ses features filles obligatoires.

J'ai créé un outil de sélection nommé Select/Unselect dans la section « Modifiers » de ma représentation « featureTree ».

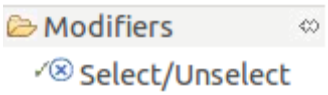


Figure 72 – Aperçu de l'outil de sélection.

S'agissant d'appliquer une modification de l'attribut isSelected sur l'élément que l'utilisateur clique, je me place dans le contexte de cet élément (opération Change Context).

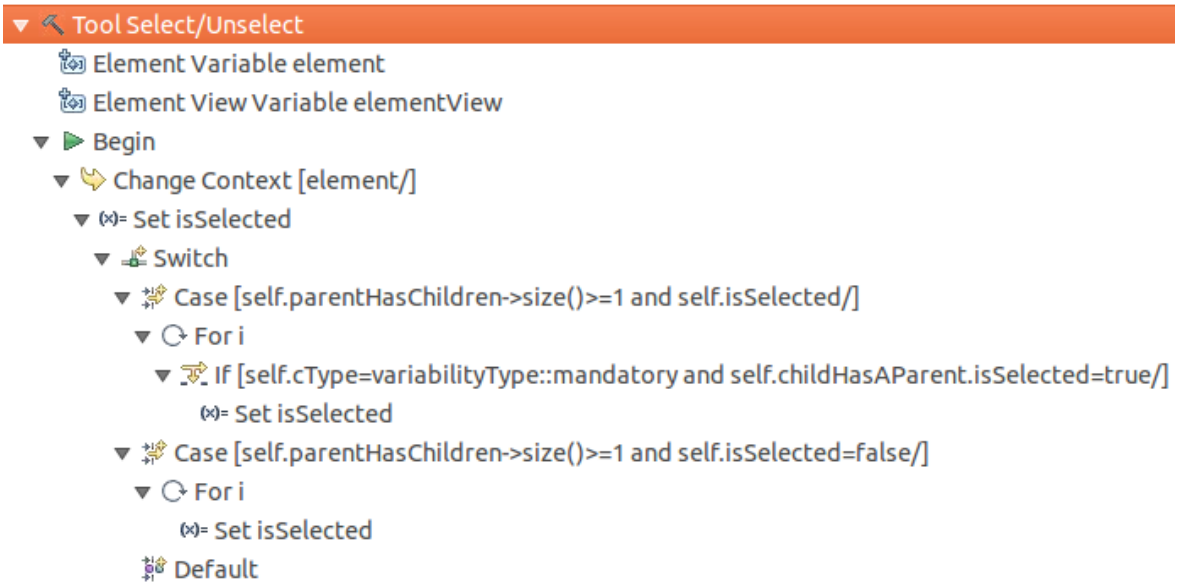


Figure 73 – Spécification de la répercussion d'une sélection

Puis je spécifie une opération de définition de valeur de la variable isSelected. L'expression de la valeur est [not self.isSelected/] pour simplement inverser la valeur de l'attribut isSelected de l'élément graphique que l'utilisateur clique. Puis j'effectue les répercussions possibles dans la structure Switch grâce à deux « case » :

- Le premier s'applique si l'instance cliquée a au moins un enfant et est sélectionnée. Une boucle For est spécifiée pour itérer parmi l'ensemble des enfants.

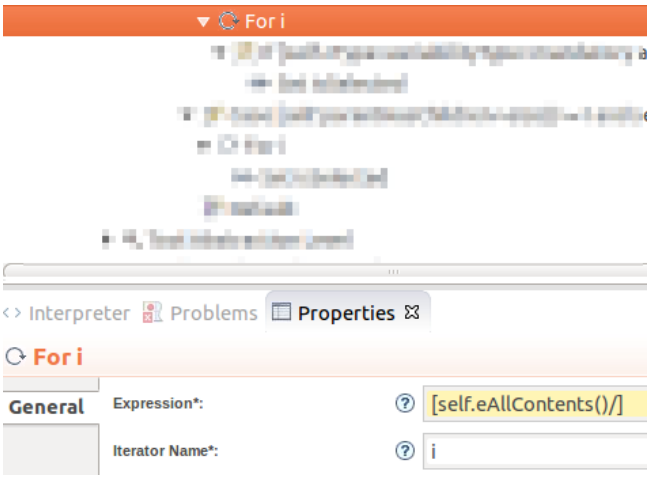


Figure 74 – Spécification de l'itération parmi les enfants.

Pour chacun d'entre eux, s'il est obligatoire et si son parent est sélectionné (redondance par simple prudence) je change la valeur de son attribut isSelected à [true/].

- Le second s'applique s'il a au moins un enfant et qu'il n'est pas sélectionné. La même boucle For est spécifiée et cette fois-ci, ce qui suit s'applique à tous ses enfants. Tous les enfants voient leur attribut isSelected passer à false.

D'autres mécanismes permettent encore d'automatiser certaines sélections. Dans l'outil de définition du lien parent/enfant, j'ajoute le changement de valeur de isSelected de l'instance dont on définit le nouveau Parent à :

- True lorsqu'elle est obligatoire et que le nouveau parent est un FeatureGroup ou un FeatureNode sélectionné.

- False lorsque le nouveau parent est un FeatureGroup ou un FeatureNode non sélectionné.

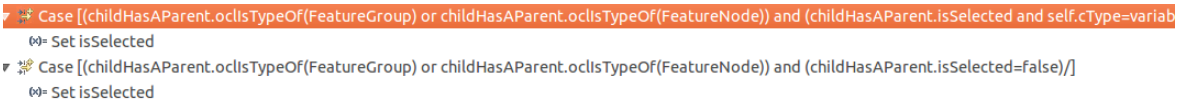


Figure 75 – Implémentation de la logique de sélection dans la création de relation Parent/enfant.

Il serait aussi tout à fait possible de modifier l'attribut isSelected des features impliquées dans des contraintes de type requires et excludes. Mais je ne souhaite pas le faire car il faudrait aussi éventuellement changer le statut des parents et des enfants de ces features pour rester cohérent. Le changement pourrait rapidement impacter de nombreuse features et si l'utilisateur change d'avis ou s'est simplement trompé de feature impliquée dans cette

relation, le rétablissement des valeurs souhaitées peut être fastidieux. Je laisse ici plutôt le soin à l'utilisateur de valider son diagramme. Les contraintes OCL permettront d'afficher les incohérences.

IV.2.2.3 **Spécification du dérivateur**

On souhaite obtenir le modèle du produit dérivé en supprimant les features non sélectionnées du modèle de pré-dérivation. Pour cela, je spécifie deux nouvelles représentations, un diagramme et un arbre.

Concernant le diagramme, j'ai repris les mêmes spécifications que pour mon premier diagramme (permettant d'obtenir le modèle de pré-dérivation) en y ajoutant un simple filtre pour l'affichage des features (au sens large du terme). Ne voulant afficher que les features sélectionnées, j'ai renseigné le champ « Precondition Expression » présent dans les options avancées des spécifications des nœuds. Par exemple, pour n'afficher que les FeatureGroups sélectionnées :

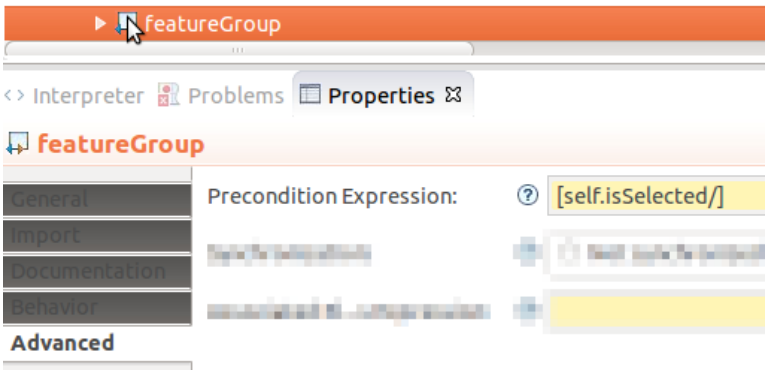


Figure 76 – Filtre d'affichage des features du produit dérivé.

Seules les instances dont l'attribut isSelected est définit à true seront affichées. Avec mon modèle d'exemple, j'ai sélectionné grâce à l'outil de sélection, le noyau et le delta utilisation Nomade. Les éléments optionnels ne l'on pas été. Le résultat est le suivant :

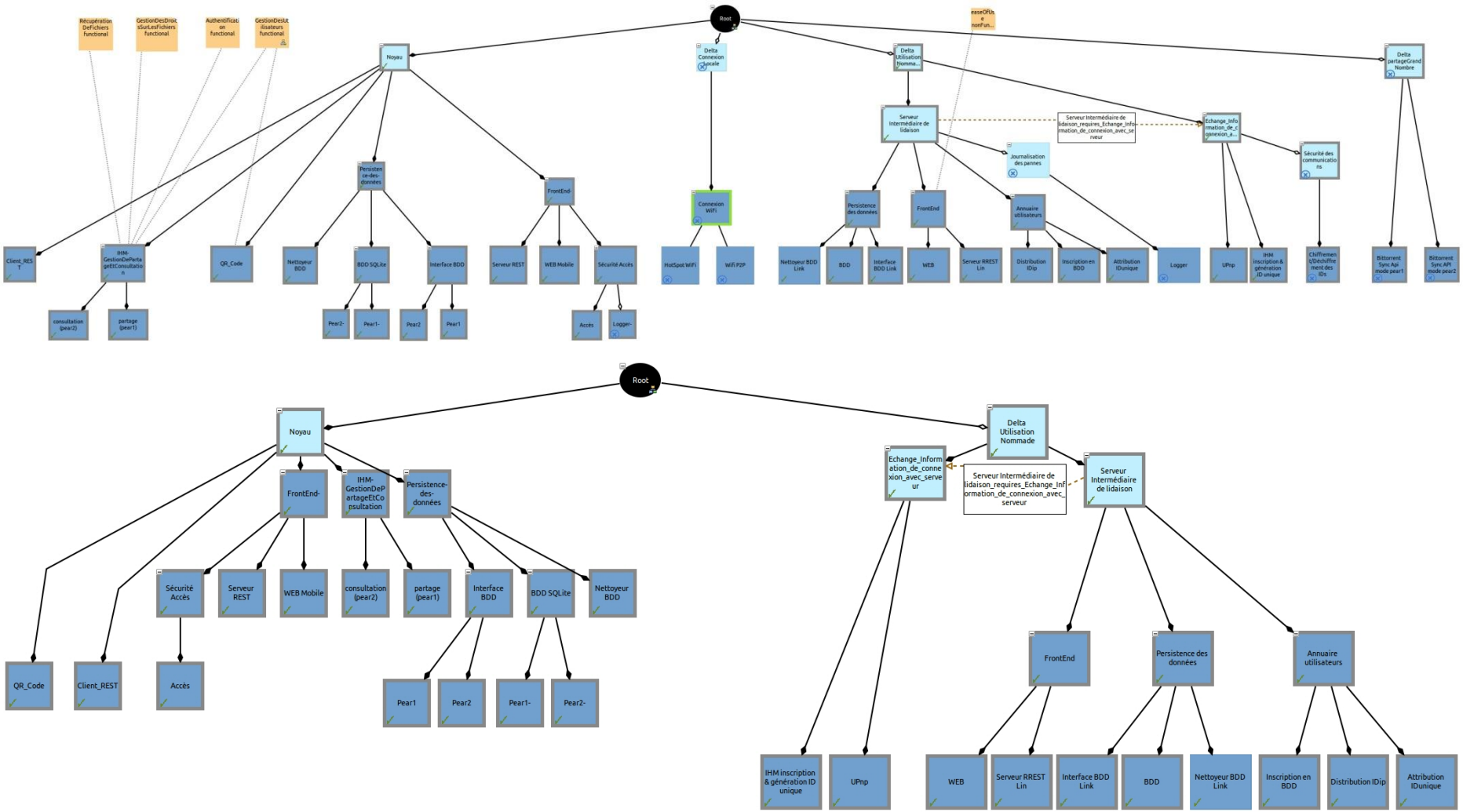


Figure 77 – Du modèle de pré-dérivation au modèle du produit dérivé, exemple de dérivation.

On observe par exemple l'absence des features de journalisation des pannes qui n'ont pas été sélectionnées dans le modèle de pré-dérivation. J'ai fini par créer une nouvelle représentation pour plus de clarté. Un arbre permettant d'avoir une représentation concise de la hiérarchie des features. Le détail de sa spécification n'apporte pas beaucoup. Un seul point diffère de la logique des autres représentations. La récursivité dans l'affichage des features et leur hiérarchie est établie par la création d'un nouveau « Tree Item » (auquel il faut associer l'élément sémantique que l'on veut représenter, ici la classe Feature) dont on renseigne deux propriétés : Semantic Candidate Expression, dont je définis la valeur à l'aide de l'expression [parentHasChildren/] puis le champ Reused Tree Item mapping que je fais pointer vers lui-même, ce qui met en place la récursivité nécessaire à la création d'un arbre de profondeur indéfinie. Ici le résultat est le suivant :

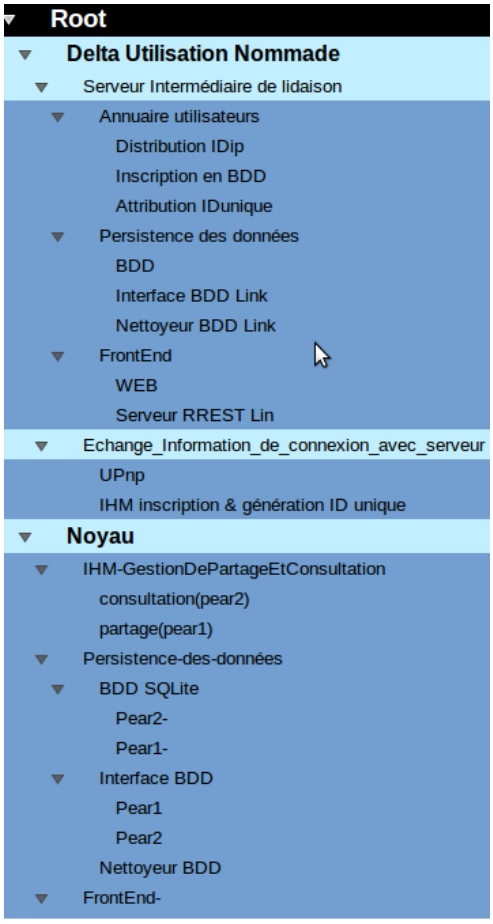


Figure 78 – Arbre des features du produit dérivé.

J'ai repris le même code couleur indiquant le niveau d'abstraction que sur les diagrammes.

IV.2.3 Distribution et utilisation du modeleur :

IV.2.3.1 **Distribution :**

Pour pouvoir être utilisé par un tiers, le modeleur peut être exporté comme n'importe quel plug-in Eclipse puis distribué et importé par l'utilisateur final. Il fonctionne toutefois dans le cadre strict de l'environnement Eclipse. Il faut que l'utilisateur l'installe avec les « modelling tools » et Sirius. Un package Eclipse modelling tools est disponible en téléchargement sur le site du projet Eclipse. Il faut encore installer Sirius, OCLinEcore et ecoreTools. Une version de java 1.6 ou supérieur doit être présente sur la machine.

IV.2.3.2 **Utilisation :**

Pour que l'utilisateur puisse obtenir autant de configurations qu'il le souhaite, il doit créer une nouvelle instance de notre métamodèle de la même manière que ce qui a été fait pour l'instance d'exemple. Il peut le faire depuis le projet importé où se trouve notre modèle d'exemple. Il devra à nouveau spécifier les features de sa ligne de produit qu'il souhaite modéliser.

V Conclusion

V.1 Et maintenant ?

V.1.1 Prochains objectifs

J'ai réussi à obtenir une architecture de référence de qualité et à fournir un outil de gestion de la variabilité. Dans le cadre de ces recherches et de ce mémoire nous sommes cantonnés à un haut niveau d'abstraction. Il reste à pouvoir obtenir un framework complet pour assister l'utilisateur dans l'élaboration de la ligne de produit, la dérivation de produits et la génération de code. Plusieurs cas sont à envisager :

- L'utilisateur n'a pas de code documenté et l'idée est de partir du code des applications pour obtenir sa ligne de produits. Après dérivation pour obtenir un produit, réutiliser le code déjà existant qui correspond à chaque feature membre du nouveau produit dérivé. C'est l'objectif le plus ambitieux à satisfaire. Il faudrait pouvoir identifier les composants et les features à partir du code, puis constituer une bibliothèque contenant ce code en blocs indépendants.

- L'utilisateur a un code documenté par un diagramme UML de composants et/ou de classes. Il pourra utiliser la méthode fournie ici. Le problème de la composition sera abordé au chapitre V.1.2.

Le framework complet devra donc permettre de :

- Eventuellement, partir de code déjà existant de plusieurs applications pour en obtenir les principaux composants regroupés par fonctionnalités et établir la matrice de connectivité du noyau et des variants de l'architecture de référence de la ligne de produit.

- Principalement, d'établir un moyen de générer du code à partir des modèles de produits dérivés.

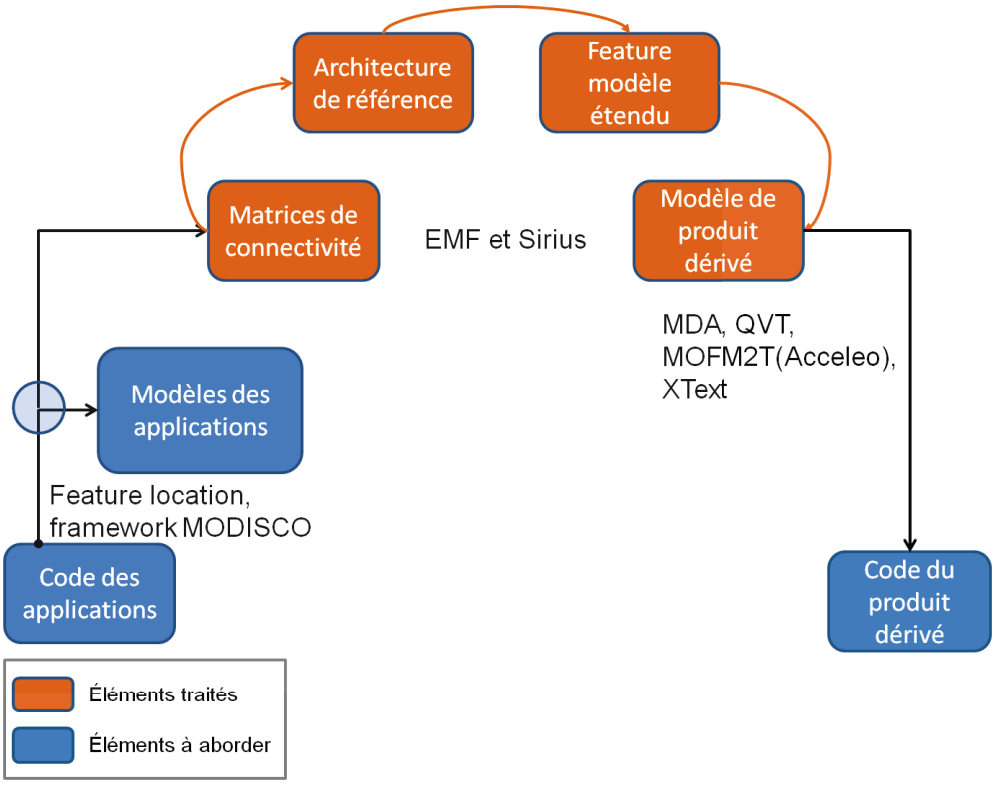


Figure 79 – Framework complet et technologies envisagées dans sa réalisation.

V.1.2 Pistes

V.1.2.1 Obtention de modèle depuis le code

Pour le premier objectif, des méthodes de feature location pourraient être envisagées pour assister l'utilisateur dans l'élaboration d'un éventuel modèle pour chaque application, puis d'une matrice de connectivité. Une étude des différentes techniques de feature location

peut être trouvée dans ce document (12). Il faut de plus avoir un code qui se prête à l'isolation aisée des composants par fonctionnalités remplies. Un code dont les classes et méthodes sont séparés par aspects. Un code éventuellement tel que le recommandent les méthodes de programmation orientées aspects faciliterait le travail. En l'état actuel de mes connaissances, je ne suis pas confiant quant à la faisabilité de ceci. Il me semble que rien ne remplace la connaissance du développeur et de l'architecte dans l'obtention d'un modèle fiable décrivant chaque application. La situation idéale étant de disposer d'une documentation type diagramme de composants UML.

V.1.2.2 **Génération de code depuis un modèle de produit dérivé**

Pour le second objectif, il est envisageable à l'avenir de fournir un framework permettant de partir de diagrammes de composants pour obtenir la ligne de produits et établir le modèle du produit dérivé, qui, avec un générateur de code (Acceleo par exemple), permettra d'obtenir le squelette du code de notre produit dérivé.

Le niveau d'abstraction retenu par la méthode permet de poser sereinement les bases d'une solution de gestion de la variabilité. Toutefois dès que l'on voudra passer des modèles de produits dérivés à leur implémentation, rien ici ne permet d'anticiper l'impact des compositions au niveau des composants et du code. Lorsque l'on va agréger des composants, ils ne sont pas exactement les mêmes dans chaque application dérivée. Ils ne rendront pas exactement les mêmes services, ni ne communiqueront exactement avec les mêmes autres composants. L'exemple le plus parlant est l'IHM. Celle-ci devra comporter plus ou moins d'éléments graphiques et d'actions attachées en fonction des fonctionnalités et des composants sélectionnés. Le choix de certains composants nécessitera de modifier l'IHM pour permettre à l'utilisateur l'accès à la fonctionnalité couverte par le composant. A un autre niveau moins évident pour l'utilisateur, les interfaces devront être modifiées ou de nouvelles créées. Une interface ne communiquant d'abord qu'avec un autre composant, devra être modifiée pour communiquer avec plusieurs, selon des contrats peut-être différents, ce qui peut rendre la création d'une nouvelle interface judicieuse. C'est la principale difficulté à résoudre pour aller plus loin. Les feature models et leurs représentations restent très abstraites en soit. Ce ne sont que des labels représentant des composants. Il faut arriver à associer ces features à d'autres représentations moins abstraites

soit en passant par d'autres représentations, elles-mêmes finalement associées à du code, soit en les associant directement à du code :

Pour la première alternative, le raffinement ou la transformation de modèle est alors orthogonal à la modélisation de la variabilité. En tant que tel, on peut utiliser le feature modeling pour la gestion de la variabilité et les MDA (pour l'anglais Model Driven Architecture) pour la transformation. Certaines approches proposent d'associer chaque feature à un modèle ou à un élément d'un modèle. On peut ainsi configurer un produit au plus haut niveau d'abstraction et ensuite transformer les modèles liés aux features en modèles plus fins (à vrai dire, les deux opérations sont commutatives) jusqu'à arriver au code. Ina Schaefer (13) pose les concepts du delta-modeling, indépendamment de toute solution technique afin de l'implémenter. L'application qui serait faite dans notre cas, serait un cas spécifique du delta-modeling. Le delta-modeling pose que le noyau de la ligne de produit peut être tout produit valide. Nous poserions que ce produit serait forcément celui tel que défini avec l'approche de N. Levy *et al.* Les delta-modèles spécifient les changements faits au modèle noyau par des additions, des modifications ou des suppressions de fragments de modèles. Une condition d'application est attachée à chaque « delta-model ». Elle détermine à quelles configurations de features doit être appliqué les changements que le « delta-model » spécifie. Les modèles du noyau et des variants sont raffinés indépendamment. A partir d'un modèle initial, à un haut niveau d'abstraction, des modèles affinés sont construits pour décrire plus en détail les aspects du produit dérivé. A chaque niveau de modélisation, la variabilité est capturée par un modèle noyau et un ensemble de delta-modèles.

Dans la seconde alternative, l'association directe de features à du code tout en anticipant l'impact de l'ajout de tel ou tel composant dans un produit est aussi possible comme le démontre l'approche Delta-Oriented Programming (DOP) (14). Une ligne de produit est représentée par un core-module et des delta-modules. Dans notre cas, le core-module serait le noyau de notre architecture de référence. Le core module est d'abord implémenté de manière classique. Les delta-modules spécifieront les changements à apporter au core-module par des opérations d'ajout, de suppression ou de modification de code. Les conditions d'application des delta-modules permettent de gérer les combinaisons de features établies dans le modèle du produit dérivé de manière explicite. L'implémentation d'un produit, d'une configuration de feature particulière, est réalisée en appliquant de manière

incrémentielle au core-module tous les delta-modules dont la condition d'application est vérifiée dans cette configuration. Ce serait l'alternative que j'essaierais d'adopter car elle me semble moins lourde en terme d'étapes pour l'obtention d'un code. Un exemple d'implémentation de cette approche est donné par le projet DeltaJ qui permet d'organiser les classes en modules de deux types : core et delta. Le core module est un simple ensemble de classes classique alors que les deltas modules regroupent l'ensemble des opérations qui permettent d'ajouter/modifier/supprimer des classes déclarées dans d'autres modules delta ou dans le core module. (cf. (15) pour plus de détails)

V.1.2.1 **Proposition de modèle d'un framework complet**

Pour réaliser le framework tel que décrit par la Figure 79 – Framework complet et technologies envisagées dans sa réalisation., de prime abord, j'envisagerais de créer un métamodèle global de support à la spécification d'un modeleur sous Sirius. Sirius permet de créer des « extensions java », du code pour la manipulation des concepts du modèle sur lequel repose la spécification. (Voir figure ci-après).

On voit très clairement quelles parties sont dédiées à quelle étape. On peut facilement imaginer créer une vue Sirius pour chacune. Des relations sont mises en place entre ces préoccupations, ces étapes de la méthode. Le point commun entre toutes est la partie qualité : les exigences de qualité et les exigences fonctionnelles et non fonctionnelles sont liées à la fois aux composants des applications spécifiées et aux features du feature model étendu. On peut imaginer créer sous Sirius l'ensemble des tableaux nécessaires à la réalisation de la partie III assez facilement en ne spécifiant que les exigences à l'origine des features et leurs liens avec les exigences du modèle de qualité. Ce pourrait être une base permettant de suggérer à l'utilisateur une équivalence de composants entre les différentes applications qui formeront la base d'une SPL.

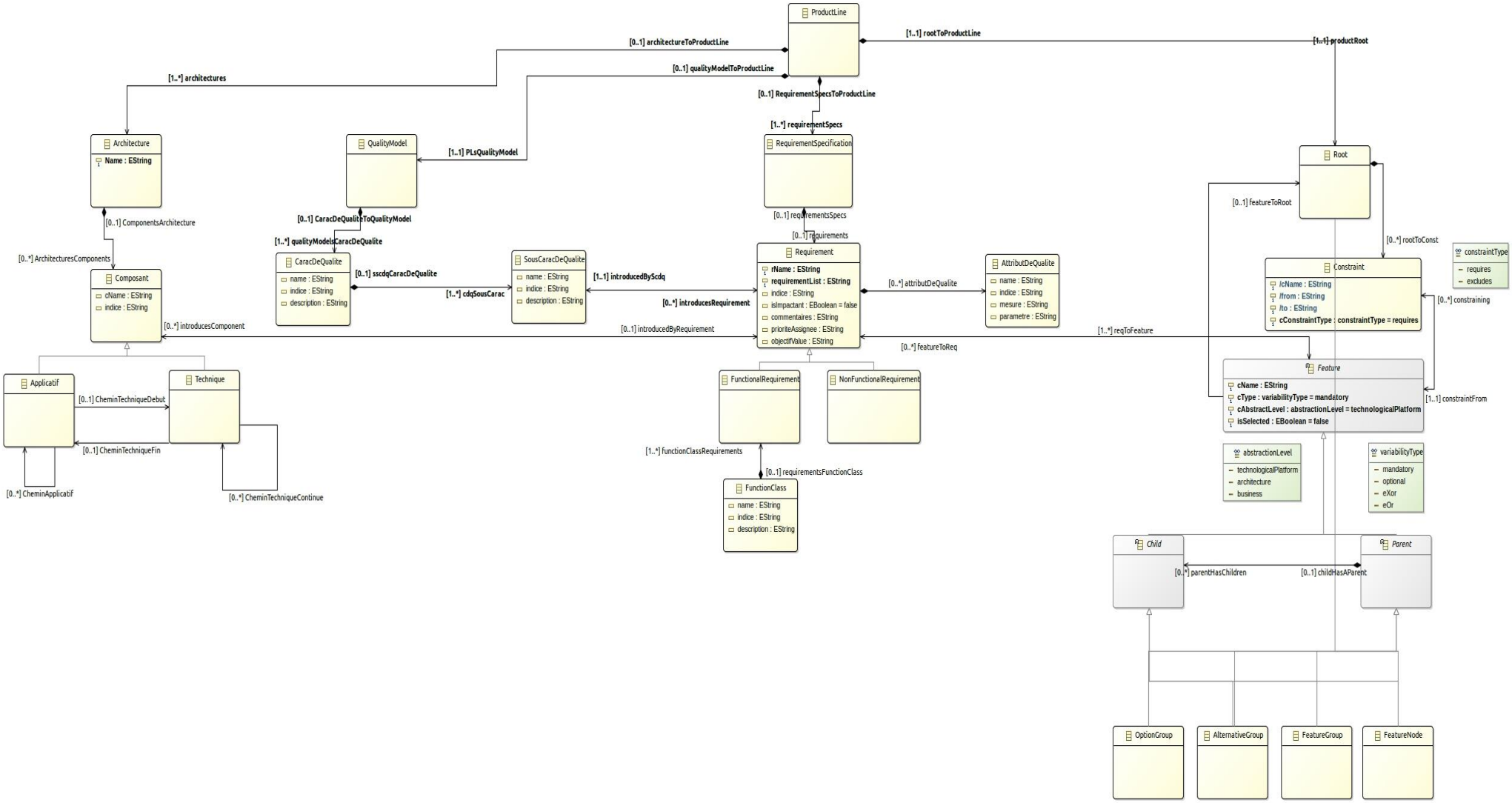


Figure 80 – Proposition de métamodèle pour un framework complet.

Le framework basé sur ce modèle permettrait d'accompagner l'utilisateur dans l'ensemble des étapes décrites dans ce mémoire, et de le mener à la génération de code. Une intégration de DeltaJ1.5 est a priori possible après un rapide examen de projets d'exemple fournis avec ce plugin eclipse. Il ne s'agit que de fichiers texte ou XML indiquant les modifications à faire. Certaines indications pourraient être déduites automatiquement du modèle sur lequel l'utilisateur travail. DeltaJ a été implémenté en utilisant le framework XText.

V.2 Travail effectué :

V.2.1 Observation sur les méthodes à appliquer

L'objectif donné était d'appliquer les recherches de N. Levy *et al.* dans l'établissement d'une ligne de produit. La méthode permet de partir de l'architecture de plusieurs applications existantes et d'obtenir une architecture de référence d'une ligne de produit. En partant des architectures données, nous devons en établir les matrices de connectivité. Ces matrices sont le support de travail principal de cette méthode, et la modélisation qui en était faite auparavant (diagramme de composants UML par exemple) doit simplement pouvoir supporter l'analogie avec un graphe orienté dont les nœuds sont des composants logiciels et les arcs orientés un lien de communication entre eux. Ceci a l'avantage de permettre à cette méthode d'être appliquée à la majorité des applications documentées, peu importe la modélisation de leur architecture.

Certaines difficultés ont été éludées puisque le travail était fait en ayant déjà en tête l'idée de créer une ligne de produits. Les composants applicatifs ayant le même rôle avaient le même nom et étaient facilement identifiables au sein des différentes applications. Ceci n'a pas permis d'évaluer la difficulté liée à la méthode dans la détermination des équivalences. De même, aucune variabilité intra-noyau tenant à des chemins techniques différents n'a été rencontrée. Toutefois la solution proposée dans la gestion de la variabilité laisse à penser qu'aucune difficulté n'aurait été rencontrée dans leur gestion. La méthode proposée pour obtenir l'architecture de référence encadre strictement le travail de l'architecte logiciel. Toutefois, une étape essentielle m'a paru floue de prime abord : la détermination des composants techniques et applicatifs. C'est un élément essentiel pour pouvoir appliquer la méthode. Le choix de la répartition des composants dans les deux classes A et T influence le déroulement de ce qui suit. Une mauvaise répartition pourrait conduire à une proposition

pauvre en matière de réutilisation, peut-être avec beaucoup ou trop peu de composants communs. Pour déterminer ces classes de composants, la méthode laisse le champ libre à l'architecte logiciel. Toutefois je pense que c'est préférable. Décrire ce qu'est un composant technique de manière plus restrictive serait un risque puisqu'on ne peut envisager toutes les d'architectures possibles. L'essentialité d'un composant ne se juge correctement que dans le contexte pour lequel les applications ont été développées. Il faut donc orienter sans restreindre. C'est ce qui a été fait dans la méthode fournie. Une fois la légère difficulté exprimée plus haut dépassée, la méthode se déroule facilement. Il suffit de suivre les étapes données. Le résultat obtenu est probant : dans le cadre des applications servant de base à ce mémoire, les composants sont facilement identifiables et le résultat peut être vérifié.

Enfin cette méthode demande pour le moment une intervention manuelle importante. Toutefois, dans le cadre d'un framework complet, des suggestions pourraient être faites à l'utilisateur pour le guider dans l'élaboration de sa ligne de produits. Le travail concernant la qualité m'apparaît finalement central. Certes, la qualité est une préoccupation essentielle en elle-même, mais dans le cadre d'une méthode globale, depuis la détermination d'une architecture de référence jusqu'à la génération de code, la méthode telle que proposée a l'énorme avantage de permettre de créer un lien de traçabilité entre fonctionnalités attendues, exigences de qualité (requirements) et composants de l'architecture. C'est certainement davantage grâce à ces liens et à la place centrale des requirements qu'il pourra être possible d'assister l'utilisateur à la détermination des composants applicatifs/techniques et à la reconnaissance des composants communs dans un framework complet.

V.2.2 Apports

Je pense avoir démontré :

- que la méthode d'obtention d'une architecture de référence est applicable, tout en apportant certains outils pour la rendre plus facilement utilisable (outil JAVA et méthode de masque).

- que l'application de la partie sur la qualité est en réalité une partie intégrante de la définition d'une SPL, contrairement à ce que j'avais initialement anticipé. Ce n'est pas une simple surcouche, un travail additionnel pour garantir la qualité des futurs produits. Elle s'inscrit très logiquement dans la démarche d'établissement de la SPL et de la gestion de la variabilité en permettant d'établir un lien de traçabilité entre les exigences utilisateurs et les composants

à implémenter pour les satisfaire. Chaque composant est justifié et documenté, ce qui facilite la création du feature diagram étendu.

J'ai adapté la méthode proposée sur le travail de qualité pour pouvoir l'appliquer après avoir obtenu l'architecture de référence en première partie. La méthode propose notamment d'établir le noyau commun des fonctionnalités de l'ensemble des systèmes étudiés (par l'intersection de leurs fonctionnalités) que l'on liera à certains composants. C'est une méthode descendante. Dans notre cas, le travail de classification des fonctionnalités lors de la détermination d'invariant et de variant fonctionnel n'était plus nécessaire une fois le noyau et les variants obtenus en première partie. J'ai recommencé le travail d'analyse des besoins indépendamment des composants que j'avais déjà établis en première partie. Lorsqu'un besoin recensé était déjà satisfait par un composant, j'ai simplement documenté l'architecture par un lien de traçabilité. J'ai aussi pu établir si la fonctionnalité était commune ou non : de par l'appartenance du composant auquel elle se rattache au noyau ou à un variant. Lorsqu'un besoin n'était pas satisfait, un nouveau composant était ajouté. J'ai donc toutefois gardé cet esprit de démarche descendante

Au niveau gestion de la variabilité, j'espère avoir apporté une solution satisfaisante en l'état, ou ouvert une piste viable dans l'utilisation des outils de modélisation Eclipse et Sirius, à défaut de l'utilisation du framework pressenti initialement.

J'ai fini par atteindre le but fixé dans le sujet qui m'était donné, mais sans parvenir à l'étape suivante, davantage par manque de temps que par impossibilité. J'ai toutefois présenté dans les sections précédentes ce qu'il reste à faire pour obtenir un framework complet, depuis la détermination des architectures des applications jusqu'à la dérivation des produits et la production de code. Ce n'est que tardivement que j'ai pu effleurer les solutions qui existent afin de permettre la réalisation de ces tâches.

V.2.3 Difficultés

La première difficulté dans le déroulement de ce stage fut pour moi l'imprégnation de la matière. Il existe énormément d'articles concernant le sujet, et bien qu'il ne soit pas si jeune, je n'ai pas trouvé de consensus quant à la manière d'établir une SPL, ni quant à la gestion de la variabilité, ni quant au champ d'application des méthodes proposées. Certaines proposent de créer une SPL en partant de rien, d'autres en partant du code, etc. Il y a encore un travail d'institutionnalisation à faire : une classification claire des différentes écoles, un

retour d'expérience suffisant pour établir laquelle semble être la mieux adaptée à quel cas. On peut facilement se perdre. Le temps nécessaire à une maîtrise partielle du sujet a pris du temps.

L'application manuelle des méthodes proposées pour l'établissement de l'architecture de référence, mais aussi pour les questions de qualité demandent beaucoup de temps.

La méthode de dérivation devait se baser sur l'outil SEQUOIA du CEA. Ce framework balise le chemin de la gestion de la variabilité et de la dérivation de produits. Il n'est pas disponible au public. J'ai donc dû étudier les différentes méthodes d'expression de la variabilité et de dérivation pour pouvoir mettre ces étapes en œuvre dans ce mémoire, avec les difficultés déjà mentionnées plus haut. Malheureusement je n'ai trouvé aucun autre framework ou outil utilisable et permettant de rester dans la continuité de la méthode apportée par N. Levy *et al.* Ce fut aussi un travail très chronophage et frustrant. J'ai heureusement trouvé parmi les projets Eclipse le projet EMF, permettant de spécifier des modeleurs dédiés à un modèle... Puis mes recherches m'ont mené vers Sirius permettant de s'affranchir des lourdeurs et de la longue courbe d'apprentissage d'EMF. Bien que l'outil Sirius soit opérationnel, de nombreux aspects sont encore à travailler pour le rendre complément fiable et moins frustrant :

- Il arrive que l'outil repositionne et redimensionne les éléments graphiques de notre graphe alors même que leur spécification n'a pas été modifiée. Ce qui fait perdre pas mal de temps. Si bien que j'ai fini par ne plus replacer les éléments graphiques avant d'avoir terminé totalement de spécifier leur représentation et de peupler le modèle qu'ils représentent.

- Certaines incohérences ou discontinuités dans la méthode de spécification. Parfois la spécification d'un attribut se fera via une expression Acceleo ou OCL dans un champ interprété, d'autres fois, par l'utilisation de variables fournies.

- L'alternance entre les contextes sémantiques et graphique ne m'a pas paru si claire. On ne sait pas toujours dans quel contexte on se trouve. Pour ma part, seul le contexte sémantique m'a servi dans mes expressions.

- A certains endroits, la notion de conteneur tel qu'il nous est demandé d'indiquer n'est pas si claire. Elle m'a l'air davantage tenir du conteneur XML (sachant que les métamodèles et leurs instances sont stockés sous forme de fichiers XML) que d'une relation de contenance éventuellement présente dans le métamodèle Ecore que l'on spécifie.

J'ai eu plaisir à établir mon propre DSL et son modeleur. La découverte de ces possibilités est malheureusement arrivée tardivement dans l'exécution de ce mémoire qui aurait été bien différent. Les premières étapes de modélisation, de détermination des chemins techniques, de l'élaboration des matrices de connectivité, de la prise en compte de la qualité auraient été inclues dans ce modeleur pour fournir un outil global accompagnant la démarche. L'assistance dans l'identification des composants depuis le code me semble toutefois difficile à mettre en place.

Pour le reste, je n'ai pas pu mettre en place ce framework plus complet par manque de temps. Je n'ai vu le moyen de le mettre en œuvre qu'après avoir découvert Sirius. Bien que techniquement possible, la réalisation d'un framework complet est une tâche trop importante pour pouvoir être exécutée dans le temps imparti. J'ai fini d'en être convaincu en étudiant l'outil FeatureIDE et son développement. Il a demandé 2 ans à plusieurs professeurs et leurs étudiants successifs pour obtenir un framework capable de modéliser un feature model et de composer des applications en suivant les contraintes indiquées et qui permette de générer du code.

Enfin, la génération de code depuis un modèle n'est pas encore une réalité : j'étais d'abord très sceptique quant à l'utilité des modèles et la génération de code depuis un modèle. On ne peut d'ailleurs toujours pas générer un programme complet depuis un modèle sans avoir à modifier le code généré. Mais la problématique de la réutilisation dans l'idée de diminuer le coût de développement s'impose d'elle-même face à des systèmes de plus en plus complexes. L'exemple de Photoshop est assez parlant : ce sont 20 millions de lignes de codes, dont seulement une trentaine de classes forment le cœur du logiciel. Le reste ne sert qu'à la gestion de la persistance et toutes ces choses accessoires mais qui lui sont nécessaires. La factorisation de code en ensembles indépendants et faiblement couplés est ce vers quoi il faut tendre pour permettre la réutilisation. Puisqu'il nous faut coder des systèmes de plus en plus complexes, la simplification du code par la factorisation (décomposer un ensemble complexe en sous-ensembles plus faciles à appréhender) et la simplification de la conception par l'abstraction sont indispensables et suivent la tendance historique du développement logiciel. Dans les années 60 l'usage des langages assembleur a été remplacé par des langages procéduraux, de plus haut niveau. Les premiers témoins furent septiques. Les compileurs n'étaient pas aussi bons qu'aujourd'hui même si certains langages permettaient au programmeur de donner des indications au compileur pour l'aider dans sa tâche de traduction

de code. Beaucoup étaient inquiets quant à la qualité du code généré, moins efficace en termes de performances qu'un code assembleur écrit à la main. Cependant l'avantage d'un langage plus haut niveau, permettant d'écrire un système plus complexe a pris le pas sur les considérations de performances. Avec le temps les compilateurs sont devenus plus performants et plus personne aujourd'hui n'envisagerait de proposer un nouveau programme en assembleur hormis certains cas particuliers. Le Model Driven Development (MDD) me semble être l'avenir du développement logiciel. Il permet déjà d'obtenir des gains de productivité grâce à de bons outils de transformation. Il faut pour le moment toujours assister l'outil dans la transformation d'un modèle vers du code mais je pense que l'avantage de travailler à un haut niveau d'abstraction devrait encore une fois pousser les développeurs à changer de centre de préoccupation, en passant du code vers les modèles.

Bibliographie

1. http://fr.wikipedia.org/wiki/G%C3%A9nie_logiciel. *Wikipedia france.* [En ligne] 2004.

2. Linden, Frank van der. *Software Product Families in Europe : The Esaps & Café Projects.* s.l. : Philips Medical Systems, 2002.

3. http://fr.wikipedia.org/wiki/Architecture_logicielle. *Wikipedia France.* [En ligne] http://fr.wikipedia.org/wiki/Architecture_logicielle.

4. Levy, Nicole et Pollet, Yann. *Définition d'une architecture de référence à l'aide de matrices de connectivité,* rapport interne CEDRIC, CNAM, Nov 2014.

5. Levy, Nicole, Losavio, Francisca et Pollet, Yann. Architecture et qualité de systèmes logiciels. *Architectures logicielles : principes, techniques et outils.* Decembre 2013, Hermès Sciences-Lavoisier, pp.265-302, (isbn : 978-2-7462-4517-4).

6. *Preparing for Product Derivation: Activities and Issues.* Padraig O'Leary, Fergal McCaffery, Ita Richardson, Steffen Thiel. 2009.

7. *Case Study of Feature Location Using Dependence Graph.* Kunrong Chen, Vaclav Rajlich. Limerick, Ireland : IEEE Computer Society, 2000. ISBN 0-7695-0656-9.

8. *Preparing for Product Derivation: Activities and Issues.* O'Leary, Padraig, et al. Bulgarie : s.n., 2009.

9. KC Kang, SG Cohen, JA Hess, WE Novak, AS Peterson. Feature-oriented domain analysis (FODA) feasibility study. 1990.

10. K. Czarnecki, U. W. Eisenecker. *Generative Programming: Methods, Tools, and Applications.* New York : ACM Press, 2000.

11. https://fr.wikipedia.org/wiki/Eclipse_Modeling_Framework. [En ligne]

12. Bogdan Dit, Meghan Revelle, Malcom Gethers andDenys Poshyvanyk. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process.* 2013, Vol. 25, 1.

13. *Variability Modelling for Model-Driven Development of Software Product Lines.* Schaefer, Ina. Linz, Autriche : s.n., 2010. In Fourth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2010).

14. Ferruccio Damiani, Ina Schaefer, Lorenzo Bettini, Nico Tanzarella. Delta-oriented programming of software product lines. [éd.] Jan Bosch Jaejoon Lee. *SPLC'10 Proceedings of the 14th international conference on Software product lines: going beyond.* s.l. : Springer-Verlag Berlin, 2010, pp. 77-91.

15. Ferruccio Damiani, Ina Schaefer, Jonathan Koscielny, Lorenzo Bettin, Sandro Schulze, Sönke Holthusen. *DeltaJ 1.5: Delta-oriented Programming for Java 1.5.* s.l. : Association for Computing Machinery, 2014. pp. 63-74. ISBN : 978-1-4503-2926-2.

16. Losavio, Francisca, et al. Graph Modelling of a Refactoring Process for Product Line Architecture Design.

Liste des figures

Liste des tableaux

Architecture, qualité et lignes de produits : Etablissement d'une architecture de référence dans un domaine applicatif.

Mémoire d'ingénieur C.N.A.M., Paris 2015

RESUME

Dans un même domaine fonctionnel, il peut exister de nombreux systèmes aux fonctions très proches. Seules les exigences non fonctionnelles et certaines contraintes vont apporter certains points de variations à ces applications. Dans la pratique, il en résulte des solutions architecturales différentes pour résoudre des problèmes proches. Le génie logiciel aborde donc ces problèmes liés à la conception et au développement de lignes ou de familles de produits, à la recherche d'économie de temps et de coûts par la planification de la réutilisation de composants logiciels.

Dans ce contexte, ce mémoire a pour objectif d'expérimenter la méthode élaborée par l'équipe de chercheurs en collaboration avec le professeur Levy pour l'établissement d'une architecture de référence d'une famille de produits, tout en garantissant la qualité des systèmes développés. Le but de la méthode proposée est de valoriser les connaissances acquises pour la conception de nouveaux systèmes proches. Elle donne des outils pour gérer les exigences et la variabilité des systèmes sur des bases formelles, par des matrices de connectivité.

Ce mémoire cherche à illustrer les avantages et inconvénients apportés par la méthode en l'appliquant à une étude de cas, tout en proposant des outils pour faciliter sa mise en œuvre.

Mots clés : architecture logicielle, dérivation de produit, domaine fonctionnel, exigences, famille de produits, gestion de variabilité, ligne de produits, matrice de connectivité, qualité.