CENTRE UNIVERSITAIRE - SALHI AHMED - DE NAAMA

INSTITUT DES SCIENCES & TECHNOLOGIES

DÉPARTEMENT DE MATHÉMATIQUES ET INFORMATIQUE

C.U.N
المركز الجامعي بالنعامة
CENTRE UNIVERSITAIRE DE NAAMA

MÉMOIRE DE FIN D'ÉTUDES

Pour l'obtention du diplôme

MASTRE EN INFORMATIQUE

Spécialité : Système d'information

Thème

Développement des applications mobiles à l'aide de l'approche MDA

Présenté par :

- Houichi Slimane
- Belaid boubaker

Encadrés par :

- Mr : KhiatiNadri

Année Universitaire : 2018-2019

Remerciements

Nous commençons tout d'abord par remercier le Bon Dieu de nous avoir donné le courage et la volonté pour réaliser ce travail.

Nous tenons à remercier cordialement toutes les personnes qui nous ont permis de le réaliser dans les meilleures conditions.

Un remerciement particulier à : Mr. Khiati Nadri, notre enseignant responsable, pour nous avoir encadrés tout au long de ce travail.

Merci pour ses conseils, sa disponibilité et son implication.

Ainsi qu'à tous nos enseignants de département informatique.

Nous remercions les membres du jury qui ont bien voulu examiner et évaluer ce mémoire.

Résumé

Ce sujet du stage situe dans le domaine du génie logiciel. Il vise à étudier l'approche d'ingénierie dirigée par les modèles pour le développement des applications mobiles.

Puisque la consommation des appareils mobiles augmente rapidement, les sociétés des applications mobiles ont envie de profiter cette situation. Mais il n'est pas facile de développer une application qui peut fonctionner sur la plateforme différente telque iOS, Android, Windows Phone, Black Berry etc. Il est nécessaire de développer l'application séparément pour chaque plateforme. En général, pour développer une application, on suit l'étape d'interroger des exigences, de faire l'analyse, ensuite on conçoit la conception et commence à coder. Ce processus prend beaucoup de temps et de coûts. Et il aurait un grand problème lorsque le produit ne correspond pas aux exigences.

Dans le cadre de ce mémoire, on propose une solution qui applique le principe de l'approche d'ingénierie dirigée par les modèles pour le développement des applications mobiles. Celle-ci aiderait au développeur de réduire le temps et le coût de développement grâce aux avantages des modèles. L'idée de l'approche d'ingénierie dirigée par les modèles est que le développement de l'application est guidé par les modèles. Elle est très bien adaptée avec l'approche << Cross-Platform >> afin de réaliser les applications mobiles pour des différences systèmes d'exploitation. Cela réduit le temps et des ressources pour le développement. Le code est automatiquement généré à partir des modèles qui sont définis par le développeur.

Mots-clés : Ingénierie dirigée par les modèles, Application mobile, Multi- plateforme, DSL, Modélisation.

Table des matières

Liste des figures

Liste des Tableaux

Introduction

1. Contexte général

Aujourd'hui l'explosion du marché des Smartphones et des tablettes représente un potentiel énorme pour le développement d'applications, tant pour le particulier dans sa vie quotidienne que pour le professionnel. Plus d'un million d'applications sont déjà disponibles, toutes les plateformes confondues. En 2016, les ventes de Smartphones ont augmenté de 14,4% par rapport à 2014 et elles ont atteint les 1,4 milliard d'unités vendues dans le monde, selon l'institut Gartner (*Gartner, 2016*). Une croissance divisée par deux donc : en 2014, la hausse était de près de 28%. Cette attractivité rapide pour les Smartphones provient notamment de la puissance des appareils qui ne cesse d'accroître et des nouvelles fonctionnalités qu'ils offrent. En outre, il est actuellement pratiquement possible de tout faire avec le Smartphone, grâce à un nouveau genre d'applications qui sont installées sur le Smartphone et fonctionnent indépendamment des autres. Ces applications ont plusieurs points forts, parmi eux la possibilité d'accéder aux fonctionnalités de Smartphone et du système d'exploitation installé (e.g. internet, GPS, liste des contacts, les capteurs embarqués, caméra, etc.). En conséquence, de nouveaux usages sont apparus. Par exemple en mobilité, il est maintenant possible d'éviter les retards dus aux bouchons et gagnez du temps en changeant d'itinéraire sur la base de données à jour sur le trafic routier. Il est maintenant facile de localiser sa position en temps réel et de la transmettre vers différentes destinations. Aussi, l'utilisateur peut à tout moment commander des biens à partir d'un Smartphone, de consulter l'état de la commande et de suivre où se situe le paquet commandé. En plus, ce genre d'appareils offre d'autres services non liés à la mobilité, par exemple, il est possible de consulter les informations météorologiques de base, telles que les prévisions, les conditions actuelles, etc., vérifier les comptes bancaires, suivre en direct les évènements tels que les différentes compétitions sportives (e.g. jeux olympiques, championnats nationaux et internationaux, etc.), chercher les recettes de cuisines, etc.

Pour proposer ces applications à un large panel d'utilisateurs, chaque fournisseur de système d'exploitation mobile est doté d'un magasin d'application en ligne. Ces magasins permettent aux utilisateurs de rechercher et installer les dernières versions de ces applications. Alors, grâce à ces diversités des applications mobiles que les Smartphones sont rendu très utilisables.

2. Problématique et objectif

Suite à l'hétérogénéité des systèmes d'exploitation mobiles, et celle des appareils mobiles, peut nécessiter le développement de différentes versions de la même application. Cependant, pour développer la même application sur différents systèmes d'exploitation (OS), le développeur doit apprendre à développer sur les SDKs de ces différents OS en utilisant leurs langages de programmation et leurs APIs. Par conséquent, le développement de la même application pour ces différentes plateformes devient une tâche épuisante.

Un enjeu important pour les entreprises souhaitant créer une application mobile est d'être une application de **qualité, présentée sur les différentes plateformes leaders du marché, développée avec le moindre coût et en un temps efficace**. Mais quelle stratégie adopter ? Faut-il développer une application spécifique pour chaque plateforme, ce qui représente un coût ? Est-il possible de développer une application et de la déployer sur plusieurs plateformes ?

Pour répondre à ces questions, plusieurs approches de développement mobile se présentent. L'approche native peut être le meilleur choix, car elle permet l'accès à l'ensemble des fonctionnalités de l'appareil et du système d'exploitation, étant donné qu'elle peut mieux tirer parti des fonctionnalités du système mobile ciblé et de l'appareil de l'utilisateur, elle offre des interfaces utilisateurs fluides et très réactives. En termes de performance et de vitesse, il ne devrait y avoir aucune limitation particulière avec une application native. L'un des véritables problèmes, c'est le coût de développement, car il va falloir réécrire l'application plusieurs fois pour cibler plusieurs plateformes (e.g. Android, iOS, Windows phone, etc.).

Dans ce contexte que s'inscrit notre sujet de master qui consiste à mettre en place une approche pour la modélisation et la génération des applications mobiles multiplateformes selon une approche native. En conséquence, notre objectif est de diminuer le coût et le temps de développement des applications mobiles multiplateformes, et d'offrir des applications de qualité bénéficiant de toutes les fonctionnalités natives des Smartphones (e.g. GPS, camera, capteurs embarqués, etc.), cela réduit le temps et des ressources de développement. Le code est automatiquement généré à partir des modèles qui sont définis par le développeur. Le code de l'application générée est de même avec le code réalisé manuellement et amène moins de problèmes d'insatisfaction.

3. Principe de notre approche

L'objectif de ce mémoire est de présenter une solution pour la modélisation et la génération de code natif pour les plateformes mobiles à travers une ingénierie dirigée par les modèles. En effet, dans un contexte d'accroissement exponentiel de la complexité des systèmes informatiques, la modélisation de ces systèmes est devenue un enjeu majeur de la réussite des projets : bonne prise en compte du besoin fonctionnel, réduction des délais et des coûts par la réutilisation des conceptions et des liens avec le code et, enfin, souplesse nécessaire pour l'adaptation des applications aux différentes technologies actuelles ou futures. L'ingénierie dirigée par les modèles apporte des améliorations significatives dans le développement des systèmes complexes en permettant de se concentrer sur une préoccupation plus abstraite que la programmation classique. Il s'agit d'une forme d'ingénierie générative dans laquelle tout ou Partie d'une application est engendrée à partir des modèles.

Chapitre 01 :

Développement des applications mobiles

1.1. Introduction

L'industrie du développement des applications mobiles ne cesse de croître en raison de l'utilisation intensive de ces dernières dans les appareils mobiles, la plupart d'entre elles fonctionnent sous les systèmes d'exploitation Android, iOS et Windows Phone. Cependant, le développement des applications conçues pour les plateformes mobiles exige plus des soucis tels que l'efficacité du code, l'interaction avec les périphériques, ainsi que la rapidité d'envahissement du marché.

Depuis la sortie du premier iPhone en **2007**, les appareils mobiles intelligents jouent un rôle important dans l'économie mondiale, ainsi, on parle plus souvent de l'économie numérique.

Dans le monde entier, les ventes des téléphones mobiles ont atteint près de 478 millions d'unités au cours du troisième trimestre de 2015, donc une augmentation de 3,7% par rapportable même période en 2014. Les chiffres et les tendances présentées dans l'étude suivante confirment ce constat (*Gartner, 2015*).

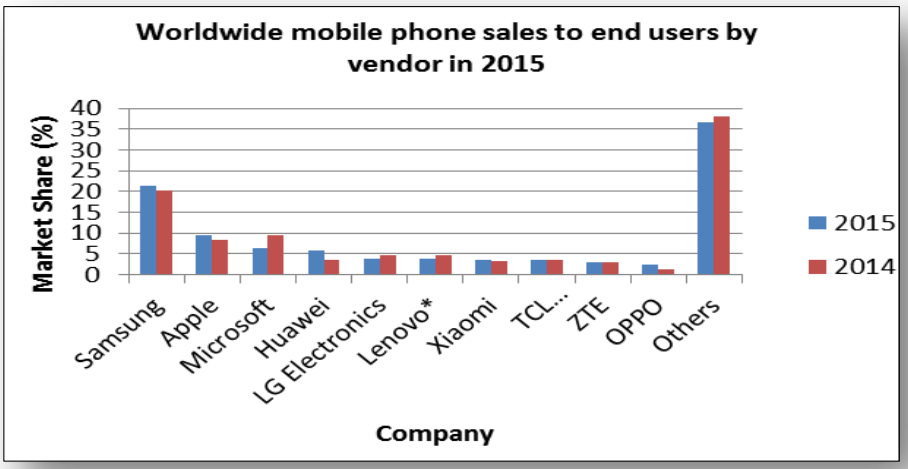

Figure1.1: Les ventes des téléphones mobiles dans le monde pour les utilisateurs finaux par fournisseur

Cette évolution est due à la croissance du marché des Smartphones, ce qui pousse les consommateurs à abandonner les téléphones classiques de plus en plus (*Gartner, 2015*). La figure "Figure 1.2" montre l'évolution des ventes de Smartphones par rapport aux ventes des appareils mobiles classiques.

Entre 2011 et 2013, la part des ventes des Smartphones a augmenté de 37%. De nos jours, environ plus de 71% des mobiles sur les marchés sont les Smartphones.

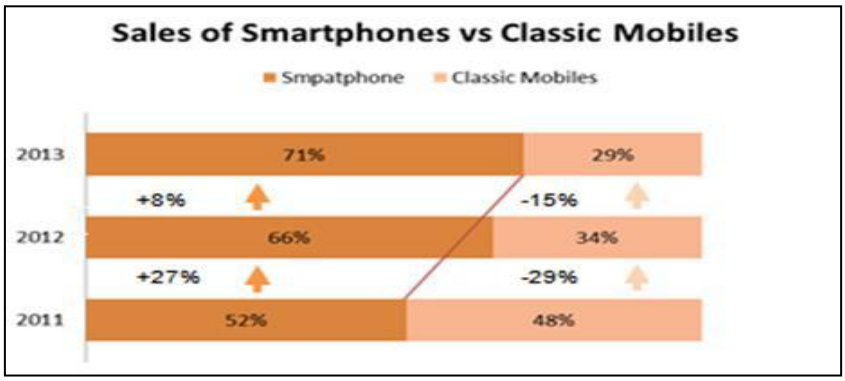

Figure 1.2:Les ventes des Smartphones vs mobiles classiques (Kerensen Consulting, 2015)

Le marché des tablettes et Smartphones est dominés par Android (*Gartner, 2015*). Le choix d'Android est justifié par sa technologie constamment novatrice, ouverte et moins chère par rapport à iOS (*voir la figure ci-dessous pour plus de détails*).

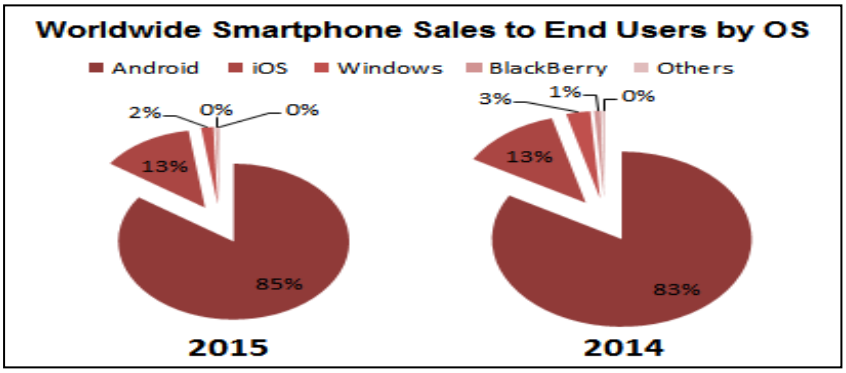

Figure 1.3: Les ventes des Smartphones dans le monde pour les utilisateurs finaux par OS (Gartner, 2015)

Un enjeu important pour les entreprises souhaitant créer une application mobile, est d'être présente sur les différentes plateformes leaders du marché. Mais quelle stratégie adopter ?Faut-il développer une application spécifique pour chaque plateforme, ce qui représente un coût ? Est-il possible de développer une application et de la déployer sur plusieurs plateformes?

Dans ce chapitre nous passons en revue les différents systèmes d'exploitation mobile leader du marché mondiale, ensuite nous présentons les défis du développement des applications mobiles, suivi d'une description des différentes approches de développement des applications mobiles, à savoir, l'approche native, l'approche web et l'approche hybride. Enfin, nous présentons notre cadre décisionnel pour les méthodes de développement mobile, suivi d'une étude de cas.

1.2. OS Mobile

Les Smartphones modernes sont riches de fonctionnalités de plus en plus sophistiqués, ce qui produit l'ouverture d'opportunités pour l'innovation logicielle. Parmi le grand nombre de plateformes pour développer un nouveau logiciel, nous examinons dans les sous-sections ci-des sous de près trois plateformes identifiées comme leaders du marché des Smartphones selon l'étude de groupe **Gratner** en 2015. Ensuite, nous comparons les plateformes selon plusieurs axes différents, telles que l'architecture logicielle, le développement d'applications, les capacités de la plateforme et les contraintes, et, enfin, le soutien des développeurs.

1.2.1 Android

Android est un système d'exploitation open source pour les terminaux mobiles, conçu parle Startup Android rachetée par Google en Août 2005. Google a publié Android en novembre2007, dans le cadre d'une alliance (Open Handset Alliance), dans le but d'être une scène open source pour le développement des logiciels sur la plateforme mobile. Android est basé sur le noyau Linux et facilite aux développeurs d'écrire du code en Java en utilisant des bibliothèques développé par Google.

La plateforme Android ne fournit pas seulement le système d'exploitation mobile, lui- même, y compris l'environnement de développement, mais fournit également une machine virtuelle sur mesure, pour exécuter les applications tout en agissant en tant que middleware entre le code et le système d'exploitation (***Ehringer, 2010***). Pour le développement d'applications, Android facilite l'utilisation de bibliothèques graphiques 2D et 3D, dispose d'un moteur SQL embarqué pour le stockage des données et des fonctionnalités réseau avancées telles que la 3G, 4G, WLAN, etc. L'API est en constante évolution et la version actuelle (Nougat 7.0) (***Wee, 2017***) est une énorme augmentation par rapport au nombre de fonctionnalités disponibles à partir de la version 1.0. Depuis qu'Android est un mobile open source, la communauté l'accueillie pour collaborer au envolement de l'environnement de programmation, système d'exploitation et l'API. Cependant, les outils de développement pour Android comprennent Eclipse et Android Studio.

Android est conçue pour des appareils mobiles au sens large. Nullement restreinte aux téléphones, elle couvre d'autres possibilités d'utilisation comme les tablettes, les ordinateurs sport ables, les bornes interactives, les baladeurs, etc.

La plateforme Android est composée de différentes couches :

- un noyau Linux qui lui confère notamment des caractéristiques multitâches ;

- une couche d'abstraction matérielle (HAL) qui fournit des interfaces standards qui exposent les fonctionnalités matérielles du périphérique ;
- Android Runtime (ART) permettant d'exécuter plusieurs machines virtuelles sur des périphériques à faible mémoire en exécutant des fichiers DEX. Chaque application s'exécute dans son propre processus et avec sa propre instance de ART ;
- des bibliothèques graphiques, multimédias ;
- un Framework applicatif proposant des fonctionnalités de gestion de fenêtres, de téléphonie, de gestion de contenu, etc. ;
- des applications dont un navigateur web, une gestion des contacts, un calendrier, etc.

Les composants majeurs de la plateforme Android sont résumés sur le schéma suivant (*traduit de la documentation Google*).

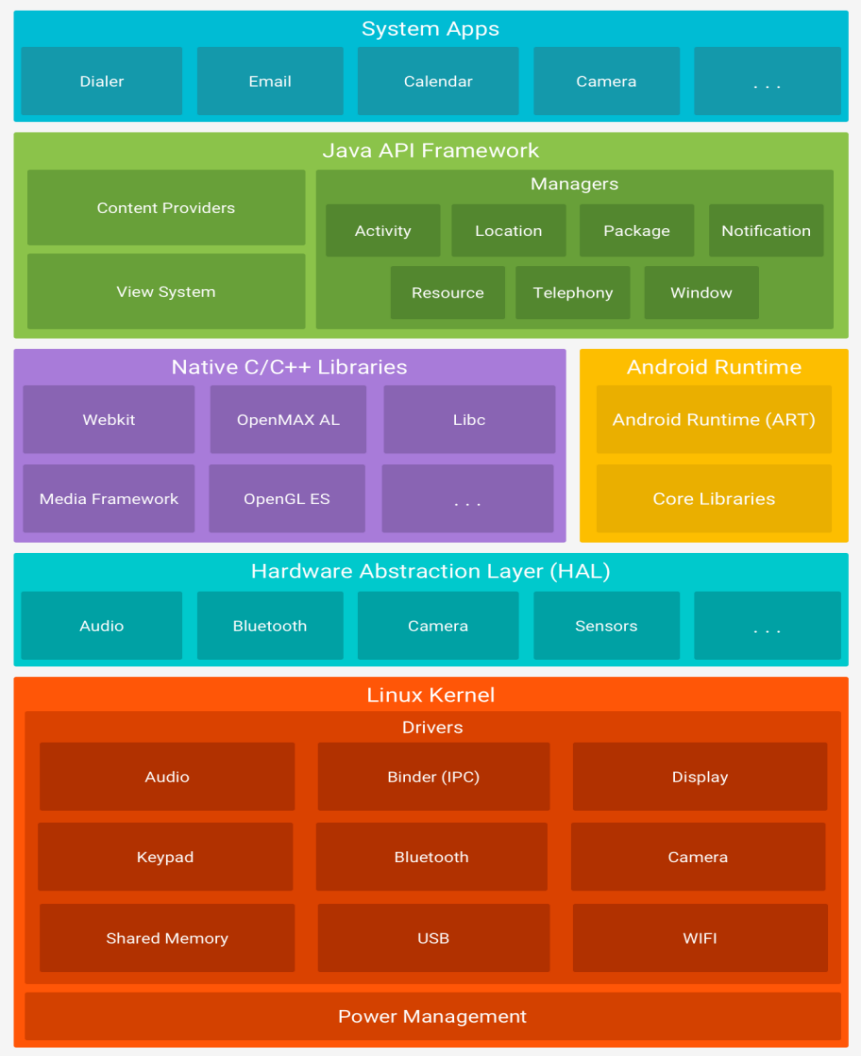

Figure1.4:Architecture Android (Android, 2017)

1.2 .2. iOS

IOS est le système d'exploitation mobile développé par Apple à l'origine pour l'iPhone, plus tard étendu à l'iPad et iPod. Il est dérivé de Mac OSx dont il partage les fondations (le Keren hybride XNU basé sur le micronoyau Mach, les services Unix et Cocoa, etc.). iOS comporte quatre couches d'abstraction, similaires à celles de Mac OS X : une couche « Core OS », une couche « Core Services », une couche « Media » et une couche « Cocoa ».

Figure 1.5:Architecture iOS (Tracy, 2012)

L'architecture du système est identique à l'architecture de Mac OS X et comprend les composants suivants (***Liu et al, 2011***):

- *CocoaTouch*: Comprend l'UIKit, qui est un cadre fondé sur Objective C et fournit un certain nombre de fonctionnalités, qui sont nécessaires pour le développement d'une application iOS comme la gestion de l'interface utilisateur ;
- *Média*: Contient les graphiques, audio et technologies vidéo orientées vers la création de la meilleure expérience multimédia disponible sur un appareil mobile ;
- *Services de base*: système de services fondamentaux, qui sont subdivisés en différents cadres et basés sur C et Objective C ;
- *Core OS*: le noyau du système d'exploitation.

1.2.3 Windows Phone

Auparavant, le système d'exploitation mobile créé par Microsoft a été appelé Windows Mobile. Après les modifications introduites par Apple (iOS) et Google (Android) en 2007,

Microsoft a décidé de prendre une nouvelle direction et a créé Windows Phone. Semblable à d'autres alternatives, telles que Android et iOS. Windows Phone est un système d'exploitation pour Smartphones. Il est généralement utilisé sur les appareils à écran tactile, et il offre des fonctionnalités comme l'accès aux réseaux, l'accès aux capteurs et l'intégration de la caméra,

Windows est accompagné historiquement d'une plateforme de développement riche et variée. Le développeur d'applications dispose d'un grand choix d'outils de qualité, quel que soit l'archétype de ses programmes : client riche, web, mobile ou service.

Les applications Windows Store ne dérogent pas à la règle et peuvent être implémentées en se basant sur des technologies éprouvées telles que Visual Studio, .NET, C++, XAML ou encore HTML5 et JavaScript. Le Windows Runtime est la plateforme sur laquelle s'appuient les applications Windows Store. Cette plateforme fournit des API conçues pour faciliter le développement d'applications natives performantes, mobiles, contextuelles, fluides et sécurisées. Un soin tout particulier a également été apporté à l'ouverture de ces API : il est possible de les utiliser depuis de nombreuses technologies telles que le code natif C++, le code managé .NET ou encore JavaScript. Dans les coulisses, le Windows Runtime et ses API sont implémentés en C++ en se basant sur l'évolution d'un composant fondamental et historique de Windows : le Component Object Model ou COM.

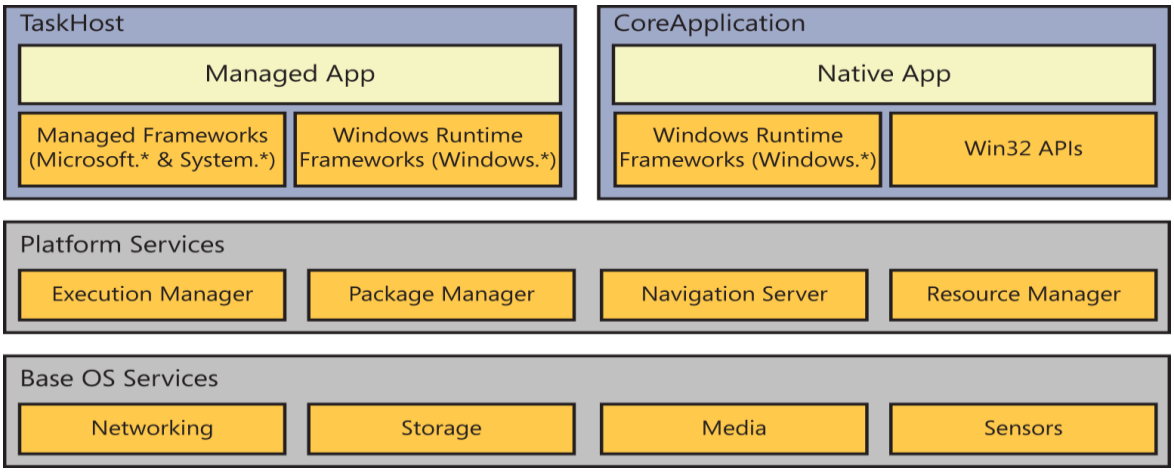

Figure1.6:Architecture Windows Phone

1.2.4 Comparaison des différentes plateformes

Au niveau logiciel, les différentes OS (Android, iOS, Windows Phone, etc.) se disputent le marché de la mobilité. A chaque OS correspond une plateforme de téléchargement d'applications, des environnements de développement et des langages de programmation

permettant le développement et la distribution des applications. Le tableau "Tableau 1.1"Présenté ci-dessous décrit les principaux OS mobile présents sur le marché.

Cette hétérogénéité des outils de développement et des langages, rend difficile le développement d'applications mobile multiplateformes. Par conséquent, elle pousse les développeurs à faire un choix sur la plateforme, tout en assurant la plus grande distribution possible.

OS	Android	iOS	Windows phone
Editer par	Google	Apple	Microsoft
Environnement de développement	Eclipse. Android Studio	XCode	Visual Studio
Langages de programmation	JAVA	Objective +C, Swift	C#,VB net
Interface graphique	XML	Cocoa Touch	XAML
Fichier executable	apk	.app	.xap
Applications sur	Google Play	Apple-iTunes	Windows Store
Machine virtuelle	Dalvik VM	Non	CLR
Développement sur	Multiplatform	Mac OS X	Windows

Tableau 1 : Descriptif des principaux OS mobile présents sur le marché (*Perchat et al, 2013*).

1.3. Les défis du développement des applications mobiles

Le développement des applications mobiles a beaucoup de restrictions et des défis tels que:

(1) Les ressources limitées des appareils mobiles même si elles sont plus puissantes Qu'auparavant (***Yung-Wei et al, 2011***):

- puissance de calcul limitée;
- espace de stockage limité ;
- connectivité affectée par le mouvement.

(2) Hétérogénéité des systèmes d'exploitation mobiles:

- différents environnements de développement pour les applications mobiles (e.g. une application développée par iOS SDK ne peut fonctionner que sur des appareils iOS et une application Android ne peut fonctionner que sur des appareils Android) (***Baixing et al, 2012***) ;
- pour développer la même application sur différents systèmes d'exploitation (OS), le développeur doit apprendre à développer sur les SDKs de ces différents OS en utilisant leurs langages de programmation et leurs API (***Biap et al, 2010***).

(3) Hétérogénéité des appareils peut nécessiter différentes versions de la même application (*Baixing et al, 2012*):

- différentes capacités de calcul et de configurations des périphériques matérielles doivent être considérées lors de l'élaboration d'une application ;
- différentes tailles d'écrans des appareils: l'écran de la plupart des téléphones mobiles est inférieur à quatre pouces, les tailles d'écran de la tablette sont de 7 pouces ou 10 pouces, et l'écran du téléviseur intelligent peut être plus grand que 60 pouces ;
- différentes méthodes de saisie: écran tactile, clavier, télécommande TV et TV à la télévision intelligente.

(4) L'expérience utilisateur: les développeurs doivent définir une simple et conviviale Interface utilisateur pour les applications développées (*Perchat et al, 2013*).

(5) Maintenance d'application: mises à jour fréquentes de la plateforme mobile peut affecter certaines applications qui les rend inutilisables dans la nouvelle version; par conséquent, la maintenance et les mises à jour de ces applications sont nécessaires (*Perchat et al, 2013*). Aussi la gestion des versions est difficile, car les utilisateurs ne peuvent pas mettre à jour l'application vers la nouvelle version lorsqu'il est libéré (*Baixing et al, 2012*).

L'entretien ou les mises à jour de l'application développée pour différentes plateformes Signifie que le développeur répète les mêmes mises à jour dans toutes les versions de différentes plateformes (*Baixing et al, 2012*).

(6) Développement multiplateforme : Le développement d'une même application mobile pour différentes plateformes, signifier la répétition de même travail à plusieurs reprises, parce que chaque plateforme fournit aux développeurs des différents langages de programmation et différents outils de développement.

Bien que le développement des applications mobiles a beaucoup de restrictions et des défis, le principal défi qui est abordé dans le présent document est de savoir comment développer l'application mobile une seule fois et de l'exécuter sur différentes plateformes mobiles, pour économiser le temps, le coût et les efforts des développeurs. Ce chapitre présentera les différentes approches et solutions pour résoudre ce problème et se termine avec les axes ouverts de recherche.

1.4. Approches de développement mobiles

Plusieurs études sur les approches pour le développement des applications mobile smulti plateformes sont produites (*Raj et al, 2012*), (*Smutny, 2012*), (*Palmier et al, 2012*)(*Serrano et*

al, 2013), (*El-Kassas et al, 2015*). Par conséquent, nous pouvons classer ces approches en trois catégories illustrées dans la figure suivante :

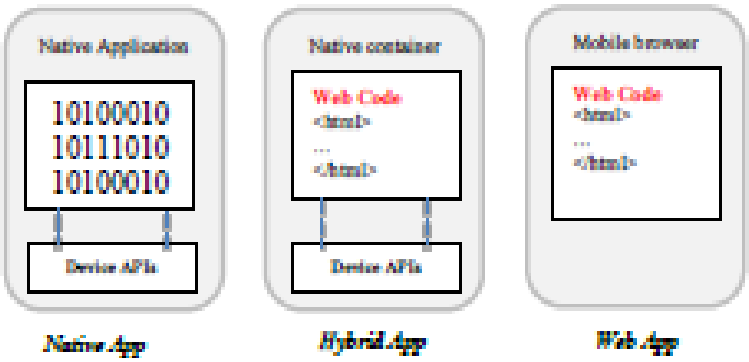

Figure 1.7:Méthode de développement des applications mobiles

Les sous sections suivantes décrivent ces trois approches de développement mobile.

1.4.1. Approche native

Avec l'approche de développement natif pur, nous pouvons créer des applications qui sont écrites pour une plateforme spécifique et exécutées sur cette plateforme uniquement. Ce type d'applications permet d'obtenir de meilleurs résultats et exploiter pleinement toutes les fonctions natives de la plateforme, telles que l'accès à l'appareil photo, les capteurs embarquées à la liste de contacts, l'activation de symboles ou l'interaction avec d'autres applications. Pour prendre en charge des plateformes telles qu'Android, iOS, Java™ ME et Windows Phone, nous devons développer des applications distinctes avec des langages de programmation différents, comme Swift pour iOS, Java pour Android, ou C# pour Windows Phone (*Raj et al, 2012*), (*Smutny, 2012*), (*Xanthopoulos et al, 2013*).

A la différence des applications Web mobiles et de bureau, les applications natives sont distribuées par le biais d'un magasin d'applications en ligne.

L'architecture d'une application native est présentée dans la figure ci-dessous:

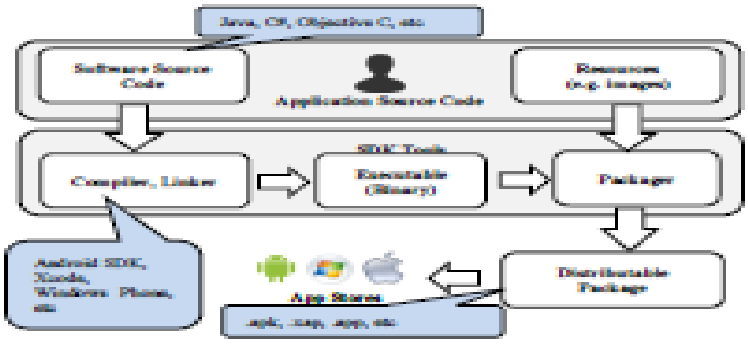

Figure1.8:Architecture d'une application native.

Les avantages et les inconvénients de cette approche sont présentés dans le tableau ci-dessous

Tableau 2 : Avantages et inconvénients de l'approche native

Avantages	Inconvénients
✓ Multitude d'API pour accéder à toutes les fonctionnalités des périphériques mobiles comme la caméra, des capteurs, l'accès au réseau, GPS, stockage de fichiers, base de données, SMS et e-mail ; ✓ Meilleures performances par rapport aux applications web et aux applications hybride ; ✓ Apparence native, interface fluide et conviviale.	✓ - Applications natives sont plus difficiles à développer et nécessitent un haut niveau d'expérience (*Xanthopoulos et al, 2013*) ; ✓ - Ils doivent être développés séparément pour chaque plateforme, ce qui augmente les temps de développement, le coût et les efforts (*Sambasivan et al, 2011*) ; ✓ - Restrictions et les coûts associés au développement et le déploiement de certaines plateformes (e.g. la licence de développement Apple et l'approbation d'Apple pour distribuer les applications à la boutique iTunes Apps) (*Smutny, 2012*).

1.4.2. Approche web

Avec l'approche de développement web, l'application s'exécute dans le navigateur du terminal mobile et utilise les technologies web standard telles que HTML5, CSS3 et JavaScript. Les applications web mobiles n'ont pas accès aux fonctions de la plateforme, carrelles reposent uniquement sur le navigateur et sur les normes web associées. Les applications web mobiles ne sont pas distribuées via un magasin d'applications. Elles sont accessibles via un lien sur un site Web ou un signet dans le navigateur du terminal mobile de l'utilisateur.

L'architecture d'une application web est présentée dans la figure ci-dessous :

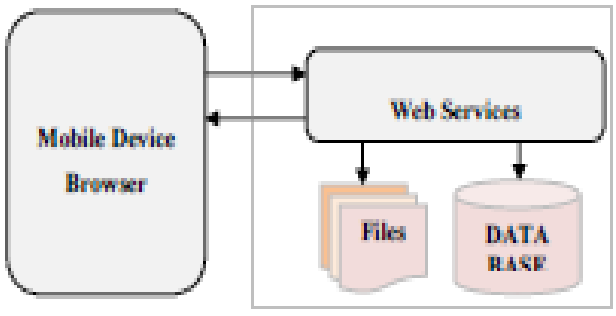

Figure 1.9:L'architecture logique d'une application web mobile (Raj et al, 2012)

Les avantages et les inconvénients de cette approche sont présentés dans le tableau ci-dessous :

Tableau 3 : Avantages et inconvénients de l'approche web

Avantages	Inconvénients
✓ Facile à apprendre et à développer en utilisant les technologies web ; ✓ Tout le traitement est effectué sur le serveur et seule l'interface utilisateur est envoyée au mobile ; ✓ Le maintien de l'application est simple parce que les mises à jour de l'application et les données sont effectuées sur le serveur ; ✓ La même application est développée une fois et peut fonctionner sur différentes plateformes en utilisant les navigateurs web mobiles.	✓ Les applications web ne sont pas disponibles dans les boutiques en ligne ; ✓ Une connexion Internet est nécessaire pour faire fonctionner l'application ; ✓ Les applications web ne peuvent pas accéder au logiciel de l'appareil mobile et aux matériels tels que la caméra, et les capteurs, GPS, etc. (*Raj et al, 2012*) ; ✓ Une application web pourrait souffrir du mauvais fonctionnement en raison de la connexion et les délais réseau (*Raj et al, 2012*) ; - Le développeur d'applications à moins de contrôle sur la manière dont les différents navigateurs interprètent le contenu.

1.4.3. Approche hybride

L'approche de développement hybride, permet de créer des applications qui utilisent toutou partie des approches de développement web et native. L'application hybride s'exécute dans un conteneur natif et utilise le moteur de navigateur pour afficher l'interface d'applications, basée sur HTML et JavaScript. Avec le conteneur natif, l'application peut accéder à des fonctionnalités de terminal auxquelles les applications web n'ont pas d'accès, telles que l'accéléromètre, la caméra et le stockage en local sur un smart phone. A l'instar des applications natives, les applications hybrides sont distribuées via le magasin d'applications de la plateforme.

L'architecture d'une application hybride est présentée dans la figure ci-dessous :

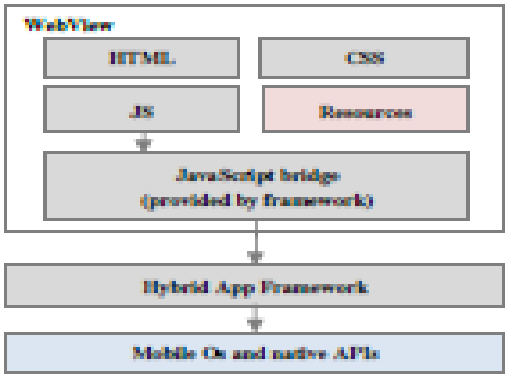

Figure1.10: L'architecture logique d'une application hybride typique

Les avantages et les inconvénients de cette approche sont présentés dans le tableau ci-dessous:

Tableau 4 : Avantages et inconvénients de l'approche hybride

Avantages	Inconvénients
✓ Une application hybride est distribuable par le biais de la boutique des applications ; ✓ L'interface utilisateur peut être réutilisée dans différentes plateformes ; ✓ L'application peut accéder aux fonctionnalités natives de l'appareil mobile.	✓ Moins performante que l'application native ; ✓ L'interface utilisateur n'aura pas l'apparence de l'application native. ✓ Pour obtenir une apparence native, le style propre à la plateforme pourrait être nécessaire.

1.4.4. Comparaison des approches de développement d'applications mobile

Chacune de ces approches de développement présentent des avantages et des inconvénients. Ainsi, nous devons sélectionner l'approche de développement appropriée en fonction des besoins spécifiques de chaque solution mobile individuelle. Ce choix dépend considérablement des caractéristiques de l'application mobile et de ses exigences fonctionnelles. La première étape d'un projet de développement d'application mobile consiste à mettre en correspondance les impératifs et les approches de développement possibles. Le tableau "Tableau1.5" présenté ci-dessous souligne les principaux aspects des trois approches de développement et aide à déterminer laquelle est la mieux adaptée pour la mise en place de application mobile.

Selon cette étude, nous avons remarqué que l'application native est mieux en termes de performances par rapport aux autres types d'applications mobiles (web et hybrides).L'application native est développée en utilisant une plateforme spécifique, API compilée pour fonctionner sur la plateforme et non pas avec un langage interprété, comme JavaScript. Mais le problème est que ces applications natives sont plus coûteuses à mettre en œuvre, limitées à une plateforme mobile particulière, et elles ont besoin d'une collection de connaissances et des langages pour les réaliser.

La figure "Figure 1.11 " présentée ci-dessous illustre la tendance de l'approche native par rapport aux facteurs coût et temps des approches de développement multiplateformes.

Aspect	Développement Web	Développement hybride	Développement natif
Facile a apprendre	Facile	Moyen	Difficile
Performances des applications	Lentes	Moyennes	Rapides
Connaissances requises du terminal	Aucun	Certaines	Nombreuses
Les utilisateurs potentiels	Maximum y compris Les Smartphones. Tablettes et autres Téléphone …….	Large	Limité a une Plateforme mobile particulière
Cycle de vie du développement (génération /test/déploiement)	Court	Moyen	Long
Portabilité des applications vers d'autres Plateformes	Elevés	Elevés	Aucun
Prise en charge des Fonctionnalités du terminal natif	Certaines	La plupart	Toutes
Distribution avec mécanisme Intégrés	non	oui	Oui
Possibilité d'écrire des Extensions pour les Fonctionnalités du terminal	non	oui	Oui
sécurité	Dépend de la Sécurité du navigateur	n'est pas bonne	Elevée
Langage de développement	Web uniquement	Native et web ou Web uniquement	Native uniquement
Compétences/outils nécessaires Pour les applications multiplateformes	HTML CSS JavaScript	HTML , CSS JavaScript mobile Développent Framework (like Phone Gap, iconic)	Objective-C ,java , C, C++, C C, C++, C# VB.net

Tableau 5 : Comparaison des approches de développement des applications mobiles

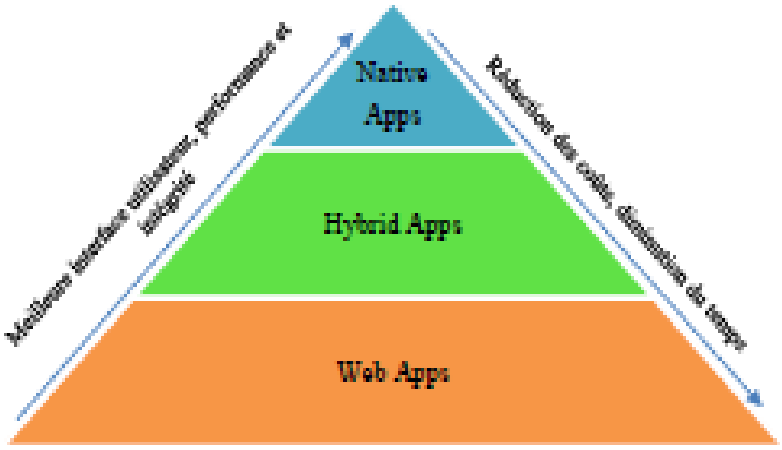

Figure 1.11:Approche native vs développement multiplateforme

1 .5. Conclusion

Au terme de conclusion, nous avons présenté dans ce chapitre une brève description des plateformes mobiles leader du marché mondial, les défis de développement mobile et les différentes approches de développement mobile.

Chapitre 02 :

L’approche MDA

2.1. Architecture Dirigée par les Modèles (*Model Driven Architecture – MDA*)

L'approche MDA est une démarche proposée par l'OMG (***OMG, 1989***) depuis 2001. Ils 'agit d'une vision particulière du Développement Dirigé par les Modèles (*MDD: Model Driven Développent*) (***Hailpern et al, 2006).*** Ce dernier, qui contrairement au MDA, n'applique pas les standards de l'OMG, est un paradigme flexible pour la définition des processus de développement qui considère les modèles ainsi que les transformations comme des artefacts principaux de ce processus. Selon (***Mellor et al, 2003***) il s'agit tout simplement de la notion selon laquelle il est possible de construire le modèle d'un système afin de pouvoir par la suite le transformer automatiquement ou semi automatiquement en une chose réelle. Les artefacts du MDD sont utilisés pour spécifier, simuler, vérifier, tester et générer le système final.

À la différence du MDD, le MDE va au-delà des activités de développement et englobe d'autres tâches basées sur un processus d'ingénierie logicielle (e.g. l'évolution basée sur un modèle) (***Cabot, 2015***).

L'idée fondamentale du MDA, en utilisant les standards de l'OMG, est que les fonctionnalités du système à développer sont définies initialement dans un Modèle indépendant de l'informatisation (*CIM : Computation Independent Model*) qui est utilisé pour la création d'un modèle indépendant de toute plateforme (*PIM : Platform Independent Model*). Ce dernier, épaulé par un modèle de description de la plateforme d'exécution (*PDM : Platform Description Model*), permet la génération (semi-) automatique par transformation d'un ou d'un ensemble de modèles spécifiques aux plateformes (*PSM : Platform Specific Model*).

Figure 12.1 : Model Driven Architecture de l'OMG.

2.2. Les différents modèles de MDA

2.2.1 Le CIM (Computation Independent Model) :

- *CIM*: modèle indépendant de tout système informatique qui utilise un vocabulaire familier au maître d'ouvrage. Il permet d'avoir une vision de ce qui est attendu du système, sans rentrer dans le détail de sa structure ni de son implémentation. L'indépendance technique de ce modèle lui permet de garder tout son intérêt au cours du temps. Il est modifié uniquement si les connaissances ou les besoins métier changent ;

2.2.2 Le PIM (Platform Independent Model) :

- *PIM* : modèle qui décrit la logique métier ainsi que le fonctionnement des entités et des services. C'est un modèle qui ne contient pas d'information sur les technologies qui seront utilisées pour déployer l'application ;

2.2.3 Le PDM (Platform Description Model) :

- *PDM* : modèle qui permet de décrire l'architecture logicielle de la plateforme d'exécution. Il contient les informations pour la transformation des modèles vers une plateforme spécifique. Les outils BluAge Forward (***Bluage, 2015***) et AndroMDA (***Bohlen, 2007***) définissent ce modèle sous forme de cartouche de génération remplaçable en fonction de la plateforme d'exécution. Cette cartouche, appelée BSPs par BluAge (*BLUAGE Shared Plug-ins*), est disponible pour les Frame Works les plus utilisés comme Struts, Spring, Hibernante, .Net, Java, etc. ;

2.2.4 Le PSM (Platform Specific Model):

- *PSM* : modèle dépendant de la plateforme technique spécifiée par l'architecte. Il sert essentiellement de base à la génération de code exécutable vers la ou les plateformes techniques cibles. Il existe plusieurs niveaux de PSM. Le premier est issu de la transformation d'un PIM tandis que les autres sont obtenus par transformation successives jusqu'au code dans un langage spécifique (e.g. JSF2, EJB3, Struts, etc.).

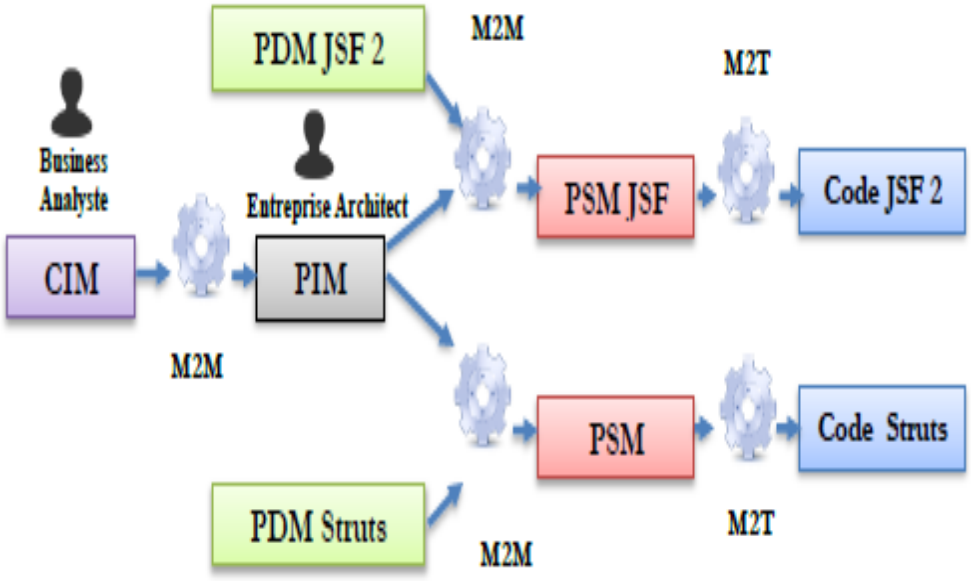

Figure 2.2:Exemple d'utilisation des modèles dans l'ingénierie vers l'avant (Forward engineering)

Dans la figure ci-dessus nous illustrons les différents types de modèles du MDA, utilisés pour la réalisation d'une application fictive. Pour aboutir à une application en JSF2 (***Burns etal, 2010***) par exemple, tout d'abord, le modèle CIM est transformé en modèle PIM. En suite, le modèle PSM est obtenu par une transformation qui prend en entrée le précédent modèle PIM ainsi que le modèle PDM qui décrit l'architecture logicielle de JSF2. Enfin le code de l'application est généré à partir du modèle PSM par une transformation M2T. Aujourd'hui plusieurs outils respectent cette approche MDA. Parmi les plus récents, nous pouvons citer:

- *AndroMDA* est une plateforme de génération de code extensible qui transforme des modèles UML en composants qui peuvent être déployés sur une plateforme donnée (e.g. JEE, Spring, .NET, etc.) (***AndroMDA, 2012***) ;
- *Acceleo* est un générateur de code open 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 (***Acceleo, 2014***).

2.3. Méta-modélisation et Multi-modélisation

La méta-modélisation est une activité consistant à définir le méta-modèle d'un langage de modélisation (***Jézéquel et al, 2012***). Elle vise donc à modéliser un langage qui joue le rôle du système à modéliser. Les notions de système, modèle et méta-modèle ainsi que les relation sentre elles sont présentées dans la Figure ci-dessous.

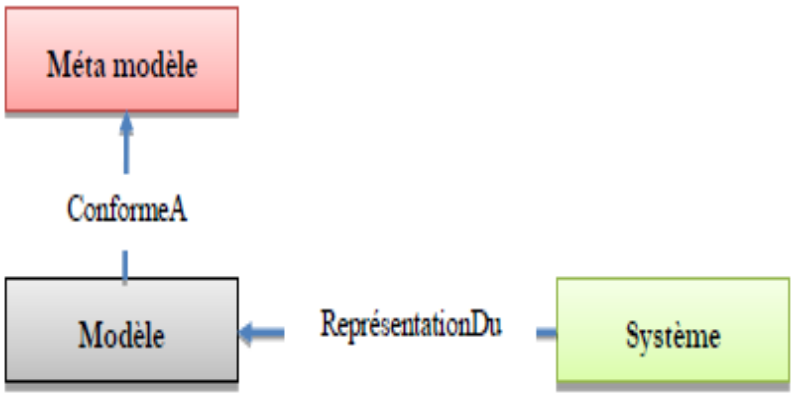

Figure 2.3:Relations entre système, modèle et méta-modèle (Bézivin, 2004)

Un méta-modèle est un modèle qui définit le langage de modélisation d'un modèle (***OMG, 2002***). Dans un méta modèle, on définit les concepts ainsi que les relations entre les concepts permettant d'exprimer des modèles (***Bézivin, 2001***). Un modèle est une construction possible, ou instance, d'un méta modèle. Comme évoqué dans la section précédente, un modèle est défini comme une représentation d'un système, construit pour un objectif précis. De cette définition découle la relation entre modèle et système nommée «représentation Du». Cependant, une fois le modèle construit, peut-on dire qu'il est une représentation correcte du système? Pour répondre à cette question, le modèle doit pouvoir satisfaire le principe de substituabilité. Ce principe est défini parMinsky (***Minsky, 1965***) de la manière suivante : «*un modèle doit être suffisant et nécessaire pour permettre de répondre à certaines questions à la place du système qu'il est censé représenter, exactement de la même façon que le système aurait répondu lui-même*».

Étant donné que les modèles peuvent ne pas représenter l'ensemble du système et ainsi faillir à satisfaire le principe de substituabilité, plusieurs modèles peuvent être utilisés conjointement pour représenter le même système. Ceci est appelé «multi-modélisation». Selon (***Bézivin, 2009***) la multi-modélisation consiste à gérer un système complexe par l'intermédiaire de plusieurs modèles, chacun englobant un certain type de connaissances. Ces modèles nécessitent à un certain moment du processus de développement d'être composés pour obtenir un modèle plus global qui représente le système. Plusieurs approches s'intéressent à la multi-modélisation, dont les approches par points de vue, par aspects et par modèles. Comme dit ci-dessus, un méta-modèle est un modèle qui définit le langage d'expression d'un modèle, c'est-à-dire le langage de modélisation (***Jézéquel et al, 2012***). En d'autres termes un méta-modèle est une représentation qui vise à définir les éléments utilisés dans un modèle. Le modèle est relié à son méta-modèle par une relation nommée «conforme A». Ils 'agit de la même relation qui relie un langage de programmation à sa grammaire. De la citation «tout est modèle» de J. Bézivin (***Bézivin, 2005***), on déduit que le méta modèle est lui aussi un modèle et est donc conforme à un autre méta-modèle (appelé méta méta-modèle).

2.4. L'architecture MDA :

Le modèle MDA est décomposé en 4 niveaux :

- *Le niveau 0* correspond au monde réel. Il contient les informations réelles correspondant au système à modéliser. Il est conforme au modèle du niveau 1 ;
- *Le niveau 1* (ou modèle) regroupe les modèles. Il décrit les informations de niveau 0. Les modèles UML, PIM et PSM appartiennent à ce niveau. Le modèle 1 est conforme au méta-modèle du niveau 2 ;
- *Le niveau 2* (ou méta-modèle), représente les langages de définition des modèles. Le méta-modèle UML qui permet de définir des modèles UML, et le méta-modèle SPEM qui permet de définir des modèles de procédé, appartiennent tous les deux à ce niveau. Il faut noter aussi que les profils UML, mécanismes d'extension du méta-modèle UML, font partie de ce niveau. Un méta-modèle est conforme à un méta-méta-modèle ;
- *Le niveau 3* (ou méta-méta-modèle) permet de décrire la structure des méta-modèles et d'étendre ou de modifier les méta-modèles existants. Le méta-méta-modèle se décrit lui-même (réflexivité).

La différence entre les langages de programmation classiques et les méta-modèles réside dans le fait que, pour les premiers, on manipule principalement une syntaxe concrète, tandis que pour les seconds, on manipule exclusivement la syntaxe abstraite. L'approche IDM consiste donc à construire un langage en définissant sa syntaxe abstraite au moyen d'un méta modèle pour ensuite créer des outils de manipulation, d'édition ou de transformation pour ce langage. Cette approche est particulièrement bien adaptée aux DSLs dont la syntaxe abstraite concise permet de décrire des modèles de systèmes.

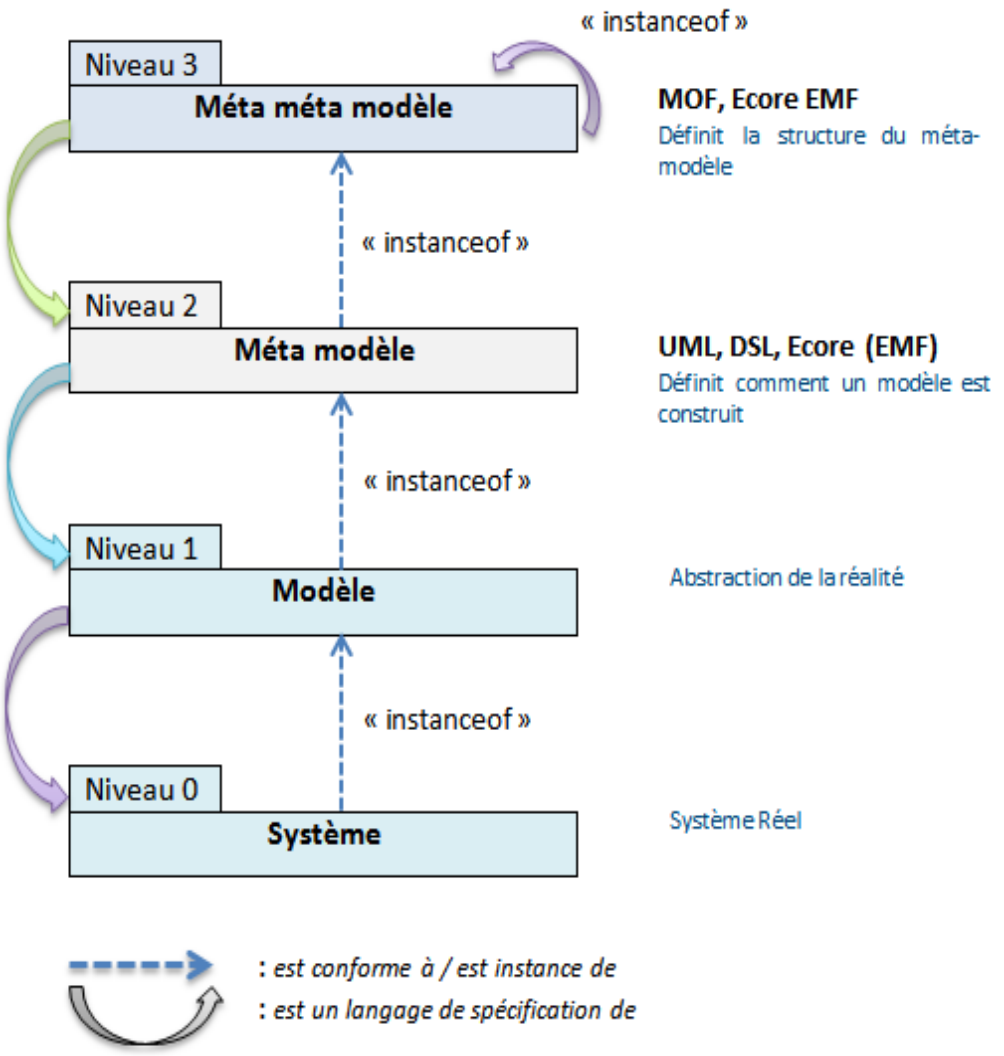

Figure 2.413:Figure 2.4:Architecture à 4 niveaux d'abstraction

2.5. Langages de méta modélisation

Nous avons vu que l'architecture à quatre niveaux de MDA permettait de faire la différence entre les entités à modéliser (*niveau 0*), les modèles (*niveau 1*), les méta-modèles (*niveau 2*) et les méta-méta-modèles (*niveau 3*). Les langages de méta-modélisation se situent au niveau M3, et permettent de spécifier des méta-modèles au niveau 2. Dans l'approche MDA, le langage MOF incarne le socle architectural. Dans l'environnement Eclipse, le langage Ecore joue le même rôle que MOF, il permet la spécification de méta-modèles.

- *Le langage MOF* (Meta Object Facility) a été adopté par l'OMG en 1997. La spécification de MOF définit un langage abstrait et un Framework pour la spécification, la construction et la gestion des méta-modèles génériques. De plus, MOF définit une plateforme pour l'implémentation de modèles décrits par les méta-modèles. Les méta modèles MOF 2.0 sont définis sous forme de classes. Les modèles conformes aux méta-modèles sont considérés comme des instances de ces classes. La figure ci-dessous Présente un extrait de ce que pourrait être un diagramme de classe représentant MOF2.0.

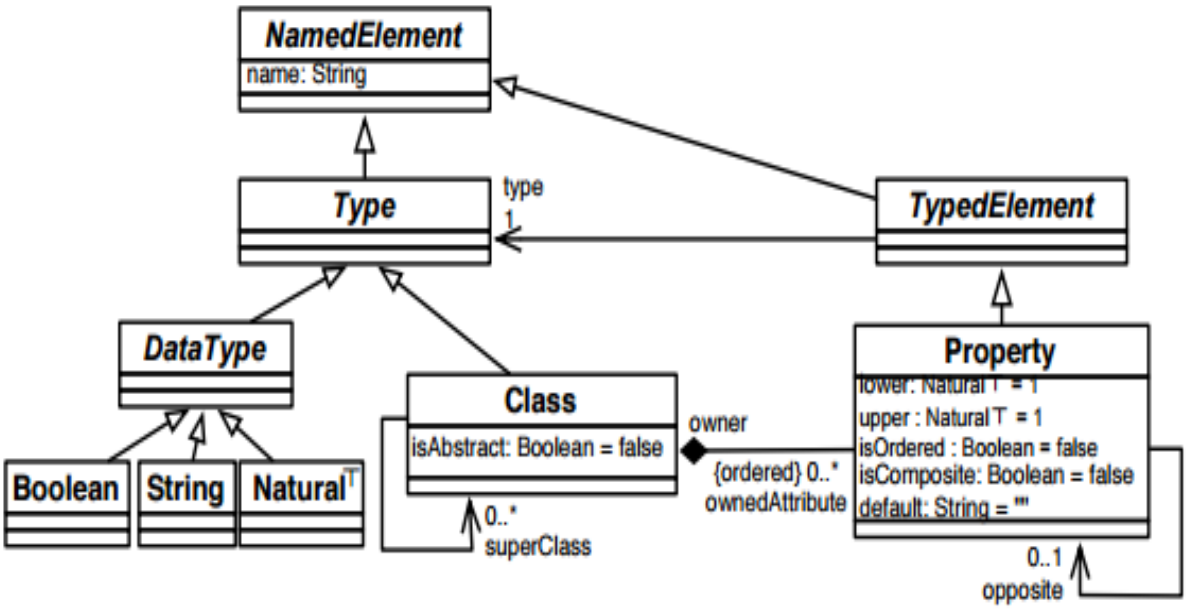

Figure 2.514: Représentation de MOF 2.0 sous forme de diagramme de classes

- *Le langage Ecore* : langage de méta-modélisation qui fait partie d'EMF (*Eclipse Modeling Framework*) et qui est le résultat des efforts du projet ETP (*Eclipse Tools Project*). EMF est un Framework de modélisation de code pour supporter la création d'outils et d'application dirigées par les modèles. La particularité des méta-modèles Ecore est qu'ils ne contiennent pas de méta-associations entre leurs méta-classes. Pour exprimer une relation entre deux méta-classes, il faut utiliser des méta-attributs et les typer par des méta-classes. EMF impose cette contrainte pour faciliter la génération des interfaces Java. Le concept d'association n'existant pas en Java, il faudrait en effet une transcription particulière. Ainsi, Le modèle ECORE permet de définir les éléments d'un modèle selon les méta-classes de la figure ci-dessous:
  - EPackage: représente un package qui peut comporter des classes ;
  - Eclass: représente une classe, qui peut se composer de plusieurs attributs et références ;
  - EAttribute: représente un attribut d'une classe. Il a un nom et un type ;
  - EReference: représente une fin d'association ou un rôle d'une association entre deux classes ;
  - EDataType: représente les types d'attributs utilisés dans les modèles, par exemple, String, int, float, etc. ;
  - EOperation: représente une méthode dans Eclass.

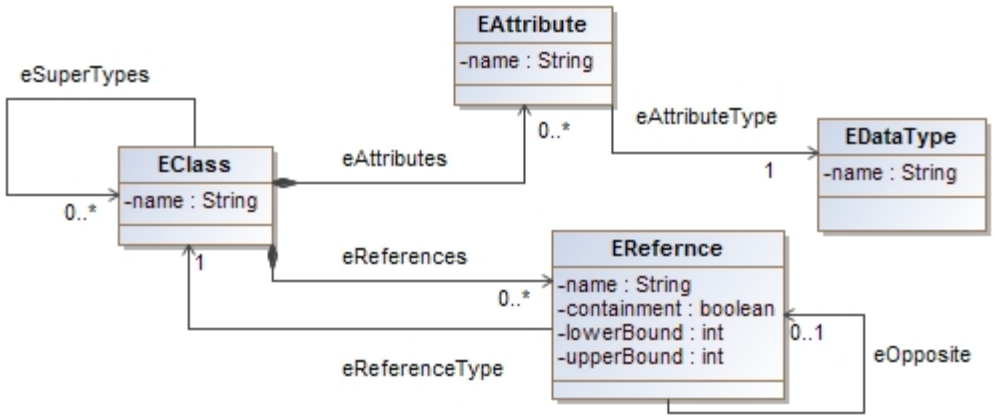

Figure 2.6: Extrait du Méta-Modèle Ecore

Tous les éléments des modèles créés en EMF doivent être placés dans un objet de type EPackage comme élément racine du modèle.

2.6. Langages de modélisation

On distingue deux types de langages de modélisation : les langages de modélisation généralistes (*General Purpose Modeling Languages – GPMLs*) et les langages de modélisation dédiés (*Domain Specific Modeling Languages – DSML*). Les DSMLs, ainsi que les DSLs (*Domain Specific Languages*), sont des langages dédiés chacun à un domaine métier spécifique, et n'ont pas vocation à résoudre un problème en dehors de ce domaine. Le principal intérêt des DSMLs est qu'ils permettent aux experts d'un domaine métier de pouvoir penser en termes proches de leur domaine lorsqu'ils spécifient leurs systèmes (***Bernoussi, 2008***). Contrairement à un DSML, un GPML est un langage de modélisation généraliste qui n'est pas restreint à un domaine particulier. Java et UML sont respectivement des exemples de GPL et GPML. Certes, les GPMLs sont souvent standardisés et sont supportés par de nombreux outils riches en termes de fonctionnalités, mais ils sont plus difficiles à apprendre et à utiliser. Pour UML, selon (***Vallecillo, 2010***), les utilisateurs ont de sérieux problèmes pour la compréhension de sa structure complexe et ont tendance à utiliser le peu qu'ils savent qui est estimé à 20%. Ce constat est consolidé par le résultat de l'étude de Hutchinson et al. (***Hutchinson et al, 2014***). La Figure ci-dessous, décrit le nombre de diagrammes régulièrement utilisés par des experts industriels selon différents cas d'étude.

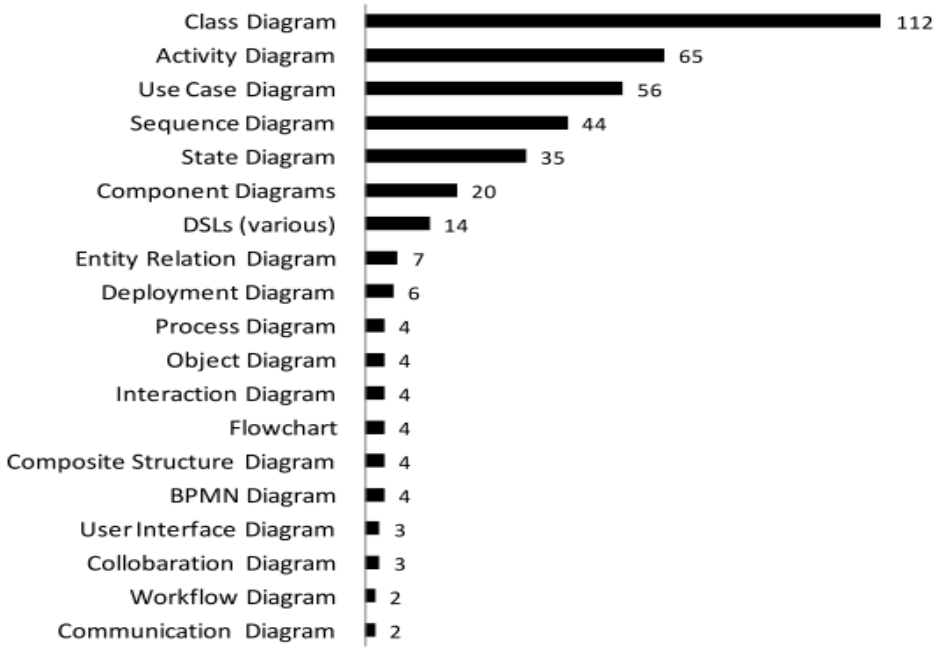

Figure 2.7:Liste des diagrammes utilisés régulièrement en UML (Hutchinson et al, 2014)

Un DSML est différent d'un GPML. Un GPML est un standard monolithique obtenu par consensus alors qu'un DSML offre aux utilisateurs des concepts propres à leurs métiers, ce qui permet de réduire le temps d'apprentissage du langage de modélisation. Un DSML permet aussi d'améliorer la productivité étant donné qu'il est plus facile de manipuler directement les concepts du domaine métier.

Un DSML est composé de trois éléments. Premièrement, la syntaxe abstraite qui décrit les concepts du langage et les relations entre eux. Deuxièmement, la (les) syntaxe(s) concrète(s) qui décri (ven) t le formalisme, textuel ou graphique, pour la manipulation du langage. Troisièmement, la sémantique qui attribue un sens à chaque concept du langage.

Martin Fowler (***Fowler, 2010***) a identifié trois types de DSMLs. Le DSML interne, le DSML externe et l'atelier de langage (*language worknbench*). Le DSML interne utilise un sous-ensemble du GPML mais dans un style personnalisé afin de gérer une partie de l'ensemble du système. G-Marker (***Bluage, 2015***) est un exemple de ce type. Le DSML externe possède une existence autonome. Il correspond au type le plus utilisé et auquel on fait référence par abus de langage par le terme DSML tout court. L'atelier de langage est un DSML pour la construction de DSMLs. Xtext (***Bettini, 2016***) et GMF (***Gronback, 2009***) sont des exemples de ce type.

2.7. Langage dédié (ou méta modèle)

2.7.1. L'ingénierie des modèles et le langage dédié

Dans une démarche IDM, les spécifications du logiciel sont décrites grâce à des méta modèles ou langages dédiés à un domaine (*Domain Specific Language, DSL*).

Un DSL est un langage adapté à un domaine d'application donné (*systèmes embarqués, systèmes d'information, etc.).* Pour le domaine considéré, le DSL offre un gain substantiel en expressivité et en simplicité d'utilisation, comparé à un langage généraliste (Java, C#, etc.) (***Mernik et al, 2005***). Le gain est comparable au gain obtenu lorsqu'on utilise une API de programmation, plutôt que des fonctionnalités de plus bas niveau.

Dans cette partie nous passons en revue les étapes de modélisation des connaissances d'un domaine par le biais de langages dédiés (DSLs). La réalisation de notre approche nécessitant de définir des méta-modèles basés sur langage dédié pour les plateformes mobiles. On part du principe que ce langage facilite l'expression des tâches de développement et de maintenance, à condition qu'ils soient suffisamment expressifs et compréhensibles pour les experts qui sont amenés à effectuer ces tâches.

2.7.2Étapes de développement d'un langage dédié

Le développement d'un DSL est une tâche difficile qui nécessite la connaissance du domaine en question, ainsi qu'une expertise en ingénierie des langages. Peu de gens possédant cette double compétence, le recours à l'implémentation de DSLs a toujours été une solution difficile à mettre en œuvre. Dans la littérature, il existe peu de travaux sur les méthodologies générales de construction de DSLs, la plupart des travaux se cantonnent à un domaine d'application donné, et sont fortement influencés par les supports outillés qu'ils proposent. Néanmoins, dans (***Mernik et al, 2005***), une étude est réalisée sur les différentes phases de construction d'un DSL, nous en synthétisons les éléments intéressants pour la mise en œuvre de notre approche.

2.7.2.1. Analyse

La phase d'analyse consiste à identifier le (ou les) domaine(s) du problème, et recueillir les connaissances spécifiques à ces domaines. Les entrées d'un tel processus sont toutes les sources (implicites et explicites) de connaissance sur le domaine, par exemple : documents techniques, connaissance fournie par les experts du domaine, code source existant, documents de veille, etc. En sortie, la phase d'analyse fournit une terminologie et une sémantique sous une forme plus ou moins abstraite.

2.7.2.2. Conception

Les approches de conception d'un DSL peuvent être caractérisées suivant deux axes : le positionnement du DSL par rapport aux langages de modélisation existants, et la nature plus ou moins formelle du DSL. Il existe plusieurs façons de concevoir un DSL. La première est de se baser sur un langage de modélisation existant : soit par restriction du langage (*application d'un profil UML*) soit par extension (*extension d'un langage minimal comme Ecore*). Dans les deux cas, les notations du DSL s'appuient sur les notations prédéfinies du langage existant, et bénéficient du fait que ces langages soient largement adoptés. La deuxième façon de concevoir un DSL consiste à définir un nouveau langage à partir de rien (*from scratch*). En pratique, cette dernière façon est extrêmement difficile à mettre en œuvre, puisqu'elle nécessite de définir toute la syntaxe et la sémantique du langage, et va à l'encontre des courants de standardisation des langages de modélisation. Une fois le DSL positionné par rapport aux langages de modélisation existants, la phase de conception consiste à le spécifier. Pour les DSLs informels, la spécification est exprimée avec un langage plus ou moins proche du langage naturel, généralement avec des représentations graphiques.

Les DSLs faits à partir d'UML ou EMF tombent dans cette catégorie. Dans le cas d'un DSL formel, on trouve des formalismes comme les expressions régulières et les grammaires pour la spécification de la syntaxe, et les systèmes de réécritureetles machines d'état pour la spécification de la sémantique.

2.7.2.3. Implémentation

Les techniques d'implémentation d'un DSL peuvent être positionnées par rapport aux techniques d'implémentation classiques, i.e. avec un langage de programmation généraliste. Les techniques d'extension de langage classiques, comme les sub-routines et les macros, peuvent être utilisées pour implémenter un DSL. L'extension par sub-routines (e.g. API Java, etc.) consiste à étendre le langage de base avec des structures de données et des opérateurs plus abstraits. L'avantage de cette approche est que l'utilisateur du DSL n'a pas besoin d'être expert dans le langage de base. Néanmoins, le risque existe de confondre la sémantique des opérateurs spécifiques au domaine avec les opérateurs de base. L'extension par macro (e.g. les tem plates C++) est indépendante du langage de base, la cohérence syntaxique n'est cependant pas garantie, elle est en effet vérifiée lors de la compilation ou l'interprétation.

Par ailleurs, d'autres techniques reposent sur la construction d'un DSL autour d'un outil et notations spécifiques. Les DSLs à base de XML par exemple font partie de cette catégorie. Pour un DSL basé sur XML, la grammaire est exprimées sous forme de DTD ou schéma XML, les opérations sont des transformations XSLT, et sont aussi codées en XML. Ainsi XML et les outils associés peuvent être utilisés pour construire un compilateur de langage de programmation. Une variante particulière de cette catégorie est la technologie EMF où une API Java est générée et compilée à partir du DSL exprimé au format XMI.

2.7.2.4. Validation

En théorie, la validation d'un DSL permet de s'assurer que toute implémentation à partir de sa spécification permet de créer des modèles valides dans le domaine. En pratique, la validation des DSL se fait la plupart du temps expérimentalement, soit par la simulation, par exemple dans le projet TOPCASED (***Farail, 2006***), soit par les scénarios d'utilisation connus a priori. La validation d'un DSL est un sujet peu traité dans la littérature. Mis à part quelques résultats quantitatifs, notamment dans (***Herndon et al, 1988***), la plupart des approches adoptent des démarches qualitatives pour mesurer les bénéfices d'un DSL. Ainsi, il existe une multitude de critères qualitatifs sur lesquels on peut se baser pour juger un DSL. Nous en synthétisons

quelques critères qui nous semblent pertinents pour la suite de notre étude, et nous classons ces critères en quatre regroupements:

- *Expressivité du langage* : l'aptitude à décrire une large famille de modèles, l'aptitude à assigner une sémantique aux modèles, etc. ;
- *Efficacité du langage* : simplicité d'utilisation, facilité de maintenance, extensibilité du langage, etc. ;
- *Support d'environnement de conception automatisé*: la possibilité de valider le DSL avec des outils, l'aptitude à transformer des modèles et à générer du code, la possibilité de composer des modèles, etc. ;
- *Méthode de développement* : il s'agit de valider les différentes étapes d'acquisition des connaissances, de conception, et d'implémentation du DSL.

2.8. Outils pour la définition des DSLs autonomes

2.8.1. Xtext

Xtext est le pilier pour la création de DSL textuel externe, c'est une solution de Eclipse Modeling Project pour la mise en place de DSLs textuels et de leurs éditeurs associés. Xtext est la solution envisagée pour permettre la formalisation de la logique mathématique des modèles exécutables ainsi que pour la saisie des expressions logiques associées à la définition d'une séquence conditionnelle.

Dans le cas de Xtext, le méta-modèle de la structure de données est inféré à partir de la description de la syntaxe du DSL. Il est donc plus simple de faire évoluer un langage, puisque les répercussions sur la structure de données sont immédiates.

La figure ci-dessous fournit une vue d'ensemble de l'outil Xtext. La forme textuelle du DSL constitue le point de départ. La grammaire du DSL est le point d'entrée de l'outil. Xtext produit, en sortie, un analyseur syntaxique, un éditeur et un méta-modèle pour le DSL.

Xtext permet de considérer la forme textuelle du DSL comme un modèle conforme à la grammaire d'entrée, donc au méta-modèle généré.

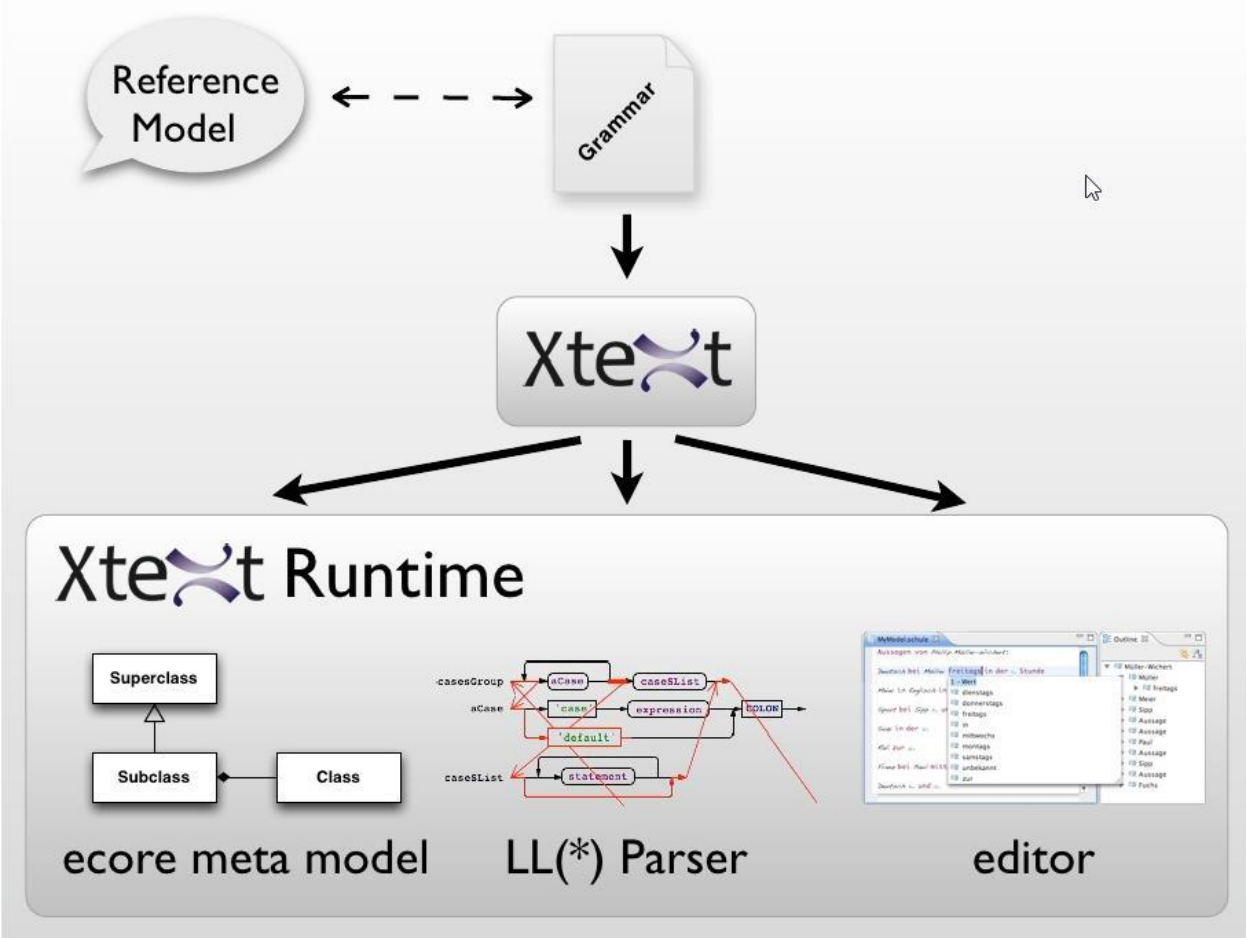

Figure 2.8:Vue d'ensemble de l'outil Xtext

2.8.2. Spoofax

Spoofaxest aussi basé sur Eclipse, mais ne s'appuie pas sur EMF. C'est un système développé par TU Delftet bien plus innovant en termes de fonctionnalités supportées, e.g. il dispose d'un langage déclaratif pour le binding et le scoping, et va plus loin que Xtext sur le support de la modularité. D'un autre côté, il est beaucoup moins répandu.

2.8.3. JetbrainsMPS

JetBrains MPSest différent de Xtext et Spoofax. Il n'utilise pas de texte brut pour l'édition et l'analyse. Par contre, il utilise une approche projection elle, où chaque action d'édition change l'arbre syntaxique abstrait (*the abstract syntax tree, ou AST, en anglais*) et ce que l'on voit et avec quoi on interagit n'est qu'une projection. Cela permet d'utiliser une gamme plus importante de notations, y compris des tableaux, des barres de fraction, et des formes graphiques. Il facilite aussi l'extension de langages et la combinaison des extensions développées indépendamment dans un même programme. Il n'est pas aussi largement utilisé que Xtext.

2.9. Synthèse

Les langages dédiés sont réputés pour leur expressivité et leur fort niveau d'abstraction. Ils sont utilisés dans différents domaines pour permettre aux experts de domaine de concevoir des solutions en utilisant des concepts et des notations qui leur sont familiers.

Le développement des DSL est une tâche difficile qui demande une connaissance du domaine et des compétences en développement des langages. À la différence des langages généralistes qui bénéficient d'un ensemble substantiel de théories et d'expériences, les langages dédiés manquent toujours de méthodes et de processus efficaces pour soutenir leur conception (***Selic, 2007***).

Dans cette section, une synthèse de la littérature a été réalisée afin d'identifier les activités impliquées dans un processus de développement de DSL. Trois activités ont été considérées comme principales : analyse, conception et implémentation. La Figure ci-dessous récapitule ce processus en spécifiant les intrants et les extrants de chacune des activités.

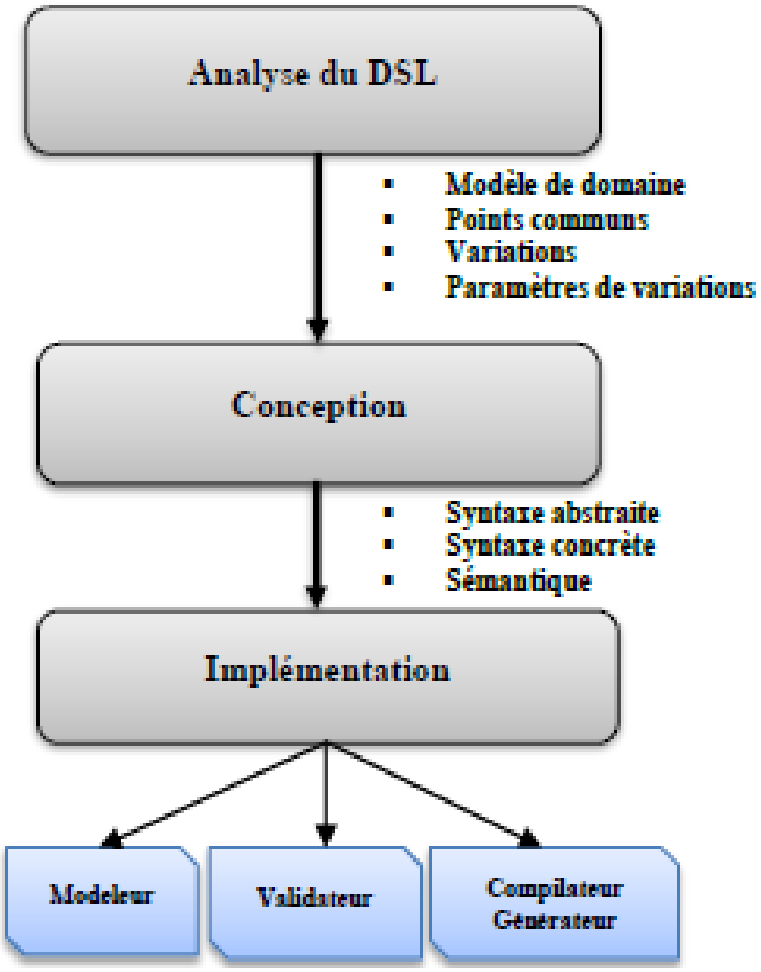

Figure 2.9:Processus de développement des DSL

2.10Les transformations MDA :

Les transformations possibles entre ces différents types de modèles sont représentées sur la figure 2.10 (Redouane Lbath Diaw, 2008)

Transformations PIM -> PIM et PSM -> PSM. Les transformations de type PIM vers PIM ou PSM vers PSM visent à enrichir, filtrer ou spécialiser le modèle. Il s'agit de transformations de modèle à modèle. Elles sont automatisables (ou partiellement automatisable) dans certains cas comme la traduction vers un autre langage, mais les transformations de type raffinement ne le sont généralement pas ; Transformation PIM -> PSM.

La transformation de PIM vers PSM permet de spécialiser le PIM en fonction de la plate-forme cible choisie. Elle n'est effectuée qu'une fois, le PIM suffisamment raffiné. Cette transformation de modèle à modèle est réalisée en s'appuyant sur les informations fournies par le PDM ;

Transformation PSM -> code. La transformation de PSM vers l'implémentation (le code) est une transformation de type modèle à texte. Le code est parfois assimilé par certains à un PSM exécutable. Dans la pratique, il n'est généralement pas possible d'obtenir la totalité du code à partir du modèle et il est alors nécessaire de le compléter manuellement ;

Transformations inverses PSM ->PIM et code ->PSM. Ces transformations sont des opérations de rétro-ingénierie (reverse engineering). Ce type de transformation pose de nombreuses difficultés mais il est essentiel pour la réutilisation de l'existant dans le cadre de l'approche MDA.

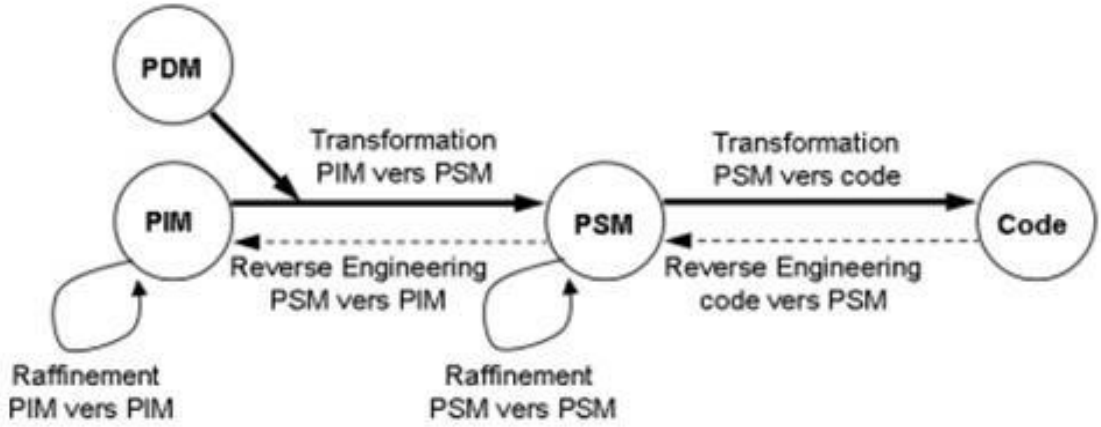

Figure 2.10: Les modèles et transformations dans l'approche MDA

2 .11 Objectifs de l'approche MDA :

L'OMG a défini l'approche MDA pour répondre aux problèmes liés à l'évolution continue des technologies. MDA est à l'origine de l'ingénierie dirigée par les modèles. Tout ce qui constitue l'approche MDA, la modélisation, les transformations, les technologies de mises en œuvre… a été pensé afin de permettre à MDA de remplir les objectifs suivants :

2 .11.1. La pérennité des savoir-faire :

La technologie évolue plus vite que les métiers de l'entreprise. Il semble alors plus intéressant de capitaliser les savoir-faire de l'entreprise indépendamment des préoccupations techniques. L'approche MDA vise principalement à rendre les spécifications métier des entreprises pérennes, indépendamment des technologies de mise en œuvre. Cette pérennité est possible grâce aux modèles, qui sont par nature des entités pérennes, et à l'utilisation d'UML qui est un standard stable et largement répandu.

2.11.2. Les gains de productivité :

Les modèles sont au cœur du processus MDA, ils facilitent la communication entre experts du domaine et développeurs, mais pas seulement. Dans MDA les modèles deviennent un outil de production grâce à l'automatisation des transformations de modèles. Le modèle devient un élément qui véhicule une information utile, précise et nécessaire, permettant l'obtention du code source de l'application par génération automatique du code. L'automatisation des transformations que permettent les modèles dans le processus de développement MDA, entraîne un gain de productivité.

2.11.3. La prise en compte des plateformes d'exécution :

À partir du PIM, MDA permet d'intégrer les spécifications techniques d'une plateforme d'exécution dans les transformations PIM vers PSM, et d'obtenir ainsi un PSM spécifique à la plateforme d'exécution, à partir duquel le code source de l'application est généré. Grâce à ce procédé, une application tire pleinement parti des fonctionnalités de la plateforme d'exécution cible. Le développement d'applications multiplateformes, ou encore la migration logicielle, sont facilités puisqu'il suffit à partir du PIM, d'effectuer les transformations PIM vers PSM en y intégrant à chaque fois, le modèle de la plateforme concernée. Obtenant ainsi pour chaque plateforme, son PSM à partir duquel le code source sera généré.

2.12. Conclusion

En raison du grand nombre et de la diversité des technologies mobiles et web basées sur les composants, le développement d’une même application pour ces différentes plateformes devienne une tâche épuisante. Vu que chaque plateforme utilise divers outils et langages de programmation, cette hétérogénéité des outils de développement et des langages de programmation rend difficile le développement des applications multiplateformes. Ainsi, il exige les développeurs de faire un choix sur la plateforme, tout en assurant la plus large diffusion possible.

Les avantages potentiels de l’approche MDA proviennent de la réduction des coûts en ayant qu'un seul code à écrire et à maintenir, ainsi que la réduction des délais de développement ; étant en mesure d'écrire un code et de cibler plusieurs dispositifs et plateformes à la fois. Notre travail s'inscrit dans cette catégorie de recherche qui vise à automatiser la génération de code pour les applications multiplateformes à partir d'un modèle textuel conforme à notre DSL. Cette approche repose sur quatre étapes principales:

- Analyse et modélisation de l’interface graphique avec un modèle textuel conforme à notre DSL ;
- Validation du modèle par Xtext ;
- Transformation PIM vers le PSM à l’aide de Xtext ;
- Génération de code en projetant dans les Tem plates des plateformes cibles.

Chapitre 03 :

Solution proposée

3.1. Solution proposée

Maintenant on va parler plus précisément sur la solution proposée. On peut expliquer étape par étape comme suite :

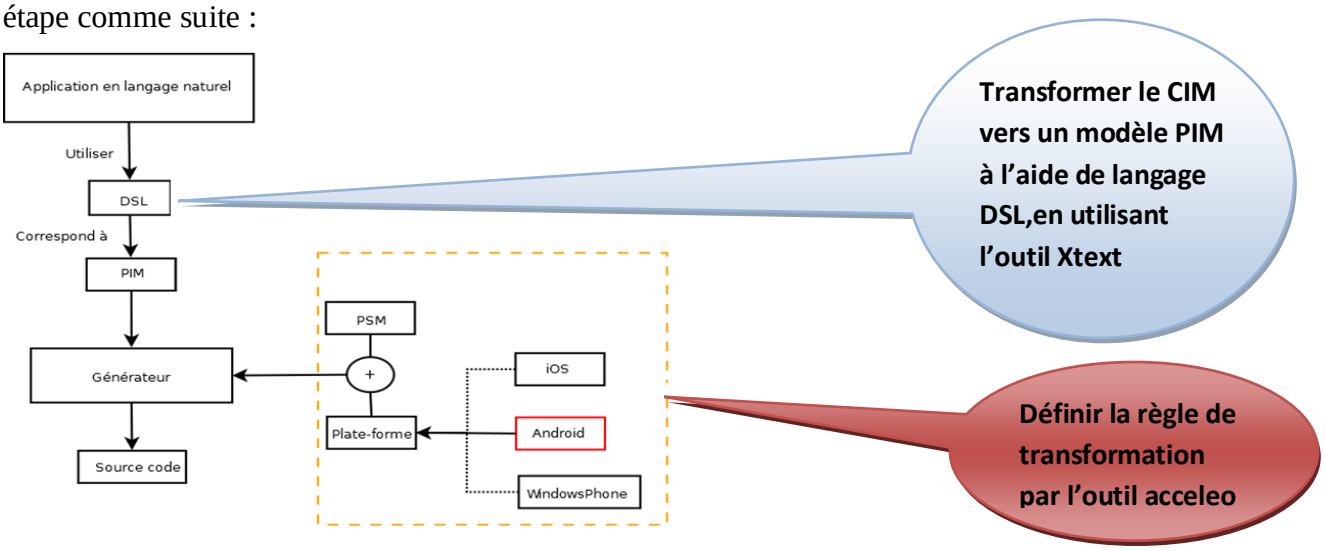

Figure 3.1:La vue l'ensemble de solution

- Au niveau de M3, le méta-méta modèle utilisé est suivi le standard MOF.
- On suppose que l'application de langage naturel est le CIM dans la figure 3.1.
- On doit transformer le CIM vers le PIM par l'aide d'un langage DSL.
- On implémente la technologie du projet Eclipse, distribué dans Eclipse Modeling Framework : Xtext(The Eclipse Foundation. Xtext., 2014) pour définir le langage DSL. Ce Framework offre toutes les fonctionnalités pour créer son propre langage et la définition de règles de génération de transformer de grammaire dans leurs programmes à un langage d'usage général comme Java, Objective C, C + +, etc.
- On implémente le Xtext pour définir le langage de méta-modèle (Mobile App- Model) et puis on utilise ce langage pour spécifier le modèle correspond au PIM.
- Ensuite, on développe le générateur de code pour les plateformes spécifiées en espérant d'obtenir le code de l'application mobile de la plateforme cible. Pour le fait, on applique la théorique de la transformation le modèle PIM vers les modèles PSM pour chaque plateforme mobile.
- Afin de définir la règle de la transformation, on utilise l'Acceleo (The Eclipse Foundation. Acceleo.,2014) qui est une implémentation de l'Object Management Group (OMG) sur le norme de MOF, MOF Models to Text (MOFM2T) (Inc Object Management Group, 2014)..

Les travaux de définir un DSL pour le PIM et de réaliser le générateur de code sont les travaux au niveau de méta-modèle (M2) dans le principe de l'approche de d'ingénierie dirigée par les modèles. Et les modèles obtenus (PIM, PSM) sont dans le niveau de modèle (M1) qui est transformé pour l'objectif d'obtenir le code exécutable.

3.1.1 Méta-modélisation

Un modèle est une représentation abstraite de la réalité. Il est utilisé pour représentes schématiquement une partie des concepts d'un programme, afin de permettre une meilleure compréhension de la façon dont l'architecture fonctionne. Pour comprendre ce qui est contenu dans un modèle ainsi que les informations qu'il doit représenter, il faut d'abord s'entendre sur la définition de son contenu. C'est le travail qui est visé lors de l'utilisation du terme << méta-modélisation >>

Normalement, il existe plusieurs techniques de méta-modélisation comme qu'on a parlé dans la partie de techniques disponibles de l'approche MDA dans le chapitre 2, Les méta-modèles dédiés peuvent donc être créés pour répondre aux besoins spécifiques. Ils sont appelés DSL (Domain Specific Language). Comme DSL sont plus largement utilisés, le développement de nouvelles méta-modèles devrait devenir mieux comprise et plus fréquente. Pour développer un méta-modèle, les étapes suivantes sont suivies :

- Définir les concepts qui devraient être modélisés et la meilleure façon de le faire.
- Représenter ces concepts sous forme de MOF ou EMF diagrammes.

Notre objectif principal est de réaliser le générateur de code pour les applications mobiles en appliquant l'approche d'ingénierie dirigée par les modèles donc l'application mobile sera le domaine considéré pour le DSL.

3.1.1.1 Conception de la méta-modélisation

Une fois défini notre domaine de langage, on doit décider sur le choix d'une syntaxe de notre langage qui doit représenter toutes les composantes de notre domaine. Ce type particulier de langage est appelé<< Domain Specific Language(DSL)>> : un langage spécifique à un domaine particulier. Enfin, on va créer une grammaire de transformation qui est capable de transformer un fichier écrit selon cette syntaxe textuelle et de construire une instance appropriée de notre modèle de domaine. La figure3.2représentel'idée du processus de la transformation le CIM vers le PIM.

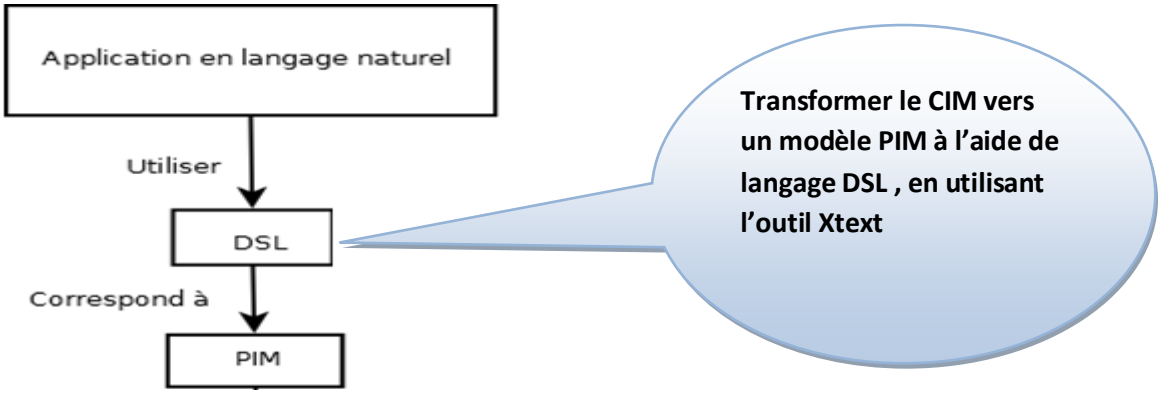

Figure 3.2:L'étape de réalisation de PIM

Pour le développement du DSL, on utilise << Eclipse Xtext >> Le Xtext (The Eclipse Foundation. Xtext., 2014) fournit l'outil pour créer et utiliser le DSL textuelles dans un environnement de <<Model Driven Développement (MDD)>> Le langage de domaine est décrit dans un méta-modèle. Et la syntaxe du langage textuelle est décrite dans un fichier de grammaire .Xtext. Le Xtext peut également déduire de notre méta-modèle de langage, de la grammaire de langage.

3.1.2. Générateur de code

Avant d'obtenir le code complet, on a d'abord le PSM. Pour générer le modèle PSM, on applique le principe de transformation de Object Management Group (OMG)qui s'appelle _ Model to Text Transformation Language (M2T) _. M2T se concentre sur la génération le codage textuelle à partir de modèle. On a choisi l'Acceleo comme l'outil de développement.

3.1.2.1 Conception pour le développement du générateur de code

Pour la conception de réalisation du générateur de code, on doit définir d'abord la règle de la transformation pour le générateur de code. Afin de faire cela, on utilise<<Acceleo>> Acceleo est nativement basée sur EMF, et est donc directement compatible avec les outils créés dans ce cadre comme Xtext. Les avantages de Acceleo sont une bonne intégration avec Eclipse et supporte les fonctionnalités du développement comme debug, tracking et tracing. Un autre avantage de Acceleo est qu'il a la fonctionnalité qui permet de la manipulation confortable de code généré, la construction du générateur natif et il s'applique au MOF, le standard de M2T (Model to Text transformation) spécifié par l'OMG.

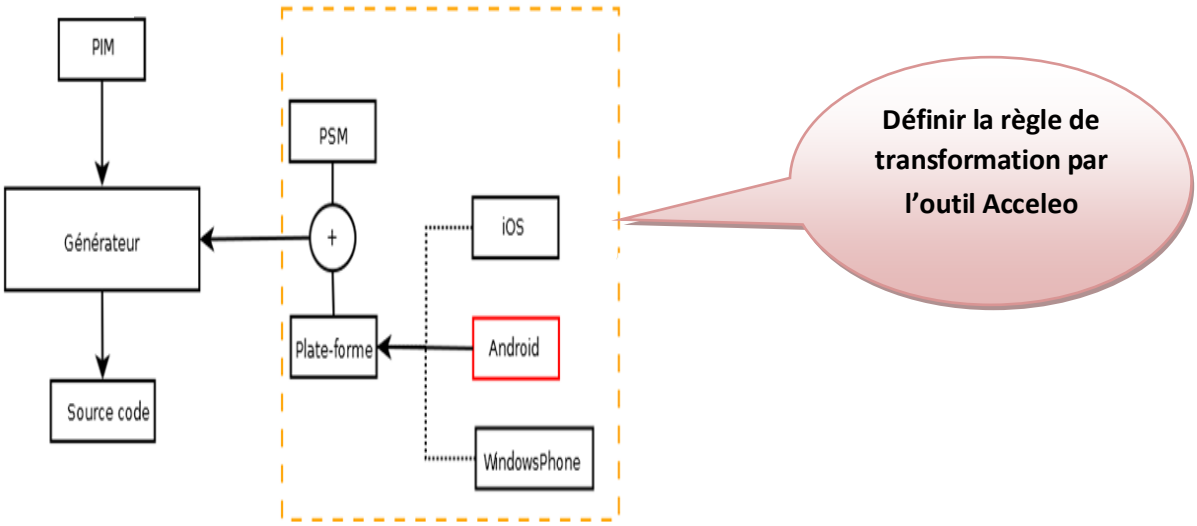

Figure 3.3: L'étape de réalisation de générateur

On définit la règle de la transformation pour le modèle PSM spécifié à la plateforme Android. Cela est le générateur de code. Ensuite le développeur fait passer le modèle PIM travers le générateur de code pour obtenir le code de plateforme cible.

On choisit d'implémenter le générateur de la plateforme Android. D'abord il faut avoir la connaissance sur cette plateforme. Celle-ci sera représentée dans la partie suivante.

3.1.2.2 Plateforme cible (Android)

La plateforme Android (Google Inc.'s. Android developers, 2014) est un logicielle libre de Google Inc., qui inclut un système d'exploitation, middleware et aussi des applications pour une utilisation sur les appareils mobiles, y compris les Smartphones.

Pour écrire le code pour les applications d'Android, nous devons travailler avec des fichiers et de nombreux sous-répertoire dans le cadre du projet de répertoire principal. Chaque répertoire a été alloué à la fonction correspondante. Il est nécessaire de savoir que comment cela pourrait bien fonctionner.

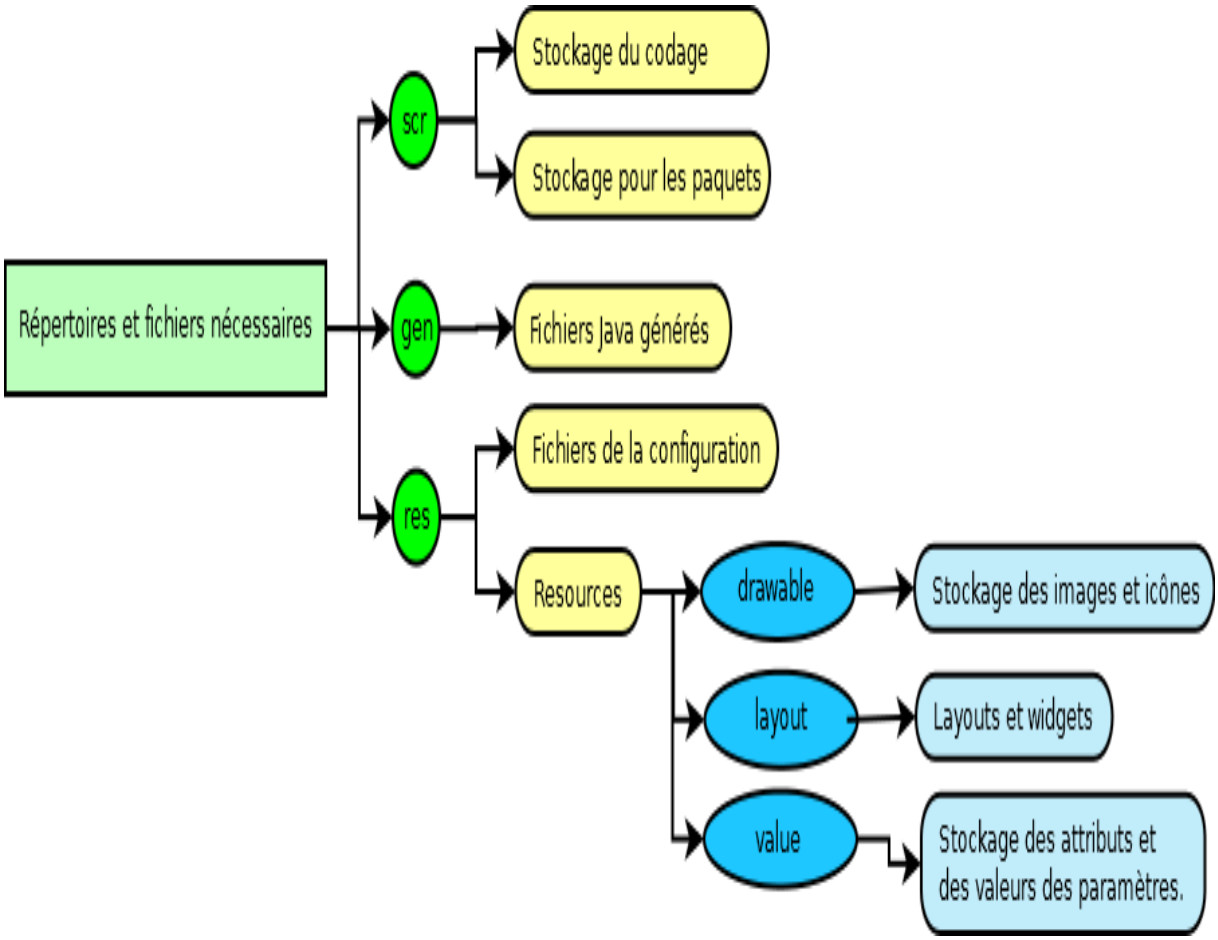

Figure3.4: Les répertoires et fichiers nécessaires de l'application Android

La figure 3.4représente les fichiers dans un projet de l'application android.

- Src : vient de mot << Source Code >>en anglais. Le << src >> est la partie de code source que nous avons créé, qui a été écrit dans le fichier <<Java>>, que nous allons voir le nom des attributs et méthodes de classe.
- gen : vient de mot << Generated Java Files >>en anglais. C'est une répartition qui a été faite automatiquement lors de la création du projet tel que R.java dans lequel se compose des textes et l'élément de l'interface en étant amené dans le projet par le plug-in d'Android. Ce fichier est comme un pointeur vers les répertoires de drawable, layout, values. res : vient de mot << Resource >> en anglais. Le res est pour stocker des fichiers

qui sont utilisés en collaboration avec le projet, tels que des images etc. Ilse décompose en 5 répertoire : drawable - hdpi, drawable - ldpi, drawable -mdpi, layout et values.

- AndroidManifest.xml est structuré en fichier xml, qui stocke des propriétés et des paramètres de la configuration d'application, telles que le nom des fonctions de l'application, la version du code, les droits d'autorisations pour accéder à l'application, etc.

La responsabilité de ce travail est de réaliser les générateurs de code à partir de l'étude de l'approche d'ingénierie dirigée par les modèles. Ces générateurs doit être capable de créer le code source pour l'application mobile à partir de modèle PIM. Le code généré doit être compilable aux applications natives pour les plateformes de l'appareil mobile .L'objectif attendu est que le code généré peut être utilisé en moins de modifications.

3.3. Conclusion

Ce chapitre présente la solution du développement des applications mobiles. On espère que cette approche puisse résoudre la problématique du développement des applications mobiles. Et pour prouver l'hypothèse, on implémentera cette solution proposée et on l'expérimentera à la suite.

Ce chapitre est la présentation de notre travail. C'est l'implémentation de la solution proposée et l'expérimentation pour examiner notre produit de projet. En fin on présente le résultat et l'évaluation.

Chapitre 04 :

Implémentation et Expérimentation

4.1 Introduction

Le travail sur la partie pratique du sujet << études de l'approche d'ingénierie dirigée par les modèles pour le développement des applications mobiles >> se contient deux parties. La première, c'est de modéliser le modèle indépendant des plateformes (PIM). Le modèle obtenu de cette partie joue le rôle comme la conception de l'application qu'on veut développer. Afin de réaliser ce modèle, on a besoin de définir un langage de modélisation, c'est un langage dédié (Domain-Specific language : DSL).

La deuxième est la partie du développement de générateur de code. On a choisi de développer le générateur de code pour l'application sur la plateforme <<Android >> Afin de réaliser les générateurs de code, on suit le principe pour le modèle spécification des plateformes (PSM).

4.2 Implémentation de langage dédié : DSL

Notre DSL est développé sous la conception de regrouper des éléments en trois composants : <<Model >>, << View >> et <<Control >>. Cela nous permet de réaliser la transformation de code pour les plateformes spécifiées. Parce qu'on a bien concevoir le structure, on peut transformer le code facilement. Il soutient également la modification plus facile de modèle si l'exigence a changé. La figure 4.1est la structure de notre DSL.

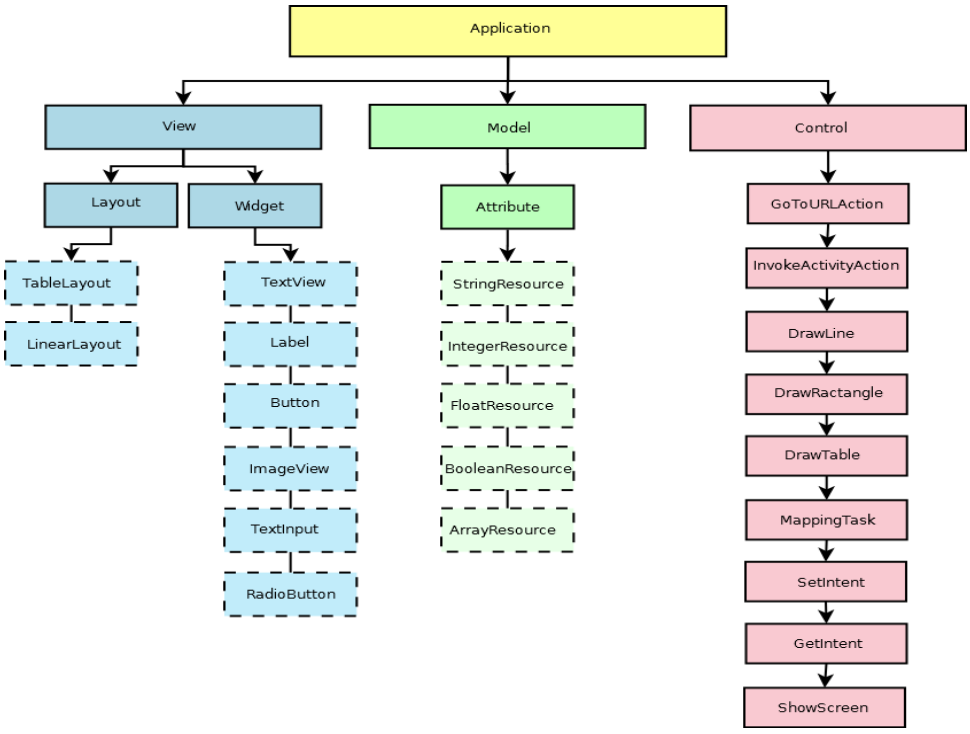

Figure 4.1: Le concept de DSL

4.2.1 Application

<< Application >> joue le rôle comme la racine de notre arbre qui contient les composants de << View >>, << Model >> et << Control >>. L'objet principal contient toutes les informations de base sur une application. Chaque projet doit contenir exactement un objet principal qui peut être dans un contrôleur arbitraire.

```
grammar org.xtext.MDAmobileDsl with org.eclipse.xtext.common.Terminals
generate mDAmobileDsl "http://www.xtext.org/MDAmobileDsl"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

MobileAppModel:
ModelElement+=Application;
Application:
'application' name=STRING '=>' packageName=QualifiedName '{'( ('version:'
versionCode=INT '=>' versionName=STRING)?& sdkVersion=ApplicationUsesSDK?&
('firstPage:' pageName=STRING)?)
(model+=Model | view+=View | control+=Controller)+'}'
;
```

4.2.2 Model

<< Model >> est le premier élément dans l'arbre est le << Model >>. Dans la partie de modèle, la structure des attributs et des valeurs des paramètres sont décrites.

```
Model:
        'Model' name=ID '{'modelElements += ModelElement+'}'
;
ModelElement:
       attributes += Attribute+
;
Attribute:
StringResource | IntegerResource | FloatResource |BooleanResource | ArrayResource
;
```

Ces attributs seront définissent le type, le nom et la description d'entité. Le type des attributs peut être String, Intègre, Float, Boolen ou Array.

```
StringResource:
'string' name=ID '=' value=STRING
;
IntegerResource:
'integer' name=ID '=' value=INT
;
FloatResource:
'float' name=ID '=' valur=FLOAT
;
FLOAT returns ecore::EDouble:
       INT '.' INT
;
BooleanResource:
'_bool' name=ID '=' value=BOOL
;
terminal BOOL returns ecore::EBooleanObject:
'TRUE' | 'FALSE'
;
ArrayResource:
'array'
(IntegerArrayResource | StringArrayResource )
;
IntegerArrayResource:
'(integer)'name=ID '=' '['(items+=INT (',' items+=INT)* )?']'
;
StringArrayResource:
'(string)'name=ID '=' '['(items+=STRING (',' items+=STRING)* )?']'
;
```

4.2.3 View

<<View >> est le deuxième élément. Les éléments de la partie de << View >> sont la mise en page (Layouts) et des composants d'interface (Widget).

```
View:
       'View' name=ID '{'viewElement+=ViewElement+'}';

ViewElement:
       Layout | Widget
;
```

Pour les << layout >>, on a maintenant définit juste 2 types, c'est << table layout>> et << linear layout >>. Chaque type de << layout >> doit être définit ses paramètre de caractères par exemple : la taille, la dimension, la marge, l'orientation, la gravité, le colon etc.

```
Layout:
      TableLayout |LinearLayout
;
```

```
TableLayout:
      'TableLayout' name=ID params+=TableLayoutParam (',' params+=TableLayoutParam)*
& params+= LayoutParams(','params+=LayoutParams)*'{' elements+=ViewElement* '}'
;
```

```
LinearLayout:
      'LinearLayout' name=ID params+= LayoutParams '{'elements+=ViewElement*'}'
;
```

Pour les << widget >> , on a définit le << widget >> de base pour les applications mobiles comme Text View, Botton, Label, ImageView, TextInput, RadioButton.

```
Widget:
      TextView|Button | Label | ImageView | TextInput | RadioButton
;
```

Text View et Label servent à présenter des textes pour l'utilisateur. Le << Text View>>est définit par le mot-clé << Text View >> suivi son nom et dans le parenthèse contient ses propriétés. C'est la même pour le Label mais ceci indiqué par le mot-clé Label.

```
TextView:
       'TextView' name=ID '(' text=STRING ')' ('{'('height:' height=WidgetSize';')?
& ('width:' width=WidgetSize';')?&('textColor:' textColor=Color';' )?& ('gravity:'
gravity =GravityType';')?& ('isScrollContainer:' isScrollContainer=BooleanVal';')?'}'
)
;
```

```
Label:
      'Label' name=ID '(' text=STRING ')' ('{' ('height:' height=WidgetSize';')? &
('width:' width=WidgetSize';')? & ('textColor:' textColor=Color';' )?&('gravity:'
gravity =GravityType';')?'}')
;
```

Botton permet à l'utilisateur la possibilité d'appeler des actions qui ont été liés à des événements de la touche. Pour la définition de bouton, il est indiqué par le mot-clé << Botton >> suivi son nom et il existe la définition par défaut suivant.

```
Button:
	'Button' name=ID '(' text=STRING ')' ('{'('height:' height=WidgetSize';')?&
('width:' width=WidgetSize';')?&('textColor:' textColor=Color';' )?& ('gravity:'
gravity =GravityType';')?& ('clickable:' actionName=STRING ';' )?'}')
;
```

ImageView sert à afficher l'image aux utilisateurs. Il est indiqué par le mot-clé <<ImageView >> suivi son nom et le chemin où se trouve l'image. Et dans le parenthèse se compose ses description.

```
ImageView:
	'Image' name=ID ('{'('height:' height=WidgetSize';')? & ('width:'
width=WidgetSize';')?&('clickable:' clickable=Action ';' )?& ('src:' src=STRING ';'
)?'}'  )
;
```

TextInput est un champ pour mettre des entrées de l'utilisateur, indiqué par le mot-clé << Text Input >> suivi son nom et ses descriptions dans la parenthèse.

```
TextInput:
	'TextInput' name=ID ('{' ('text:' labelText=STRING';')? & ('type:'
type=Texttype';')? & ('height:' height=WidgetSize';')? &
('width:'width=WidgetSize';')?& ('textColor:' textColor=Color';' )?& ('gravity:'
gravity =GravityType';')?'}')
;
```

RadioButton sert à définir le choix, il peut y avoir plusieurs choix comme une liste des choix. La valeur est <<ture>>et << false >>.

```
RadioButton:
	'RadioButton' name=ID '{' (('label' labelText=STRING)? & ('checked'
(checked?=BooleanVal))?) '}';
```

4.2.4 Control

<< Control >> est la dernière partie de notre DSL. Il sert à définir des actions de l'application. Les éléments de << View >> et les données de Model sont reliés avec les actions par la désignation dans le control. Une classe de control peut se contient plusieurs actions. Chaque action est décrite en forme de << code Fragment >>. Pour l'instant, on peut définir les actions par les événements de base comme << onTouch >>, << onLeftSwipe >>,<<

onRightSwipe >>, etc. pour les éléments de la partie << View >> et faire le mappage entre les éléments de << View >> et de Model. Les images suivantes représentent quelques parties du code des descriptions de la partie de Control.

```
Controller:
	'Class' name=ID '{'Actions += Action+'}'
;

Action:
	'Action' name = ID '{'codeFragments += CustomCodeFragment*'}'
;

CustomCodeFragment:
	{ActionTask} ('act' actions+= ActionDef+  'on' events += EventDef+)
;
```

4.3 Implémentation de générateur de code

Pour le développement du générateur de code pour l'application mobile par l'aide de l'approche d'ingénierie dirigé par les modèles, il est très important de définir la règle de la transformation parce que la transformation est la clé de génération le code de la plateforme cible à partir de modèle PIM. Une fois que le développeur a modélisé le modèle PIM qui est compté comme l'entrée de génération, le développeur doit faire passer ce modèle à travers le générateur de code.

Afin de développer le générateur de code, on peut exploiter Acceleo qui est un outil visant à définir la transformation de modèle en texte, dans notre cas, un DSL à la transformation de l'application Android.

Après avoir installé Acceleo sur l'éditeur d'Eclipse, on crée un projet Acceleo, choisit le méta-modèle de notre DSL (on a nommé le DSL <<_MDAmobileDsl_>>) qui devrait être dans la liste. Ensuite on crée le module, dans lequel on décrit la transformation. Voyant dans la figure4.2, le module se trouve au milieu de l'architecture.

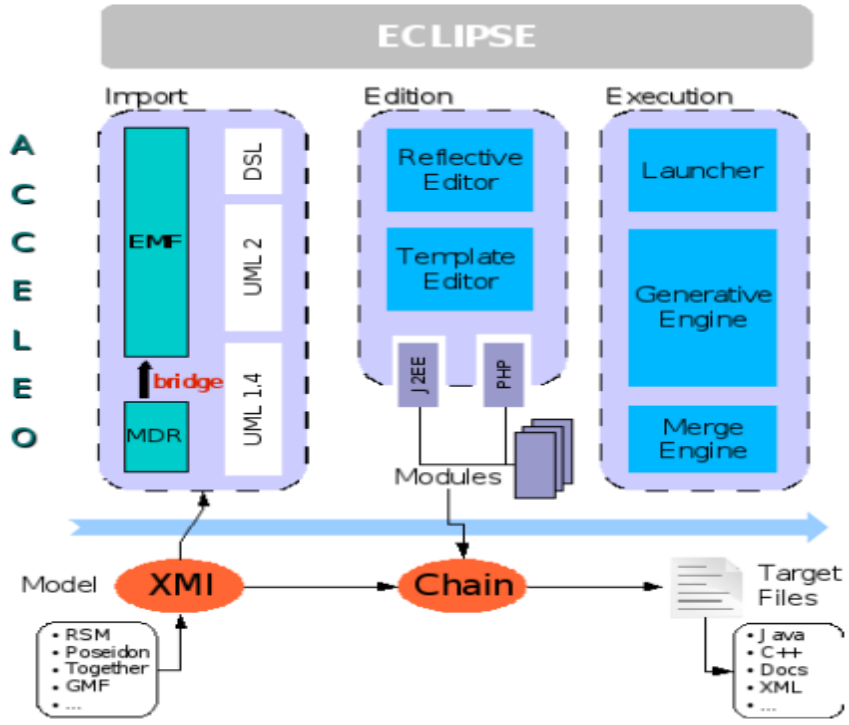

Figure 4.2 : L'architecture de Acceleo

Les modules seront la chaîne de génération pour l'objectif de générer le code. Cette chaîne doit concevoir et être décrit sur le correspondant à la conception de plateforme cible. La figure 4.3est le concept de la transformation pour la plateforme Android.

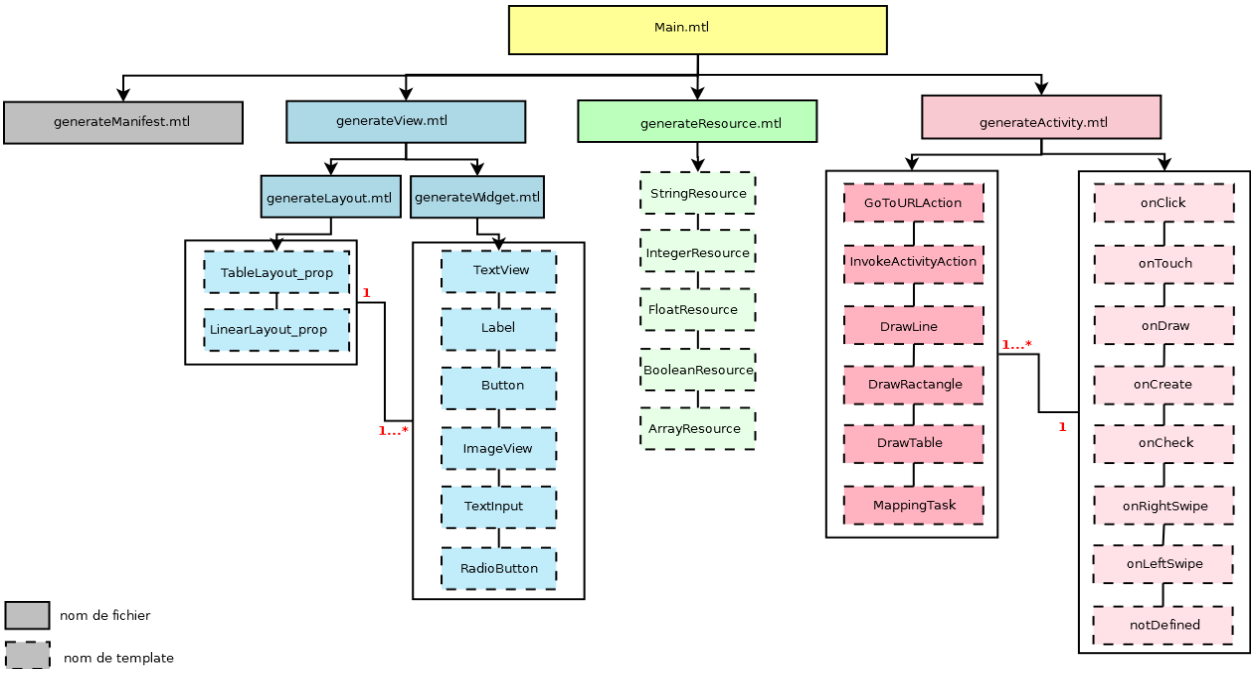

Figure 4.3: Le concept de la transformation pour la plateforme Android

Un module correspond à un fichier <<.mlt>> Selon le concept de la plateforme Android 3.4, le générateur doit être capable de générer le fichier nécessaire pour que l'application puisse fonctionner. Au début de l'exécution du modèle de générateur, le module Main qui est le module principal de modèle de générateur, faire l'appel sur des autres modules. On va présenter maintenant chaque module du modèle de générateur de code pour l'application d'Android.

4.3.1 Le module <<generateManifest>>

Le module << generateManifest >> sert à générer le fichier AndroidManifest.xml parla définition dans le Template<< generateManifestFile >> qui va tirer des informations de l'application qui est définit dans la partie de l'application dans le modèle PIM.

43.2 Le module <<generateView>>

Le module << generateView >> sert à générer le répertoire <<layout>> et le fichier dans ce répertoire. Le concept de génération est de générer le fichier .xml qui se compose les Layout et les widget. Il va faire l'appel sur le Template<< generateLayoutProp>> (qui se trouve dans le fichier generateLayout.mtl) pour générer la propriété de layout qui est définit à partir de DSL, et fait l'appel sur le Template<< generateWidget >> (qui se trouve dans le fichier generateWidget.mtl) pour générer le xml de la propriété des widget. Un layout peut se compose plusieurs widgets dedans. On met l'appel des Template<< for>>.La boucle va s'arrêter lorsqu'il n'y a plus des Layout et des widgets définies à partir de DSL.

4.3.3 Le module <<generate Resource>>

Le module generate Resource sert à générer le répertoire <<values>>et les fichiers dans ce répertoire. Les valeurs de ressource qui sont définies dans la partie de Model de modèle PIM sera transformé en forme de fichier.xml tel que 'res/values/gen strings. Xml','/res/values/gen-bools.xml' etc. selon le type de l'attribut.

4.3.4 Le module <<generate Activity>>

Le module generate Activity sert à générer le répertoire <<src>> le et les fichiers.java.Une class dans le modèle PIM est généré à un fichier.java. Les actions seront générées sous la méthode de l'événement définit. La Template<< eventcode>> sera appelé lorsque la définition de <<Event Def >> de la partie Control dans le modèle se compose l'événement correspondance au paramètre dans le Template. Et c'est pareil pour le Template de << action Code>> qui attente la définition dans le << Action Def >> de modèle PIM .Sous le projet de l'Acceleo, on a créé le répertoire << model >> pour mettre le modèle PIM qui est le modèle d'entrée et le répertoire<< out >> pour mettre le code généré.

4.4 Application de la méthode proposée

Dans notre étude, on réalise deux modèles PIM à partir de notre DSL pour l'expérimentation. Un modèle PIM est le modèle d'une simple application mobile et un autre PIM est le modèle d'une application. Ces deux modèles jouent le rôle de modèle d'entrée afin de générer le code de plateforme cible. Cette partie du rapport représente la méthodologie, les résultats et les analyses des travaux pratiques.

4.4.1 Processus de développement

Pour tester notre DSL, on essaie de réaliser deux modèle PIM de l'application mobile dans le projet Acceleo pour Android. Les étapes qu'on va présenter suivants, sont le processus de développement une application mobile en utilisant notre générateur de code.

- Sur le projet Acceleo << org.acceleo.example.droid>>, faites un clique à droit sur le dossier de
  <<Model>>.
- Choisissez Nouveau Fichier.
- Créez un fichier nommé <<model >>.
- Ecrire le code selon la syntaxe de notre DSL.

Ensuite on passe le modèle PIM à travers le générateur comme les étapes suivantes.

- Sur le projet Acceleo org.acceleo.example.droid, faites un clique à droit sur le projet.
- Choisissez Run configuration.
- Complétez la configuration de l'exécution comme la figure 4.4
- Cliquez << Run >>, vous obteniez le résultat comme la figure4.5

4.2.2 Application 1 : Une application simple

Pour le premier test de générateur, on crée une application très simple. Les fonctionnalités de cette application ne sont pas compliquées. Voici le concept de l'application simple.

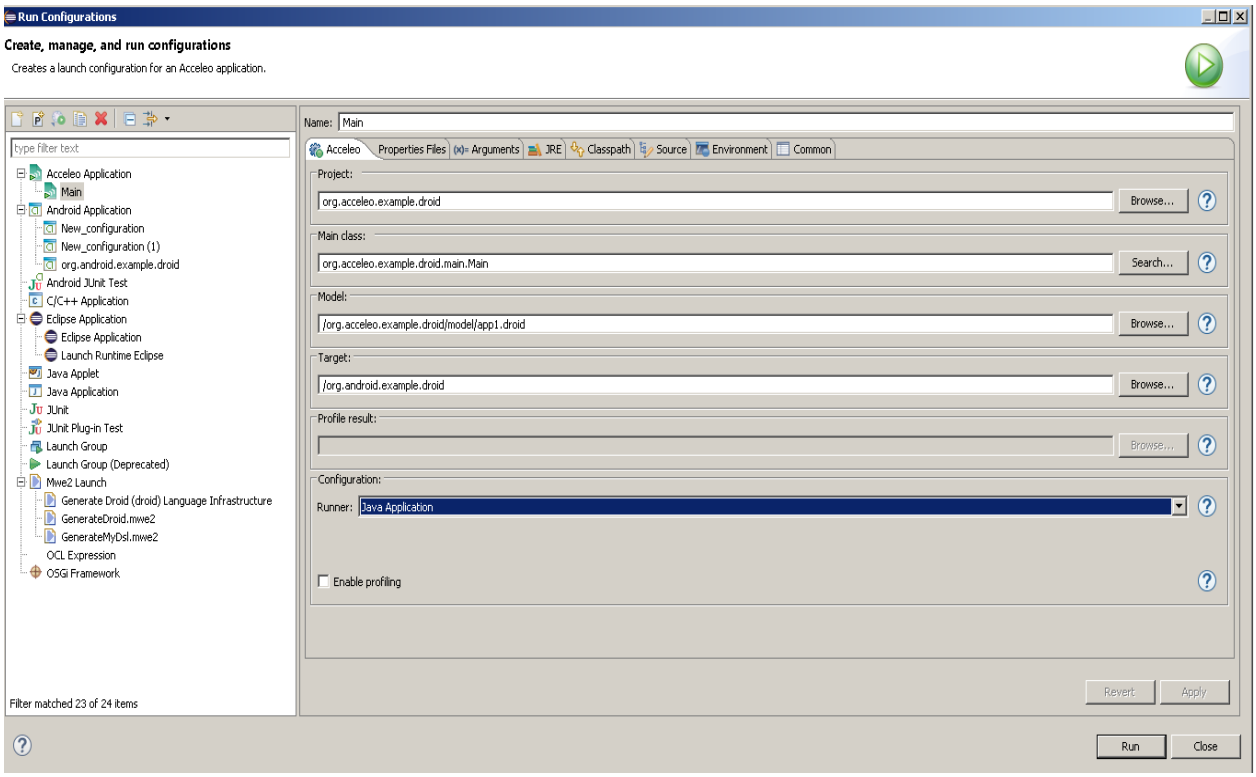

Figure 4.4:La configuration de l'exécution d'Acceleo

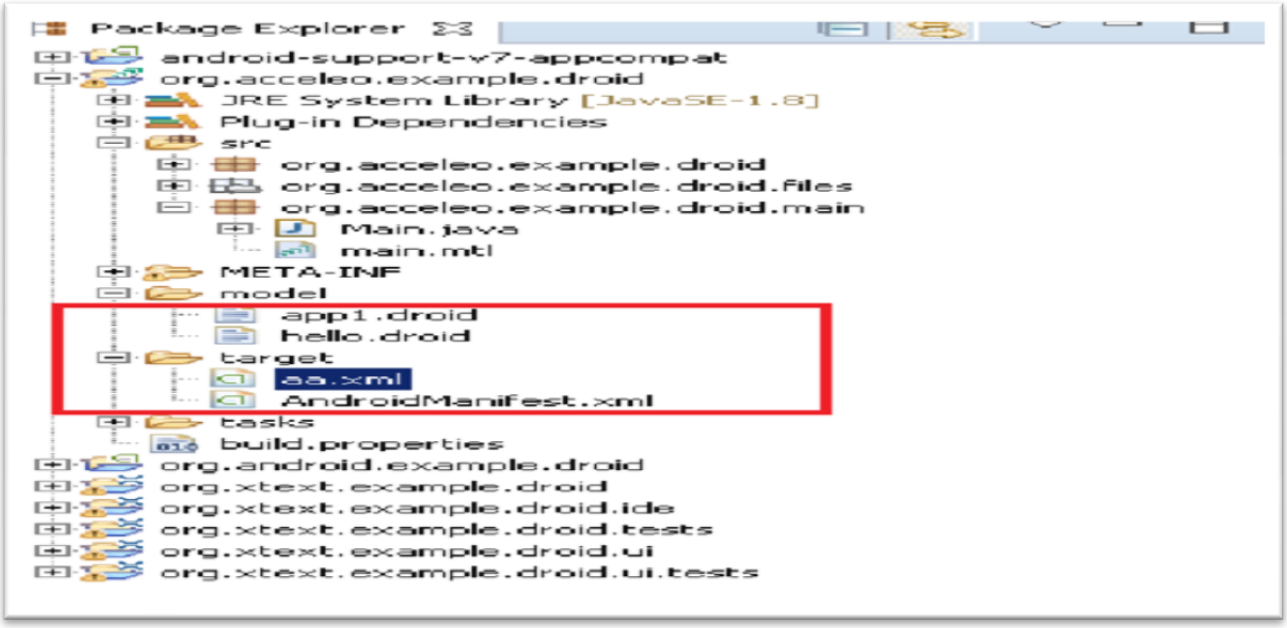

Figure 4.5:La location du code généré

4.2.2 .1 Description de l'application

Cette application se compose deux pages. Ce sont le Main page et Seconde page. La conception d'interface est représenté dans le tableur 6 et 7. Les fonctionnalités de l'application sont comme les suivants :

- La page Main
  - Un <<label>> doit être capable d'afficher le nom de l'application.
  - Un <<text input>> est prêt pour qu'un utilisateur puisse entrer le nom.
  - Le nom entré par l'utilisateur dans le <<text input>> doit être transmettre à la page Second lorsque l'utilisateur touche le bouton <<Show>>.
  - Un bouton <<Next>> peut amener l'utilisateur à la page Second.
  - La page Second
  - Un <<label>> doit être capable d'afficher le nom de l'application.
  - Un <<label>> affiche le nom d'entré à partir du <<text input>> de la page Main.
  - Un <<Image Viewer>> doit être capable de représenter un image.
  - Un bouton <<Seach>> peut amener l'utilisateur au site web <<http /www.google.com>>.

Une fois on a décrit l'application, on transforme des exigences à un modèle PIM par le langage DSL.

4 .2.2.2 Résulta

Après avoir modélisé un modèle PIM correspondant à la conception, on fait passer le modèle à travers le générateur de code et obtient le résultat suivant :

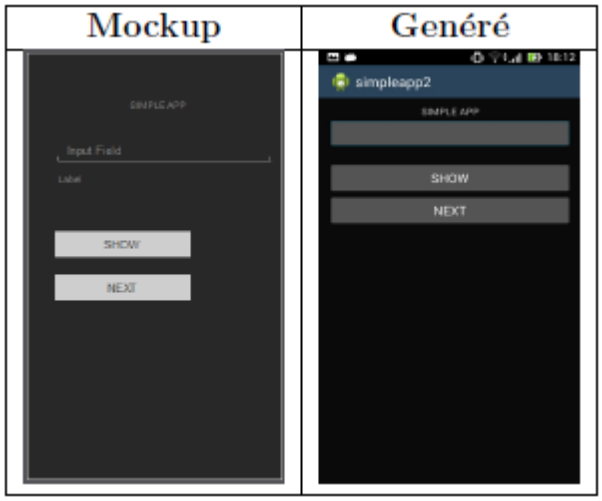

Tableau 6: La comparaison d'interface de Main

Pour le page Main au niveau de fonctionnalité, notre générateur est capable de généré le code qui peut faire l'application fonctionnant comme tous les concepts cibles.

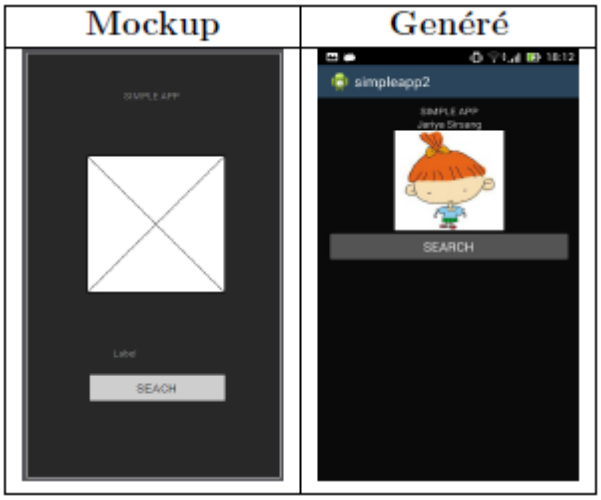

Tableau 7: La comparaison d'interface de Second

C'est pareil pour le page Second. Si l'on fait la comparaison des fonctions d'application entre ces à partir de la conception et ces à partir de code généré, ils sont la même.

4.2.2.3 Conclusion

Selon le concept de l'application simple qu'on a définit, notre générateur de code est capable de généré le code pour l'application d'Android. On peut faire la conclusion qu'il peut générer les fonctions comme la suit :

- View : On a fait le test sur le layout <<Linear Layout>>, le <<Label>>, le <<Text Input>>, le <<Button>> et le <<Image Viewer>>. Le générateur de code a donné le code qui peut fait fonctionner bien pour tous les éléments testés. Le code en forme << xml >> est bien structuré comme le code qui est réalisé manuellement.
- Control : Sur la partie de <<Control>>, on va parler de test sur les fichiers .java qui fait marcher des actions d'application. Le code généré est capable de faire marcher l'application.

4.3 Evaluation

A la fin de notre l'étude sur l'approche d'ingénierie dirigée par les modèles pour le développement des applications mobiles, on peut appliquer cette approche pour le développement d'application mobile de plateforme d'Android.

Dans cette partie, on souhaite de donner l'analyse sur le résultat de notre travail comme la suit :

- L'approche choisi : après avoir étudié, on trouve que parmi les plusieurs approches d'ingénierie par les modèles, l'approche MDA est le plus agréable pour le développement des applications mobiles. Le MDA est standardisé par l'Objet Management Group (OMG) donc la méthodologie de cette approche est claire et pour implémenter on peut le faire de façon appropriée en basant sur le standard. Cependant il existe de nombreuses techniques dans le cadre de l'approche MDA pour la modélisation et la transformation de modèles.
- Au niveau de méta-modèle, on a développé un langage dédié pour le modèle des applications mobile. On a conçu ce DSL afin de modéliser le modèle qui est segmenté en trois parties : la conception d'interface, des ressources et des actions de l'application. Cela permet d'implémenter et de réviser facilement le modèle lorsque le développeur a besoin de modifier le modèle. Cependant, ce DSL n'est pas très couvert son domaine à cause de la taille de domaine et de notre expérience sur le développement des applications mobiles. Autrement dire que le domaine des applications mobiles a plusieurs fonctionnalités mais notre DSL fournit juste quelques fonctions principales. Aussi pour l'interface, il reste encore beaucoup de types layout et de widgets
- Au niveau du développement de générateur de code, c'est dommage qu'on puisse développer seulement pour une plateforme Android. Mais la méthodologie pour développer le générateur de code pour les autres plateformes (iOS, Blackberry, Windows Phone etc) est pareille. Ce qu'il est essentiel est qu'on doit avoir des connaissances de chaque plateforme sur le concept, la structure, les fonctionnalités fournis par la plate-forme etc. Néanmoins on peut générer le codage de l'application Android à partir de modèle PIM sans modification pour une application simple et besoin de la modification pour le cas de l'application. En raison que le développement d'un jeu, il est plus compliqué que d'autres applications, on a besoin de modifier et ajouter la partie de logique dans le code généré à partir de notre générateur.

Chapitre : 05

Conclusion et Perspective

5.1 Conclusion

L'objectif de ce mémoire est d'étudier l'approche d'ingénierie dirigée par les modèles a Fin de trouver la solution qui aide les développeurs de développer plus rapidement des applications mobiles. On a étudié des avantages de l'approche d'ingénierie dirigée par les modèles et trouvé que l'idée de l'approche d'architecture dirigée par les modèles (Model Driven Architecture :MDA) est plus appropriée pour le développement des applications mobiles. C'est la raison pour laquelle on a proposé la solution appliquant l'approche MDA afin de développer des applications mobiles.

Pour le fait, on commence à concevoir un langage dédié (Domain Specific language :DSL) qui joue le rôle de méta-modèle sur le domaine de l'application mobile par le Framework Xtext qui permet de développer le langage de programmation et les langages dédiés (DSL). On définisse le concept de notre DSL en trois parties. La première partie est conçue pour définir l'interface de l'application. Elle se compose des dispositions pour la mise en page (Layouts) et des composants d'interface (Widget). La deuxième partie a le concept de définir le contrôle de l'application. Pour qu'une application puisse fonctionner, il a besoin de définir des actions, ces actions sont définit grâce à cette partie.

En fin la troisième partie est conçu pour fournir la définition des modèles des variables dans l'application. Autrement dit que cette partie sert à définir des ressources qui seront utilisé dans l'application. Le produit de cette étape est un langage dédié pour notre domaine de l'application mobile en forme de plugin sur l'éditeur Eclipse.

Ce langage permet aux développeurs de construire un modèle indépendant de plateforme (PIM) sur le type de fichier <<.Droit>>. Une fois qu'on a le langage dédié, l'étape suivant est de définir le modèle spécifique à la plateforme(PSM). On doit définir la grammaire de la transformation pour des différentes plateformes. Malheureusement, à cause du temps limité et la limite d'expérience sur le développement mobile, on peut définir le PSM juste pour la plateforme Android. L'Acceleo est choisi comme l'outil de cette étape. On a décrit la grammaire en respectant la structure de la plateforme Android. Le résultat de cette étape est le générateur de code pour l'application mobile de plateforme Android. Après que le développeur modélise le modèle PIM par notre DSL, il doit faire passer le modèle vers notre générateur afin d'obtenir le codage de l'application mobile de chaque plateforme.

5.2 Perspective

Même si l'on a réussi de faire le générateur de code à partir de modèle comme le principe de l'approche d'ingénierie dirigée par les modèles, il reste encore le travail à améliorer et à compléter. Puisque le domaine de l'application mobile est large, notre DSL ne couvert pas tout le domaine. Il nous faut fournir plus d'éléments qui permettent de modéliser une application mobile plus compliquée par exemple : au niveau de l'interface (View), il a besoin d'autres Layouts et widgets. Au niveau de l'action aussi, il faut ajouter des éléments qui aident à définir des autres fonctions. Comme l'application du jeu, il est nécessaire de définir la logique du jeu, malheureusement on n'atteint pas à fournit le soutien de cette fonction. Ceci est important d'améliorer dans l'avenir. Pour le développement du générateur de code, on a pris la technique M2T, implémentée par l'outil Acceleo. On a développé seulement le générateur de code pour la plateforme mobile. Il sera mieux de développer aussi les générateurs des autres plateformes comme l'iOS, le Blackberry, le Windows Phone.

Références

- Acceleo (2014). Acceleo - transforming models into code. Retrieved from http://www.eclipse.org/acceleo/.Albert, M., Muñoz, J., Pelechano, V., & Pastor, Ó. (2006, November). Model to text transformation in practice:generating code from rich associations specifications. In International Conference on ConceptualModeling (pp. 63-72). Springer Berlin Heidelberg.
- Allilaire, F., Bézivin, J., Jouault, F., & Kurtev, I. (2006). ATL-eclipse support for model transformation.In Proceedings of the Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference,Nantes, France (Vol. 66).
- Android (2017). Platform Architecture. Retrieved fromhttps://developer.android.com/guide/platform/index.html#api-framework.
- AndroMDA (2014). Generate components quickly with AndroMDA. Retrieved fromhttp://andromda.sourceforge.net.
- Baixing, Q., Chen, T., Dai, H., Peng, B., & Wu, M. (2012, June). A Cross-platform Application Development
- Barbier, F., Deltombe, G., Parisy, O., & Youbi, K. (2011, February). Model driven reverse engineering:
- Increasing legacy technology independence. In Second India Workshop on Reverse Engineering (Vol.125, pp. 126-139).
- Bernoussi, I. (2008). Les DSL, un standard dans 2 ans ? Programmez. Le magazine des développeurs.
- Bézivin, J., & Gerbé, O. (2001, November). Towards a precise definition of the OMG/MDA framework.
- Bézivin, J., & Briot, J. P. (2004). Sur les principes de base de l'ingénierie des modèles. L'OBJET, 10(4), 145-157.
- Bettini, L. (2016). Implementing domain-specific languages with Xtext and Xtend. Packt Publishing Ltd.
- Bluage. (2015). Blu Age Forward. Retrieved from http://www.bluage.com/en/en_product/en-baforward.
- Bohlen, M. (2007). AndroMDA. Retrieved from http://www.andromda.org.
- Burns, E., Schalk, C., & Griffin, N. (2010). JavaServer Faces 2.0. McGraw-Hill.

Références

- Cabot, J. (2015). Clarifying concepts: MBE vs MDE vs MDD vs MDA. Retrieved fromhttp://modelinglanguages.com/clarifying-concepts-mbe-vs-mde-vs-mdd-vs-mda/.
- Corral, L., Janes, A., & Remencius, T. (2012). Potential Advantages and Disadvantages of Multiplatfonn
- Ehringer, D. (2010). The dalvik virtual machine architecture. Techn. report (March 2010), 4, 8.
- Farail, P., Gaufillet, P., Canals, A., Le Camus, C., Sciamma, D., Michel, P., & Pantel, M. (2006). The
- TOPCASED project: a toolkit in open source for critical aeronautic systems design. Embedded Real
- Time Software (ERTS), 781, 54-59.
- Fowler, M. (2010). Domain-Specific Languages Addison-Wesley Professional. Boston, MA, USA.
- Gartner (March, 2015). Gartner Says Smartphone Sales Surpassed One Billion Units in 2014. Retrieved fromhttp://www.gartner.com/newsroom/id/2996817 (Accessed on December 3, 2015).
- Gronback, R. C. (2009). Eclipse modeling project: a domain-specific language (DSL) toolkit. PearsonEducation.
- Hailpern, B., & Tarr, P. (2006). Model-driven development: The good, the bad, and the ugly. IBM systemsjournal, 45(3), 451-461.
- Herndon, R. M., & Berzins, V. A. (1988). The realizable benefits of a language prototyping language. IEEETransactions on Software Engineering, 14(6), 803-809.
- Hutchinson, J., Whittle, J., & Rouncefield, M. (2014). Model-driven engineering practices in industry: Social,organizational and managerial factors that lead to success or failure. Science of Computer144Programming, 89, 144-161.
- Jézéquel, J. M., Combemale, B., & Vojtisek, D. (2012). Ingénierie Dirigée par les Modèles: des concepts à lapratique. (p. 144). Ellipses.
- Kerensen Consulting (2015). Evolution des usages Mobiles, prévision 2015.
- Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. ACMcomputing surveys (CSUR), 37(4), 316-344.
- Minsky, M. (1965). Matter, mind and models. International Federation for Information Processing (IFIP), 1, 45-50.

Références

- OMG (2002). Meta Object Facility (MOF) Specification, Version 1.4, April 2002.Object Management Group (2003). MDA Guide Version 1.0.1. omg/2003-06-01.
- Perchat, J., Desertot, M., & Lecomte, S. (2013). Component based framework to create mobile cross-platformapplications. Procedia Computer Science, 19, (pp. 1004-1011).
- Raj, C. R., & Tolety, S. B. (2012, December). A study on approaches to build cross-platform mobileapplications and criteria to select appropriate approach. In India Conference (INDICON), 2012 AnnualIEEE (pp. 625-629). IEEE.
- Sambasivan, D., John, N., Udayakumar, S., & Gupta, R. (2011, November). Generic framework for mobileapplication development. In Internet (AH-ICI), 2011 Second Asian Himalayas InternationalConference on (pp. 1-5). IEEE.
- Selic, B. (2007, May). A systematic approach to domain-specific language design using UML. In Object andComponent-Oriented Real-Time Distributed Computing, 2007. ISORC'07. 10th IEEE InternationalSymposium on (pp. 2-9). IEEE.
- Tracy, K. W. (2012). Mobile application development experiences on Apple's iOS and Android OS. IeeePotentials, 31(4), 30-34.
- Vallecillo, A. (2010, June). On the combination of domain specific modeling languages. In EuropeanConference on Modelling Foundations and Applications (pp. 305-320). Springer Berlin Heidelberg.
- Xanthopoulos, S., & Xinogalos, S. (2013, September). A comparative analysis of cross-platform developmentapproaches for mobile applications. In Proceedings of the 6th Balkan Conference in Informatics (pp.213-220). ACM.
- Yung-Wei, K. W., Lin, C. F., Yang, K. A., & Yuan, S. M. (2011, October). A cross-platform runtimeenvironment for mobile widget-based application. In Cyber-Enabled Distributed Computing andKnowledge Discovery (CyberC), 2011 International Conference on (pp. 68-71). IEEE.
- Rédouane Lbath Diaw, Samba and Bernard Coulette. Etat de l'art sur le développement logiciel dirigé par les modèles. 2008.
- The Eclipse Foundation. Xtext. 2014. URL http://www.eclipse.org/xtext/.
- The Eclipse Foundation. Acceleo. 2014. URL http://www.acceleo.org/.
- Inc Object Management Group. Omg's metaobject facility(mof). 2014. URL http://www.omg.org/mof/.

Références

- Google Inc.'s. Android developers. 2014. URL http://developer.android.com/guide/index.html.