Ministère de l'enseignement supérieur et de la recherche scientifique
Centre Universitaire de Naâma
- Salhi Ahmed -

Mémoire de fin d'études
Pour l'obtention du diplôme de Master en Informatique
Option : Systèmes d'information

Thème :

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

Réalisé par :
Mohammed Kadri

Encadré par :
Nadri Khiati

2019/2020

Préface

Je tiens à remercier tous ceux qui ont été une cause pour que je puisse réaliser ce travail.

Dieu merci pour son aide.

Je remercie mes parents et mes proche pour leurs soussienne et inquiétude et pour les bonnes intentions qu'ils one eux l'occasion de montrer durant toute cette période de ce projet.

Je tiens à remercier énormément mon encadreur qui a été d'une très remarquable, c'était plus que je ne pouvais jamais imaginer.

Je remercie mes collègues et amie qui n'ont pas hésité à m'aider dans toutes formes.

Je remercie aussi le personnel de l'université qui a créé les conditions nécessaire pour ce travail.

Résumé

Le nombre utilisateurs de smartphone est en augmentation continue de jours en jours jusqu'au point où il est devenue très dominant dans le cadre des technologies utilisées. Cette augmentation à induit une forte demande en matière d'applications mobiles qui semble devenir utilisées dans tous les aspects de la vie quotidienne. Vu que ces smartphone utilisent des plateformes différentes les fournisseurs de logiciel se trouvent amenée à fournir les mêmes applications qui portent les mêmes fonctionnalités pour différentes plateformes. Ce qui engendre un cout considérable en matière de développement et maintenance logiciel.

Vue les avantages des applications natives par rapport aux autres méthodes du développement mobile, ce sujet essaie de proposer une solution qui permet de produire des applications qui sont au même temps multiplateformes mais aussi avec les qualités des applications native.

La solution consiste à créer un langage spécifique au domaine de développement mobile, qui permettra de définir un modèle pour l'application qui sera ensuite automatiquement transformé en des codes natifs pour les différentes plateformes ciblées. Et ce en utilisant l'ingénierie dirigé par les modèles.

Mots clés : Ingénierie dirigée par les modèles, Application mobile, Multiplateforme, DSL.

Abstract

The number of smartphone users is continuously increasing day by day until the point where it has become very dominant within the technologies used. This increase has led to a strong demand for mobile applications that seems to be more and more present used in many aspects of daily life. As these smartphones run on different platforms, software providers find themselves obliged to provide the same applications which have the same functionality for each and every targeted platform. This generates a considerable cost in terms of software development and maintenance.

Given the advantages of native applications compared to other methods of mobile development, this subject tries to propose a solution that makes it possible to produce applications that are at the same time cross-platform and full of the qualities of native applications.

The solution is to create a language specific to the mobile development domain, which will allow a model to be defined for the application which will then be automatically transformed into native codes for the various targeted platforms. It will be realized using model-driven engineering.

Key words: Model driven engineering, Mobile applications, Cross-platform, DSL.

ملخص

يوما بعد يوم فإن عدد مستعملي الهواتف الذكية في تزايد مستمر إلى حد طغيانها على سوق المنتجات التكنولوجية المستعملة. تسبب هذا التزايد في توليد طلب قوي من ناحية برامج الهاتف الذكي والتي صارت مستعملة في كثير اليومية. ولما كانت هته الهواتف المتوفرة تشتغل على أنظمة مختلفة، صار أصحاب البرامج من جوانب الحياة مضطرين إلى توفير نفس البرامج التي تملك تنفذ نفس العمليات للأنظمة المختلفة. الأمر الذي كلفهم ثمنا معتبرا من ناحية تطوير وصيانة هته البرامج.

نظرًا لمزايا التطبيقات الأصلية مقارنة بالطرق الأخرى لتطوير تطبيقات الهواتف الذكية، يحاول هذا الموضوع اقتراح حل يجعل من الممكن إنتاج تطبيقات تعمل في نفس الوقت عبر الأنظمة المختلفة ولكن أيضًا مع خصائص التطبيقات الأصلية.

يتمثل الحل في إنشاء لغة خاصة بمجال تطوير برامج الهواتف الذكية، مما يسمح بتحديد نموذج للتطبيق والذي سيتم بعد ذلك تحويله تلقائيًا إلى اللغة أصلية لمختلف الأنظمة الأساسية المستهدفة. وذلك باستخدام الهندسة النموذجية.

كلمات مفتاحية: الهندسة الموجهة بالنماذج، تطبيقات الهواتف الذكية، متعدد الأنظمة، لغة خاصة بمجال معين.

Liste des figures

Liste des tableaux

Sommaire

Introduction générale

Contexte (Introduction)

D'après Statista[1] le nombre actuel d'utilisateurs de smartphones dans le monde est de 3.5 milliards. Ça signifie que 45.12% de la population du monde possède un smartphone. Ce chiffre a considérablement augmenté depuis 2016, où il y avait 2.5 milliards seulement soit 33.58% de la population mondial de cette année.

Voice une figure qui représente le nombre d'utilisateurs de smartphones par milliards par année.

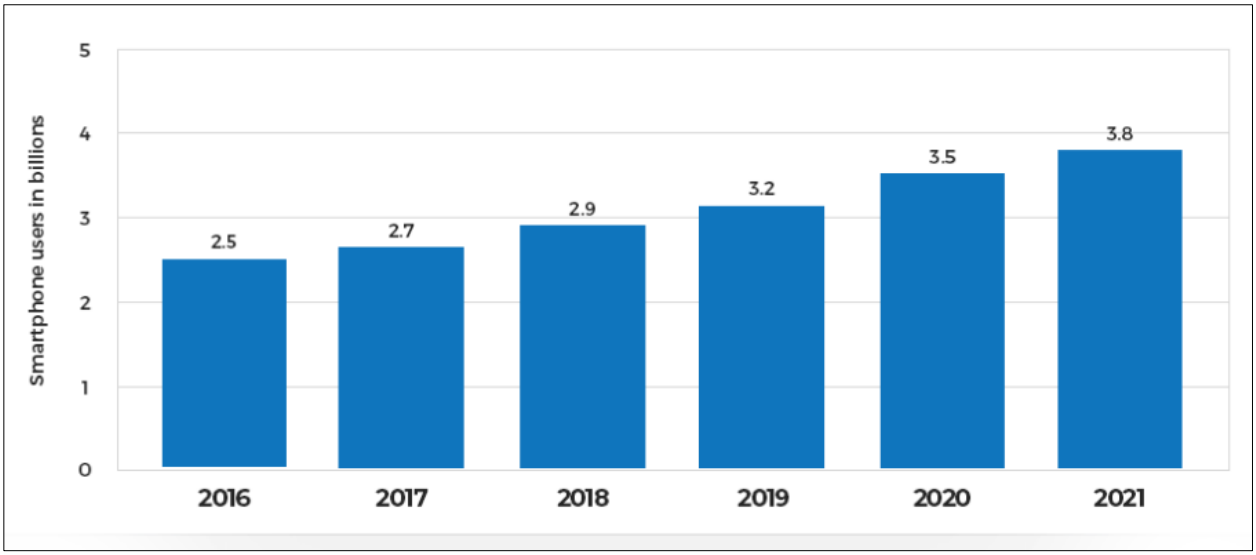

Figure 1 Nombre d'utilisateurs de smartphones par année

On remarque que le taux d'utilisateurs de smartphone a augmenté de 40% depuis 2016 jusqu'à 2020. On estime qu'en 20252 72% des utilisateurs d'internet vont exclusivement utiliser les smartphone pour accéder au web.

On se trouve dans une réalité où les smartphone ne cessent d'envahir le marché. Grace à cela, le marché des applications mobiles est de plus en plus élargi. Chaque jour, les sociétés lancent des nouvelles applications pour répondre à l'augmentation de consommation des appareils mobiles et aux besoins des utilisateurs.

[1] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/
[2] https://www.cnbc.com/2019/01/24/smartphones-72percent-of-people-will-use-only-mobile-for-internet-by-2025.html

Problématique

Le marché des smartphones est distribué sur différents entreprises et il se trouve que ces entreprises utilisent des plateformes différentes pour leurs produits. Cette divergence a fait que les développeurs d'applications mobiles se trouvent souvent amenés à développer leur application sous plusieurs formats afin de répondre à la forte demande des clients qui utilisent différents plateformes mobiles. Cette situation s'avère assez couteuse pour les développeurs car ils ont à supporter les charges de la réalisation et la maintenance d'une application pour différentes plateformes mobiles, bien que ce soient toujours les mêmes fonctionnalités qui doivent être réalisées pour chaque plateforme. La question qui se pose c'est est ce qu'il y'a un moyen pour développer une application qui soit utilisable sur plusieurs plateformes ?

Pour répondre à cette question, 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 etc.).

Objectifs de l'étude

L'approche d'ingénierie dirigée par les modèles est une nouvelle approche intéressante pour le développement des applications mobiles. Cela peut simplifier le développement et réduire l'intervalle entre des requêtes et l'implémentation. Le modèle représente le métier principal de l'application, ce modèle servira de base pour générer automatiquement le code source pour chaque plateforme cible. De ce fait on aura éliminé un cout considérable on matière de développement et maintenance logicielle, en mettant en place une approche pour la modélisation et la génération des applications mobiles multiplateformes selon une approche native. Ca va permettre de diminuer le coût et le temps de développement des applications mobiles multiplateformes tout en offrant des applications de qualité bénéficiant de toutes les fonctionnalités natives des Smartphones.

Plan du mémoire

Dance ce mémoire nous allons en premier lieu analyser notre domaine d'étude. Nous commencerons par les plateformes mobiles et les méthodes de développent de leurs applications pour ensuite faire une étude sur l'ingénierie dirigée par les modèles. Après cela nous allons proposer une solution théorique à notre problèmes pour enfin proposé un exemple de mise en œuvre.

1. Technologie mobile :

1.1. Introduction

L'industrie du développement des applications mobiles est en croissement continue 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 1.52 milliard d'unités en 2019, le taux de croissance n'est pas stable cette année à cause des perturbations des marchés mondiaux induites par l'apparition de la pandémie (Gartner, 2020).

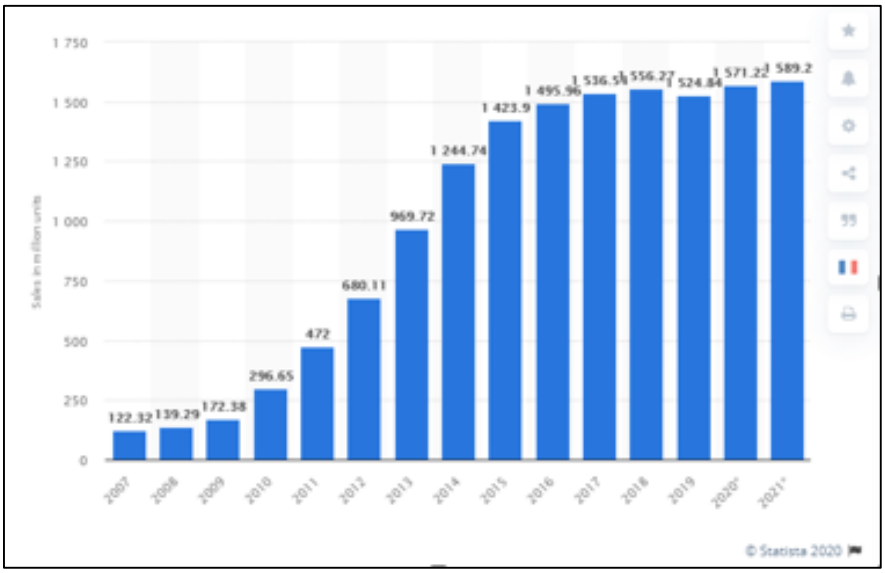

Figure 2 Les ventes des téléphones mobiles dans le monde pour les utilisateurs finaux

Cette évolution est due à la croissance du marché des smartphones, ce qui pousse les consommateurs à abandonner les téléphones classique de plus en plus (Gartner, 2015). La figure

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.

Le marché des tablettes et smartphones est dominés par Android (statCounter, 2020). Le choix d'Android est justifié par sa technologie constamment novatrice, ouverte et moins cher par rapport à iOS.

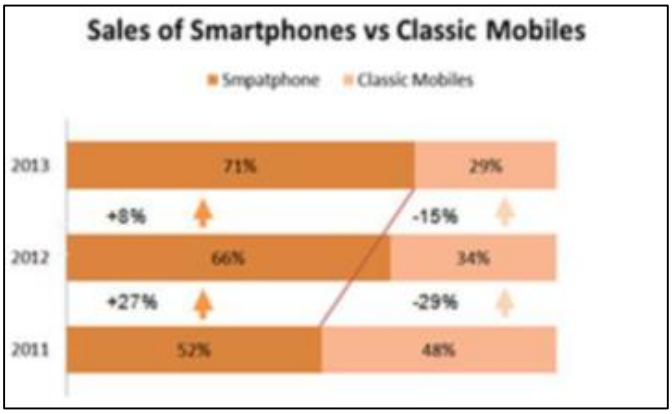

Figure 3 - Les ventes des Smartphones vs mobiles classiques (Kerensen Consulting, 2015)

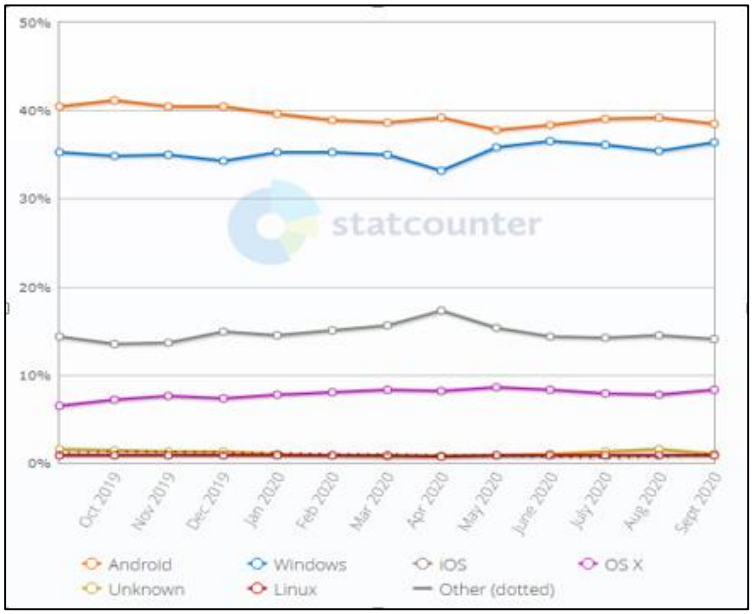

Figure 4 Part de marché du système d'exploitation mobile dans le monde (StatCounter, 2015)

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-dessous de près deux plateformes identifiées comme leaders du marché des smartphones selon ***StatCounter*** en 2020. 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 par le Startup Android rachetée par Google en Août 2005. Google a publié Android en novembre 2007, 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 (et kotlin depuis 2017) 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 (sqlite) et des fonctionnalités réseau avancées telles que la 3G, 4G, WLAN, etc. L'API est en constante évolution et la version actuelle (android 10) est une énorme augmentation par rapport au nombre de fonctionnalités disponibles à partir de la version 1.0. Depuis qu'Android est un système d'exploitation 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 portables, 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 :

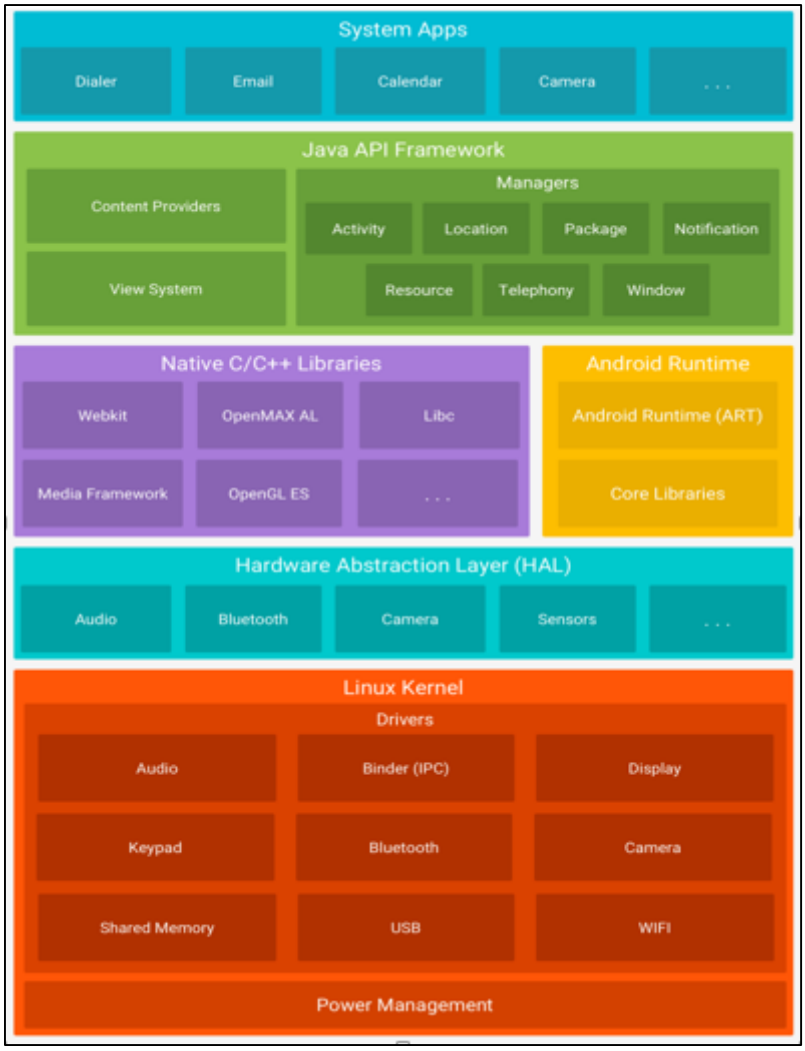

Figure 5 Architecture Android (Android, 2020)

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 6 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.

Il est à noter que depuis 2014, Apple a introduit un nouveau langage de programmation « Swift » officiellement supporté par Apple pour le développement de ces produits.

1.2.3. Comparaison des plateformes

Au niveau logiciel, les différentes OS (Android, iOS, 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 conséquence, elle pousse les développeurs à faire un choix sur la plateforme, tout en assurant la plus grande distribution possible.

OS	**Android**	**iOS**
Editer par	Google	Appel
Environnement de développement	Eclipse, Android Studio	XCode
Langages de programmation	JAVA/Kotlin	Objective-C, Swift
Interface graphique	XML	CocoaTouch
Fichier exécutable	.apk	.app
Applications sur	Google Play	Apple-iTunes
Machine virtuelle	Dalvik VM	Non
Développement sur	Multiplateforme	Mac OS X

Table 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 ait beaucoup de restrictions et des défis, le principal défi qui est abordé dans le présent mémoire 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.

1.4. Approches de développement mobiles

Plusieurs études sur les approches pour le développement des applications mobiles multiplateformes 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 :

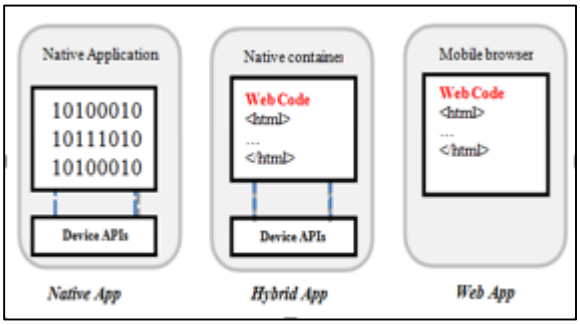

Figure 7 Méthode de développement des applications mobiles

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 embarqués et 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 et iOS, nous devons développer des applications distinctes avec des langages de programmation différents, comme Swift pour iOS ou Java pour Android (***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 (App Store).

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

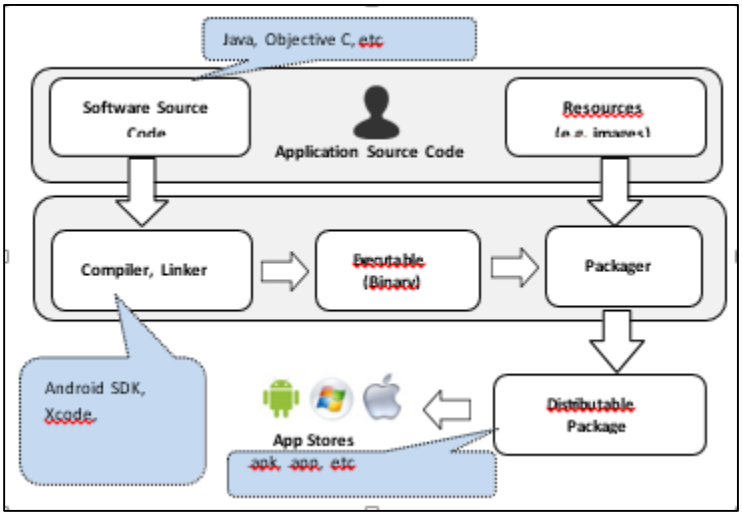

Figure 8 Architecture d'une application native

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

Avantages	inconvénients
• présence 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 convivial	• 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 d'entretien (***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***).

Table 2 Avantages et inconvénients de l'approche native

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 car elles 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:

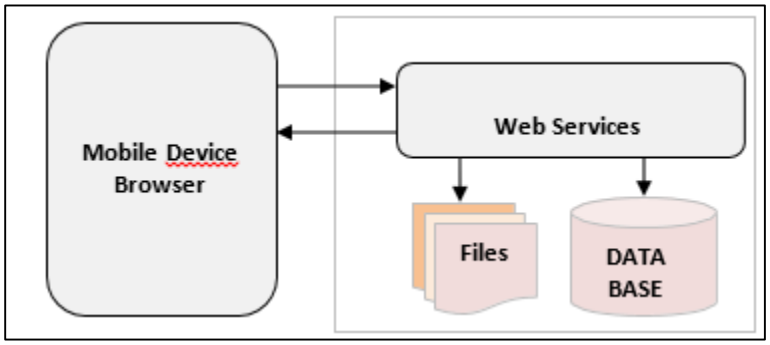

Figure 9 L'architecture logique d'une application web mobile (Raj et al, 2012)

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

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.

Table 3 avantages et inconveignatns de l'approche web

1.4.3. Approche hybride

L'approche de développement hybride, permet de créer des applications qui utilisent tout ou 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 smartphone. 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:

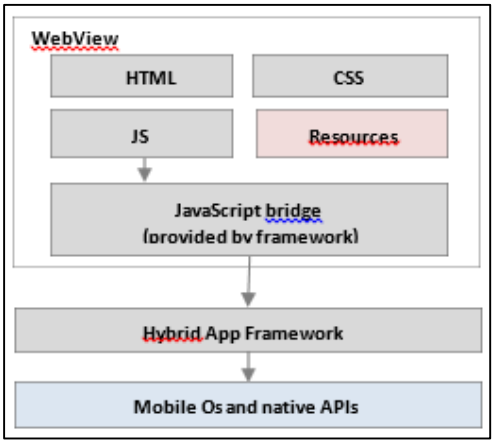

Figure 10 L'architecture logique d'une application hybride typique

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

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'applicationpeut 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.

Table 4 Avantages et inconvénients de l'approche hybride

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.

Aspect	Développement Web	Développement hybride	Développement natif
Facile à 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 téléphones...	Large	Limité à une plateforme mobile particulière
Cycle de vie du développement (génération/test/déploiement)	Court	Moyen	Long
Portabilité des applications vers d'autres plateformes	Elevée	Elevée	Aucun
Prise en charge des fonctionnalités du terminal natif	Certaines	La plupart	Toutes
Distribution avec mécanismes Intégrés	Non	Oui	Oui
Possibilité d'écrire dès 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 development framework (like PhoneGap, Ionic)	Objective-C/Swift, Java, C++, C#

Table 5 Comparaison des approches de développement des applications mobiles

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

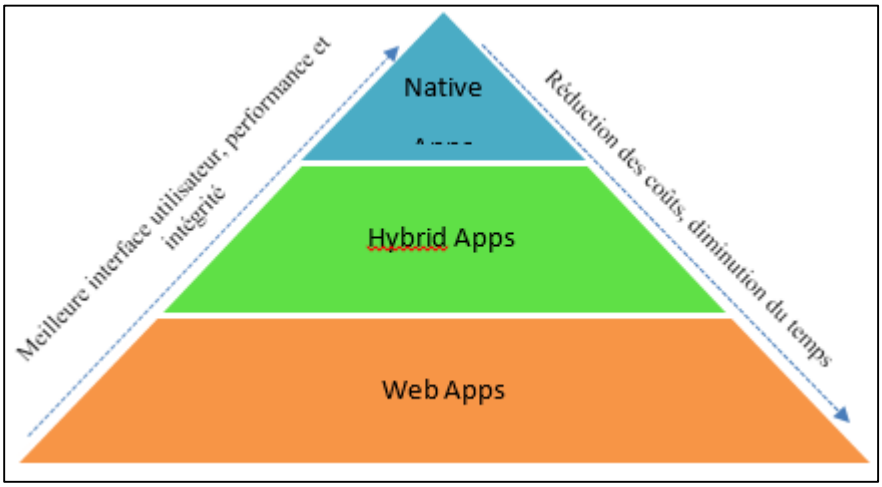

Figure 11 Approche native vs développement multiplateformes

1.5. Conclusion

Ce chapitre nous a permis de faire une analyse des différents types de plateformes de smartphones et des méthodes de développement d'application mobiles. La comparaison montre l'avantage et la qualité qui est dans les applications natives.

2. Ingénierie Dirigée par les Modèles (MDE : Model Driven Engineering)

2.1. Introduction

Dans ce chapitre nous présentons dans un premier temps l'architecture MDA. Ensuite, nous détaillions les notions qui sont à la base des principes généraux de l'ingénierie dirigée par les modèles (IDM), à savoir, la méta-modélisation d'une part et la transformation de modèles d'autre part. Nous sommes focalisés particulièrement sur les axes principaux de l'approche MDA, à savoir, les techniques et langages de modélisation et méta-modélisation, les techniques et langages de transformation de modèles.

2.2. Présentation de MDA (MDA: Model Driven Architecture) :

L'approche MDA est une démarche proposée par l'OMG (***OMG, 1989***) depuis 2001. Il s'agit d'une vision particulière du Développement Dirigé par les Modèles (*MDD: Model Driven Development*) (***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 (*Model Driven Engineering*) 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*). Les rôles de chacun de ces modèles sont les suivants:

2.2.1 Le CIM (Computation Independent Model) :

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) :

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) :

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 Frameworks les plus utilisés comme Struts, Spring, Hibernante, .Net, Java, etc.

2.2.4 Le PSM (Platform Specific Model):

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 transformations successives jusqu'au code dans un langage spécifique (e.g. JSF2, EJB3, Struts, etc.).

2.2.5. Exemples d'outils qui respectent approche MDA

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 relations entre elles sont présentées dans la Figure ci-dessous.

Figure 12 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 «représentationDu». 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 par Minsky (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».

Etant 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 «conformeA». Il s'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-modèle).

Pour assurer une cohérence des niveaux d'abstraction, l'OMG a défini une architecture de modélisation sur quatre niveaux (*voir la figure ci-dessous pour plus de détails*).

- *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 (*Ingénierie Des Modèles*) 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.

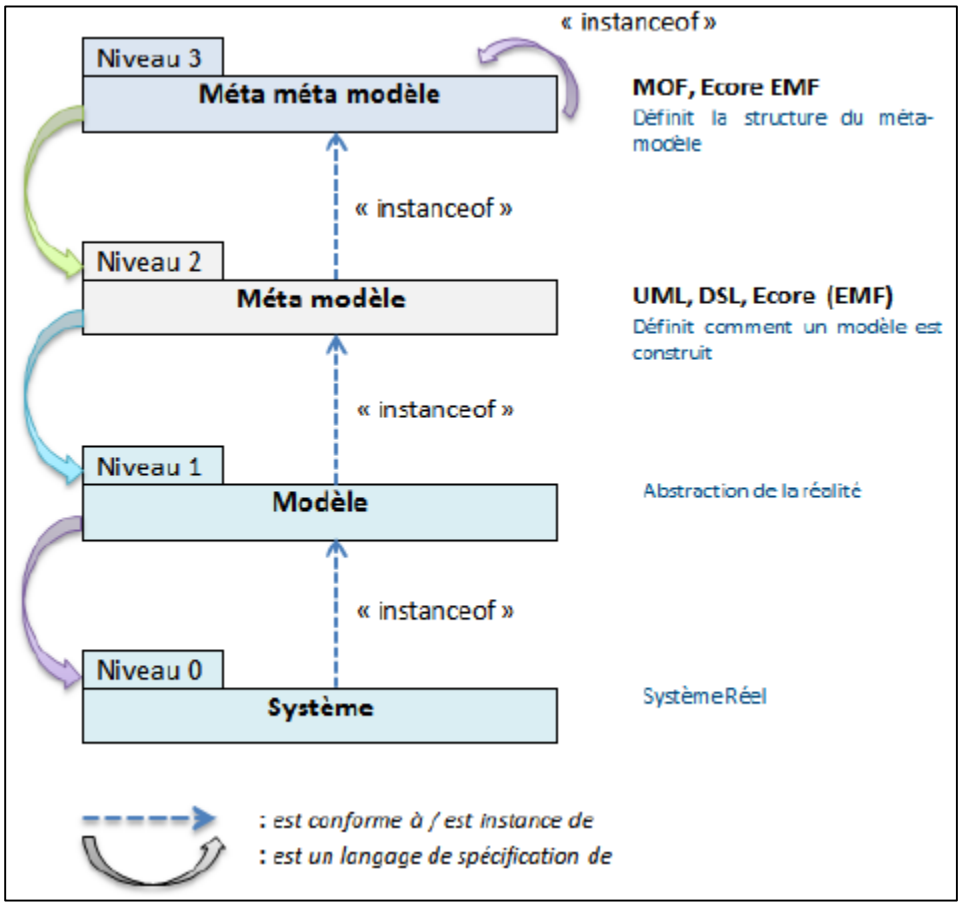

Figure 13 Architecture à 4 niveaux d'abstraction

2.4. 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 MOF 2.0.

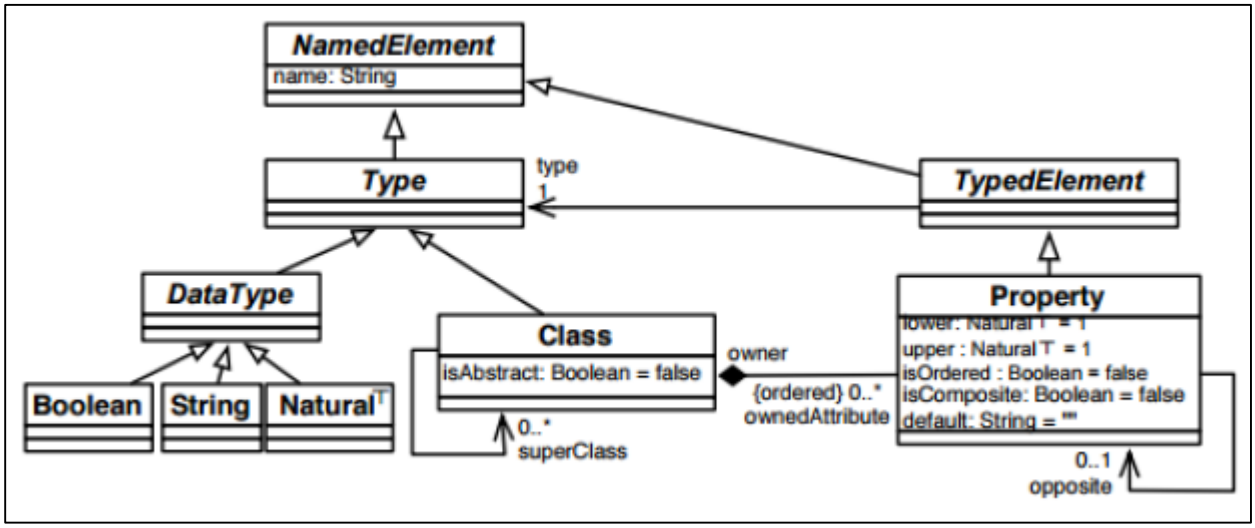

Figure 14 Représentation de MOF 2.0 sous forme de diagramme de classes

- 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.

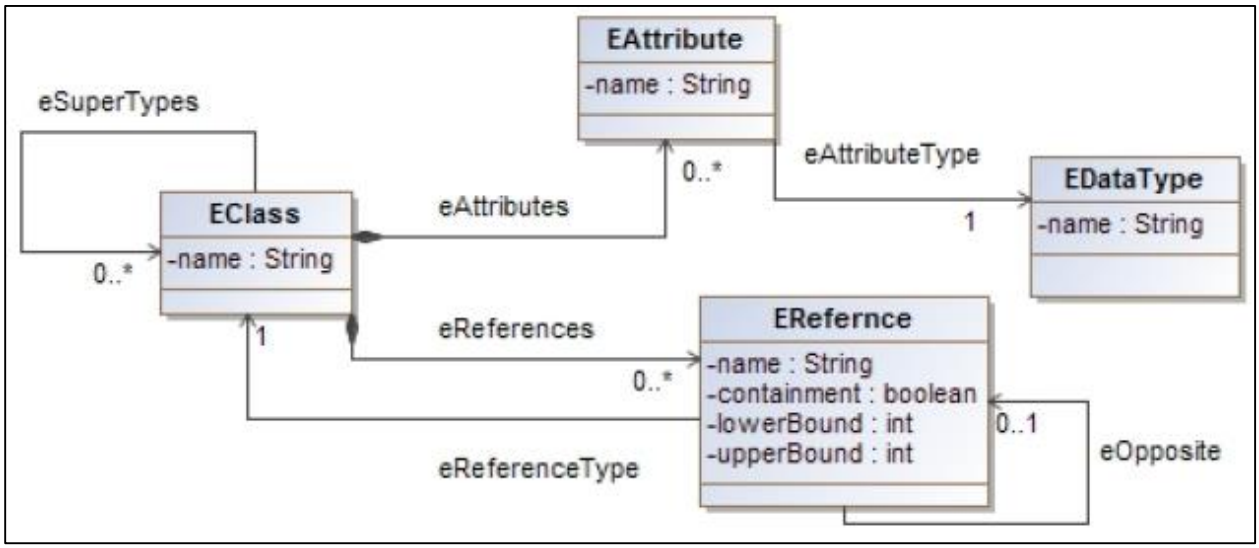

Figure 15 Extrait du Méta-Modèle Ecore

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

2.5. 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.

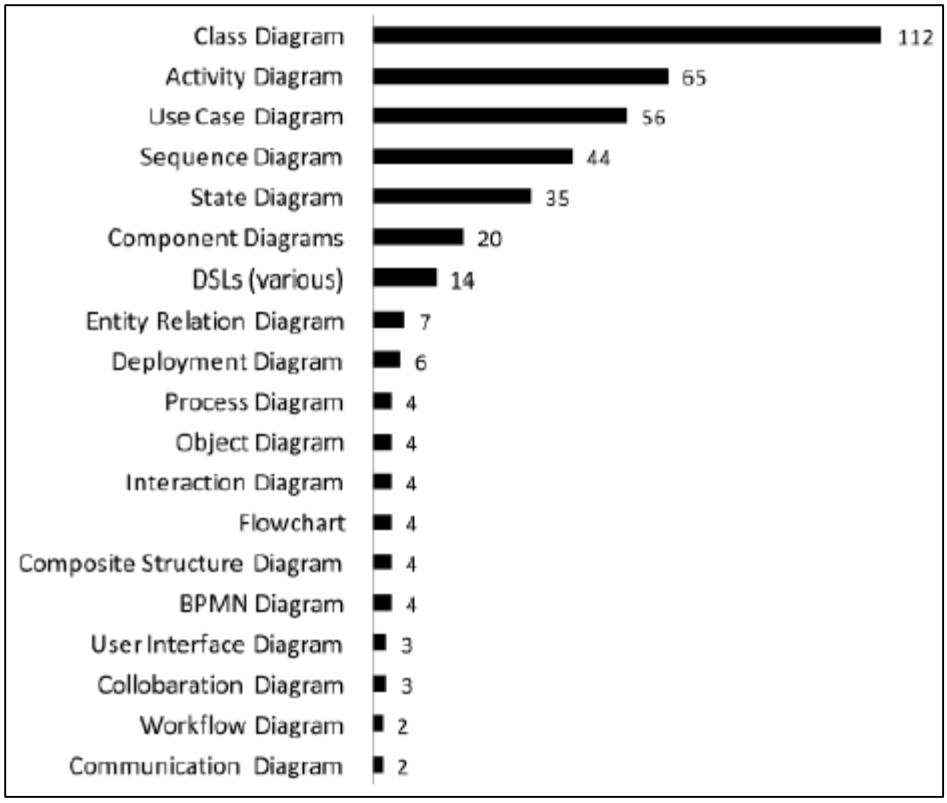

Figure 16 Liste des diagrammes utilisés régulièrement en UML (Hutchinson et al, 2014)

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 décrit(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.

2.6. 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. Outils pour la définition des DSLs autonomes

2.7.1. Xtext

Xtext[3] est le pilier pour la création de DSL textuel externe, c'est une solution d'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.

[3] http://www.eclipse.org/Xtext/

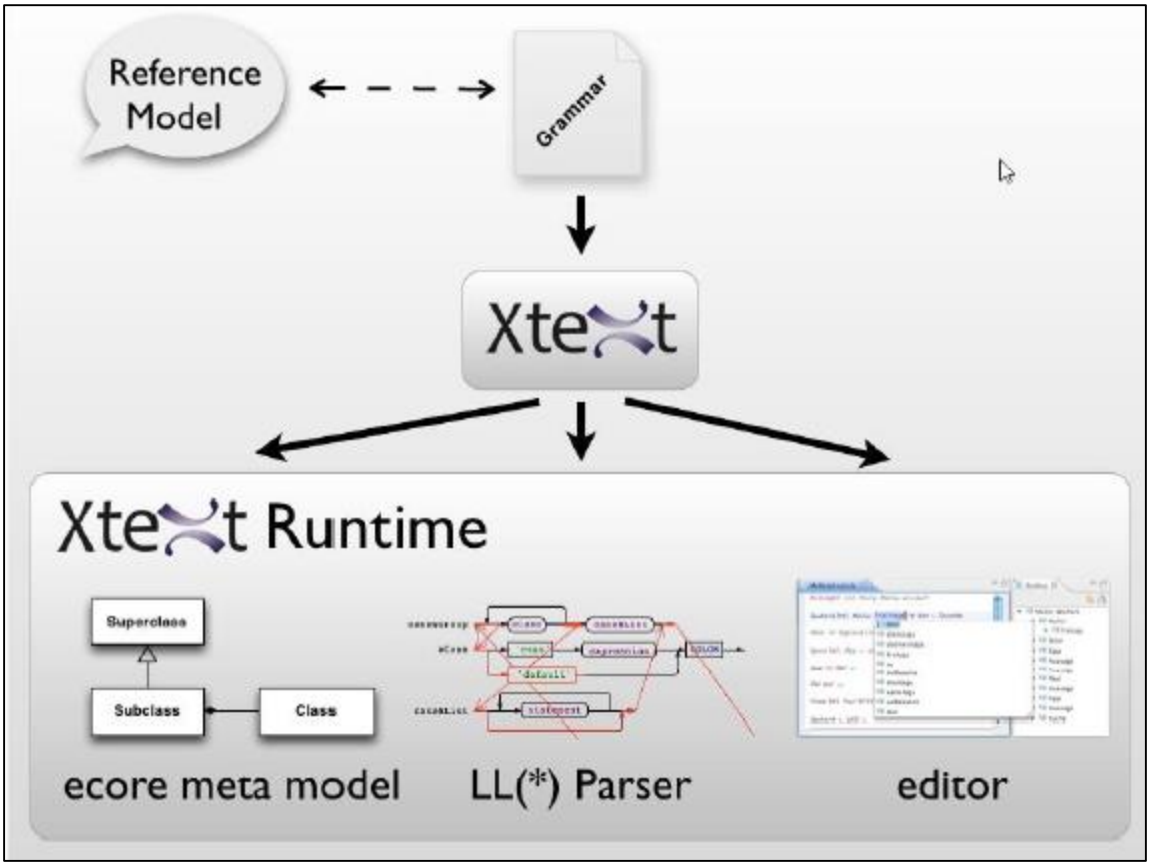

Figure 17 Vue d'ensemble de l'outil Xtext

2.7.2. Spoofax

Spoofax[4] est aussi basé sur Eclipse, mais ne s'appuie pas sur EMF. C'est un système développé par TU Delft[5] et 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.

2.7.3. JetbrainsMPS

JetBrains MPS[6] est différent de Xtext et Spoofax. Il n'utilise pas de texte brut pour l'édition et l'analyse. Par contre, il utilise une approche projectionnelle, 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

[4] http://strategoxt.org/Spoofax
[5] http://www.tudelft.nl/
[6] http://www.jetbrains.com/mps/

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.

2.8. 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.

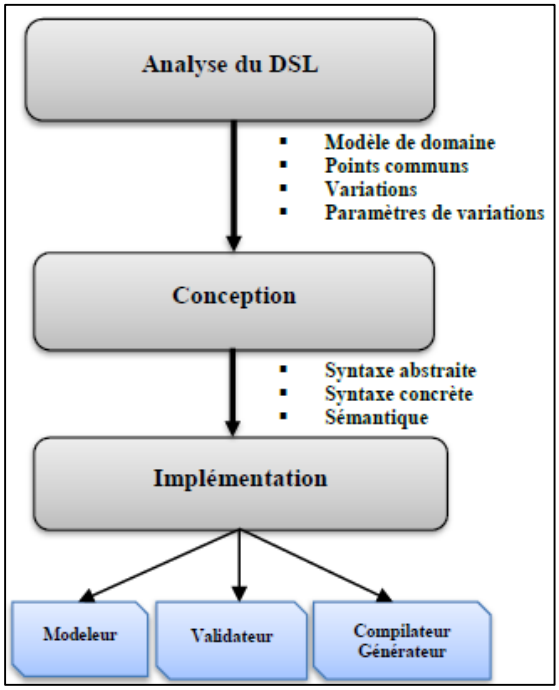

Figure 18 Processus de développement des DSL

2.9. Les transformations MDA

La communauté IDM (Bézivin, 2004) [3] consiste à dire qu'une transformation de modèles est la génération d'un ou de plusieurs modèles cibles à partir d'un ou de plusieurs modèles sources. L'un des objectifs d l'IDM est que l'IDM consiste à pouvoir rendre opérationnels les modèles à l'aide de transformations. Donc la transformation est le cœur de l'IDM. Cette notion est bien représentée dans l'approche de MDA.

Les transformations possibles entre ces différents types de modèles sont comme suit :

- 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 PIM -> PSM et PSM -> code. 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.

La figure qui suit illustre ces transformations :

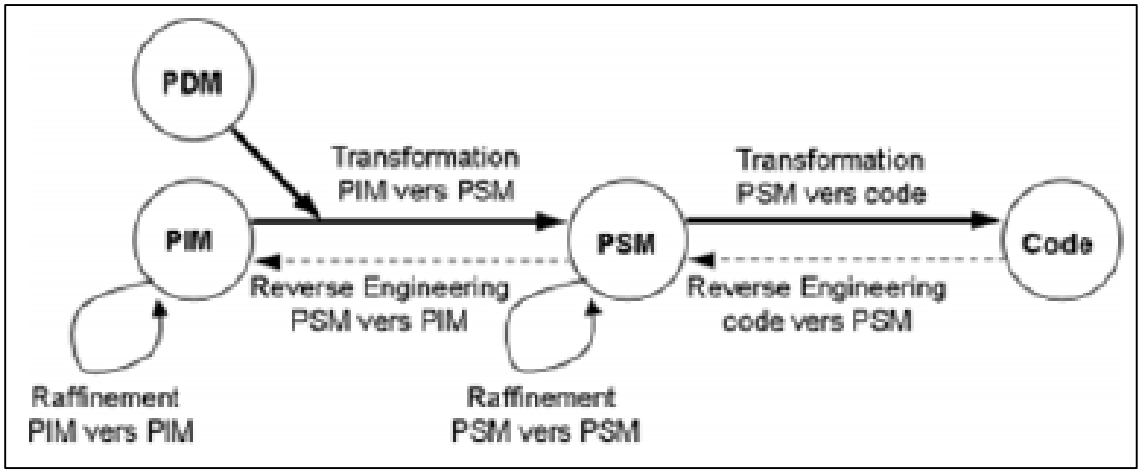

Figure 19 Les modèles et transformations dans l'approche MDA

2.10. Conclusion

En guise de conclusion, le développement des systèmes d'information dirigés par les modèles a pour principal objectif de concevoir des applications en séparant les préoccupations fonctionnelles de leurs préoccupations techniques et en plaçant les notions de modèles, méta-modèles et transformations de modèles au centre du processus de développement.

Nous avons abordé dans ce chapitre les différentes notions qui sont à la base des principes généraux de l'IDM, à savoir la méta-modélisation et la transformation des modèles. Nous avons également présenté les techniques et langages de modélisation, méta-modélisation ainsi que la transformation des modèles.

3. Solution proposée

3.1. Introduction

Dans cette partie du rapport représente la solution proposée pour l'implémentation de l'approche d'ingénierie dirigée par les modèles à partir de l'étude théorique. On présente des techniques et des outils utilisés.

3.2. Présentation de la solution

On va parler plus précisément sur la solution proposée. On peut l'expliquer étape par étape comme suit :

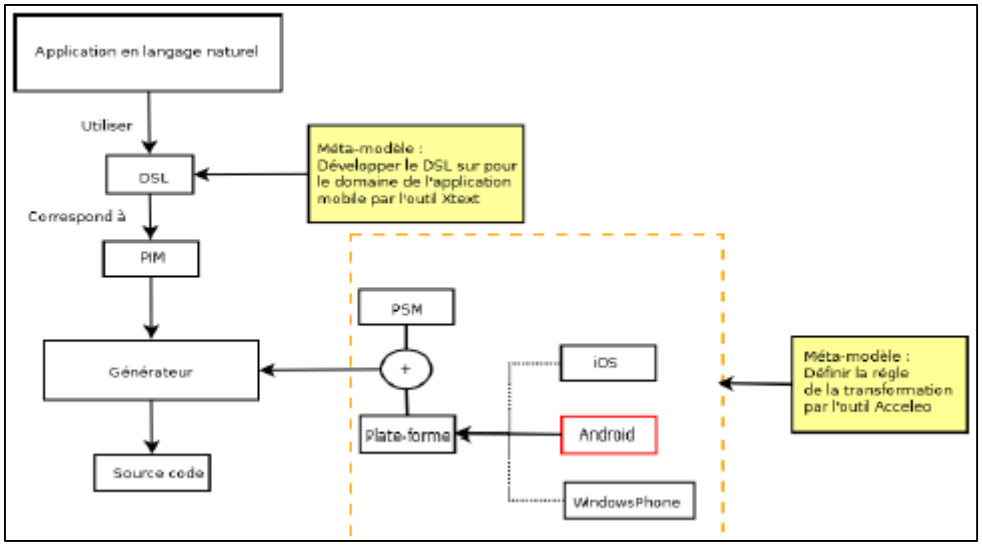

Figure 20 La vue l'ensemble de solution

Au niveau de M3, le méta-modèle utilisé est suivi le standard MOF.

On suppose que l'application de langage naturel est le CIM.

On doit transformer le CIM vers le PIM par l'aide d'un langage DSL.

On implémente la technologie du projet Eclipse, distribuée dans Eclipse Modeling Framework : Xtext 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 a un langage d'usage général comme Java, Objective C, C + +, etc.

On implémente le Xtext pour définir le langage de méta-modèle (MobileApp-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[7] qui est une implémentation de l'Object Management Group (OMG) sur la norme de MOF, MOF Models to Text (MOFM2T)[8].

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.

3.3. Méta-modélisation

Un modèle est une représentation abstraite de la réalité. Il est utilisé pour représenter 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 vise 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 décrites de l'approche MDA.

Les méta-modèles dédies peuvent donc être crées pour répondre aux besoins spécifiques.

Ils sont appelés DSL (Domain Specific Language). Comme DSL sont plus largement utilises, le développement de nouvelles méta-modèles devrait devenir mieux comprise et plus fréquente. Pour développer un méta-modèle, les étapes suivantes sont suivies :

[7] http://www.acceleo.org/.
[8] http://www.omg.org/mof/.

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.

3.3.1. 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 figure représenté l'idée du processus de la transformation le CIM vers le PIM.

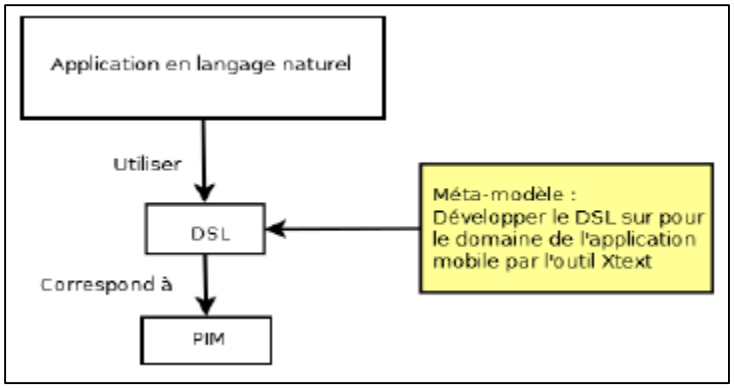

Figure 21 L'étape de réalisation de PIM

Pour le développement du DSL, on utilise Eclipse Xtext. Le Xtext fournit l'outil pour créer et utiliser le DSL textuelles dans un environnement de Model Driven Developement (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.

3.4. 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 Langage (M2T). M2T se concentre sur la génération le codage textuelle à partir de modèle. On a choisi l'Acceleo comme l'outil de développement.

3.4.1 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 basé 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.

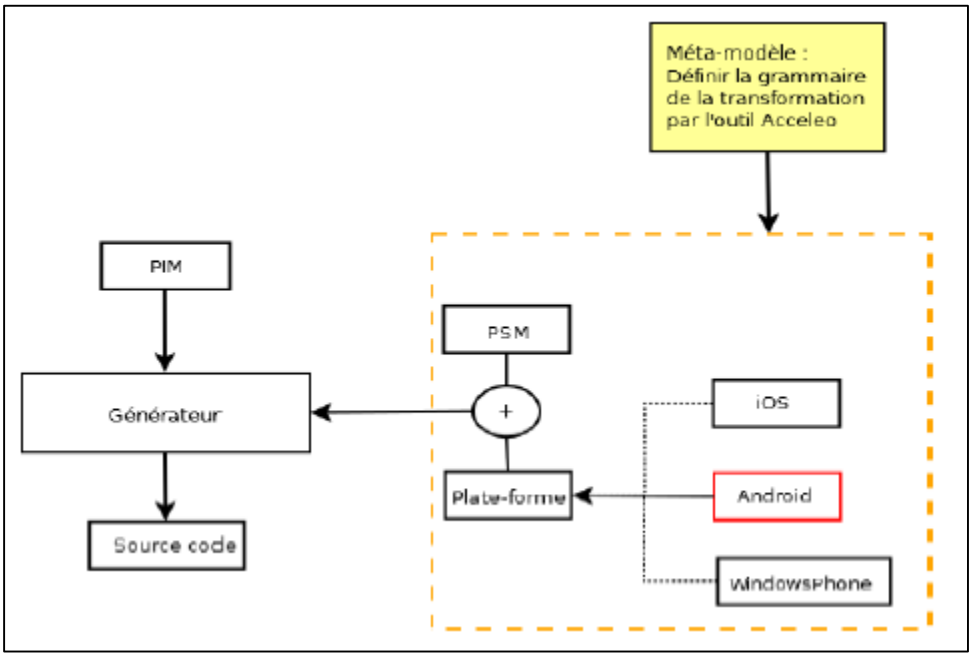

Figure 22 L'étape de réalisation de générateur

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.

3.4.2 Plateforme cible (Android)

La plateforme Android[9] 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-répertoires 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.

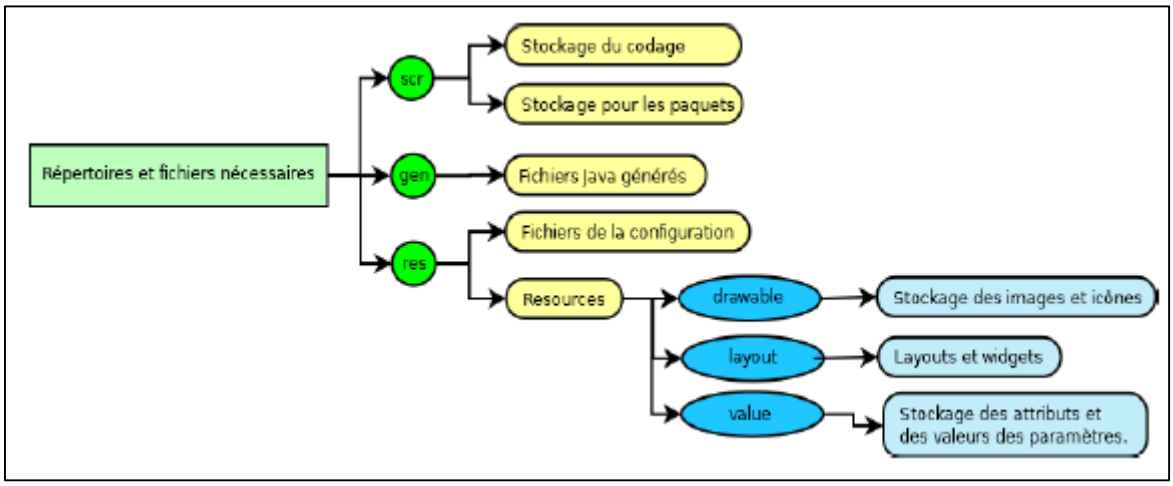

Figure 23 Les répertoires et fichiers nécessaires de l'application Android

[9] http://developer.android.com/guide/index.html.

La figure représente 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. Il se 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, le 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 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 modification.

3.5. Conclusion

Ce chapitre présente une analyse de la solution du développement des applications mobiles à travers les modèles. Cette approche est censée résoudre la problématique posée. Pour tester la solution, une implémentera est proposée et exposée dans ce qui suit.

4. Réalisation

4.1. Introduction

Ce chapitre est la réalisation du travail. C'est l'implémentation de la solution proposée et l'expérimentation pour examiner le produit de projet.

Le travail se compose de deux parties. La première, c'est de modéliser le modèle indépendant des plateformes (PIM). Le modèle obtenu de cette partie joue le rôle de 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. La deuxième est la partie du développement de générateur de code. Le choix portera sur un générateur de code pour une application Android. Afin de réaliser les générateurs de code, on suit le principe du modèle spécification des plateformes (PSM).

4.2. Implémentation du DSL

Le DSL est développé sous la conception de regrouper des éléments en trois composants : View, Ressources (model) et Screen (controller). Cela permet de réaliser la transformation de code pour les plateformes spécifiées. Quand la structure est bien conçue, la transformation du code ce fait facilement. Il soutient également la modification de modèle en cas de besoin. La Figure est la structure du DSL.

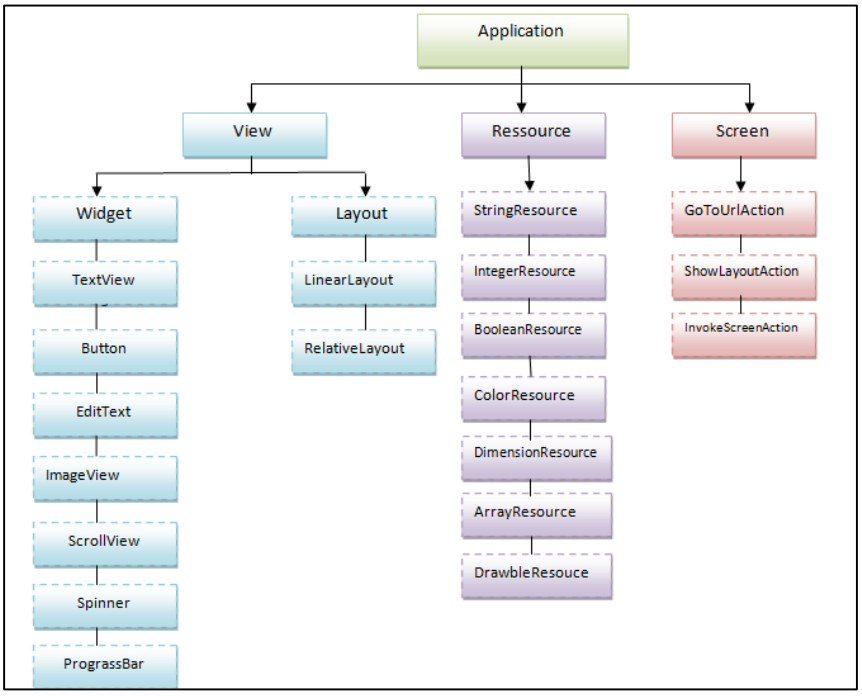

Figure 24 Le concept du DSL

4.2.1. Application

"Application " C'est la racine de la hiérarchie, elle contient les composants principaux et les informations nécessaires sur une application tel que le package, version Android, version sdk et les uses permissions.

```
grammar org.xtext.example.droid.Droid with org.eclipse.xtext.common.Terminals

generate droid "http://www.xtext.org/example/droid/Droid"

Application:
    'application' name=ID
        '=>' packageName=PackageName
         (
            ('version:' versionCode=INT '=>' versionName=STRING)?
          & (sdkVersion=ApplicationUsesSDK)?&(useper=UsesPermission)*
         )
       (screens+=Screen | VIEW+=View| resources+=Resource)+
;

UsesPermission:
"android.permisssion."appel=ID
;

ApplicationUsesSDK:
      'sdk:'
    '{'
    (
        ('min:' minSdkVersion=INT ';')?
      & ('max:' maxSdkVersion=INT ';')?
      & ('target:' targetSdkVersion=INT ';')?
    )
    '}'
;
```

Figure 25 Grammaire Xtext pour le concept Application

4.2.2 Ressource (Model)

Représente les valeurs qui peuvent être assignées aux widgets et layouts propriétés. Le type des attributs peut-être: String, Integer, Float, Boolen ou Array.

```
Resource:
 StringResource | IntegerResource | BooleanResource
     | ColorResource | DimensionResource
     | ArrayResource | DrawableResource
;
```

Figure 26 Grammaire Xtext du concept Ressource

```
DrawableResource:
    (BitmapDrawableResource | TransitionDrawableResource)
;

BitmapDrawableResource:
     name=ID '=' filename=ID
;

TransitionDrawableResource:
     name=ID
        from=[BitmapDrawableResource] '<->' to=[BitmapDrawableResource]
;

ArrayResource:
    IntegerArrayResource | StringArrayResource
;

StringArrayResource:
     name=ID '=' '['
        (items+=STRING (',' items+=STRING)* )
    ']'
;

IntegerArrayResource:
     name=ID '=' '['
        (items+=INT (',' items+=INT)* )
    ']'
;

DimensionResource:
    name=ID '=' value=DimensionValue
;
```

Figure 27 Grammaire Xtext des attributs du concept Ressource

4.2.3 View

La partie view représente l'interface utilisateur de l'application. Le langage fournit deux types de vue le contenu (widgets) et le contenant (layout).

```
View:

    Widget | Layout

;
```

Figure 28 Grammaire Xtext du concept View

Widgets: Les éléments widgets tels que les labels, field, les boutons sont regroupés et disposés à l'intérieur des éléments de conteneur (layout).

```
Widget:
     TextView | Button | ImageView | EditText | Spinner |ScrollView|PrograssBar
;
```

Figure 29 Grammaire Xtext du concept Widget

4.2.4 Screen

Représente un écran d'application et une interface utilisateur affichée.

```
Screen:
    'screen' name=ID
    '{'
    (
        ('show' layout=[Layout])
      | widgets=ViewCollection
    )
    '}'
;
```

Figure 30 Grammaire Xtext du concept Screen

Action: Sert à définir les actions de l'application. Une classe Screen peut contenir plusieurs actions, chaque action est décrite en forme de ncode Fragment z Nous pouvons définir les actions par des événements tels que onCreate, onClick, onTouch, onDéfined.

- InvokeActiviyAction : cette action fait l'appel d'autres actions qui sont définies. La définition de InvokeActiviyAction commence par le mot-clé invoke suivi par le nom d'action dont nous avons envie d'appeler ;
- GoToURLAction : cette action est utilisée lorsque l'application se compose d'une fonction qui permet l'accès internet. La définition de GoToURLAction commence par le mot-clé goto suivi par l'url du site-web destinataire ;
- ShowScreen : C'est l'action qui soutient la fonction de l'affichage de l'interface. Sa définition commence avec le mot-clé show suivi par le nom de l'interface.

```
Action:
    (GoToURLAction | ShowLayoutAction | InvokeScreenAction)
;

GoToURLAction:
    'goTo' url=STRING
;

ShowLayoutAction:
    'show' layout=[Layout]
;

InvokeScreenAction:
    'invoke' activity=[Screen]
;

ButtonTarget returns InvokeScreenAction:
    'to' screen=[Screen]
;
```

Figure 31 Grammaire Xtext du concept Action

4.3. Implémentation de générateur de code

Une fois que le modèle PIM qui est compté comme l'entrée de génération est modélisée, Il est très important de définir les règles de transformation car elles sont la clé de génération du code de la plateforme cible à partir de modèle PIM.

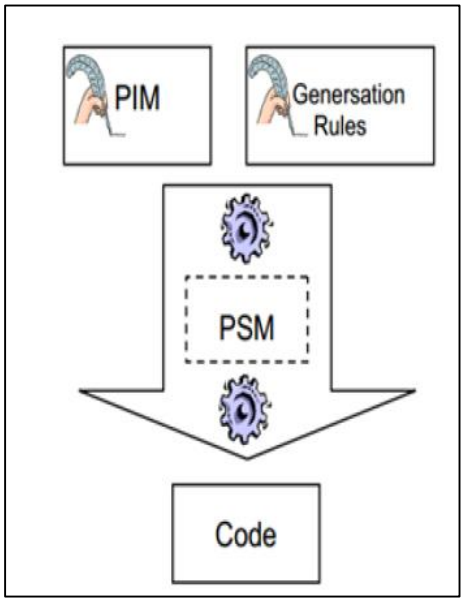

Figure 32 Les transformation de l'approche MDA

Afin de développer le générateur de code, nous pouvons exploiter l'Acceleo qui est un générateur de code source de la fondation Eclipse permettant de mettre en œuvre l'approche MDA (Model

Driven Architecture) pour réaliser des applications. Notre génération de code Android est basée sur le langage DSL, qui est utilisé par Acceleo pour générer la structure d'application. La relation entre les interfaces telle que l'association et l'héritage, sont respectés lors de la génération de code. Lorsque le code PIM représente une composante de l'API Android, la génération comprend éventuellement les importations selon la déclaration et les paramètres d'attributs.

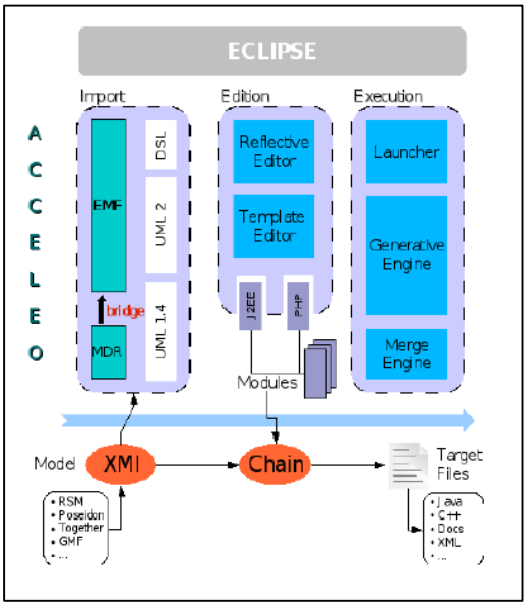

Figure 33 L'architecture de acceleo

4.3.1 Le module generateManifest

Le module generateManifest sert à générer le fichier AndroidManifest.xml par sa définition dans le template "generateManifestFile" qui va tirer des informations de "application" qui est définit dans la partie de l'application dans le modèle PIM.

Ici un code source qui sert d'exemple pour générer le fichier AndroidManifest.xml. La génération ce fait sur deux étapes :

En premier lieu on délègue la génération du fichier manifest au template main. Ca ce fait comme suit :

```
[comment encoding = UTF-8 /]
[module main('http://www.xtext.org/example/droid/Droid')]
[import org::acceleo::example::droid::files::application /]

[template public main(aApplication : Application)]
[comment @main/]
        [aApplication.generateManifestFile()/]
[/template]
```

Figure 34 Code pour délèguer la génération du fichier manifest au template main

Ensuite on génère AndroidManifest.xml par le code qui suit :

```
[comment encoding = UTF-8 /]
[module application('http://www.xtext.org/example/droid/Droid')/]

[template public generateManifestFile(aApplication : Application)]
       [file ('AndroidManifest.xml', false, 'UTF-8')]
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="[aApplication.packageName/]"
[if (aApplication.versionCode > 0)]
      android:versionCode="[aApplication.versionCode/]"
[else]
      android:versionCode="1"
[/if]
[if (aApplication.versionName <> null)]
      android:versionName="[aApplication.versionName/]"
[else]
      android:versionName="1.0"
[/if]
>

[if (aApplication.sdkVersion <> null)]
              <uses-sdk
       [if (aApplication.sdkVersion.minSdkVersion <> null)]
                      android:minSdkVersion="[aApplication.sdkVersion.minSdkVersion/]"
       [else]
                      android:minSdkVersion="10"
       [/if]

       [if (aApplication.sdkVersion.maxSdkVersion <> null)]
                      android:maxSdkVersion="[aApplication.sdkVersion.maxSdkVersion/]"
       [/if]

       [if (aApplication.sdkVersion.targetSdkVersion <> null)]
                      android:targetSdkVersion="[aApplication.sdkVersion.targetSdkVersion/]"
       [/if]

              />
[else]
    <uses-sdk android:minSdkVersion="10" />
[/if]

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name="[aApplication.packageName/].[aApplication.screens->first().name/]Activity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
</manifest>
[/file]
[/template]
```

Figure 35 Code pour générer AndroidManifest.xml

4.3.2 Le module generateView

Le module generateView sert à générer le répertoire layout et les fichiers qui se trouvent dans ce répertoire. Le concept de génération est de générer le fichier .xml qui se compose des layouts et des widgets. Il va faire l'appel au template generateLayout (qui se trouve dans le fichier Layout.mtl) pour générer les propriétés de layout qui sont définies à partir du DSL, et fait l'appel au template generateWidget (qui se trouve dans le fichier Widget.mtl) pour générer le xml de la propriété des widgets. Au cas où on a plusieurs widgets qui composent un layout, on fait appel au template "for".

4.3.3 Le module generateResource

Le module generateResource sert à générer le répertoire « value » et les fichiers de ce répertoire. Les valeurs de ressource qui sont définies dans la partie de Ressource de modèle PIM seront transformées en forme de fichier.xml tel que ('res/values/gen-app-strings.xml').

4.3.4 Le module generateScreen

Le module generateScreen sert à générer le répertoire « src » et les fichiers.java. Une classe dans le modèle PIM définie par "screen" sera transformée en forme de fichier.java. Les actions seront générées à partir de l'appel de template "GenerateAction".

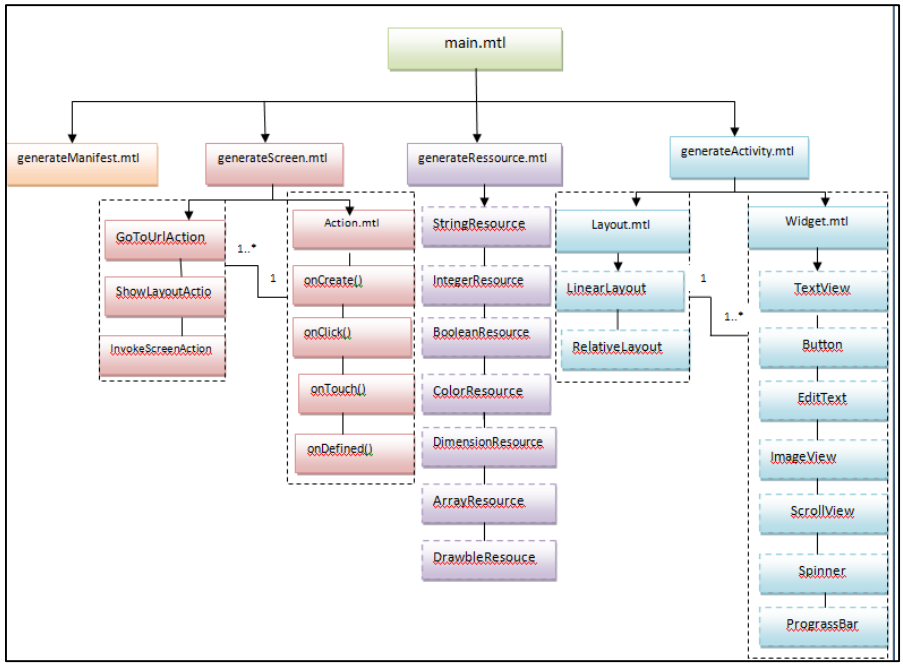

Figure 36 Le concept des transformations pour la plateforme Android

4.4. Application de la méthode proposée

Dans cette étude, on réalise un modèles PIM à partir du DSL par but d'expérimentation. Un modèle PIM est le modèle d'une simple application mobile. Ce modèle joue le rôle de modèle d'entré 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 le DSL, on essaie de réaliser un modèle PIM de l'application mobile dans le projet Acceleo pour Android. Les étapes qui vont être présentées, sont le processus de développement d'une application mobile en utilisant le générateur de code précédemment introduit. On procède comme suit :

- Sur le projet Acceleo, faites un clic à droit sur le dossier de model.
- Choisissez Nouveau Fichier.
- Créez un fichier avec l'extension « .mda ».
- 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, faites un clic à droit sur le projet.
- Choisissez Run configuration.
- Complétez la configuration de l'exécution.
- Cliquez Run, vous obtenez le résultat dans le répertoire « out ».

4.4.2. Exemple d'une application mobile simple

L'application est composée de deux classes, main-activity et second-screen. L'interface résultante est représentée dans la Figure (35), et le modèle PIM responsable de transformation est représenté dans la Figure (36).

4.4.2.1. Description de l'application

La description est la suivante:

- Classe main-activity
    - "Textview" pour afficher un commentaire dans la première intent ;
    - "Boutton Next" qui permet l'utilisateur de passer au deuxième intent.

- Classe second-screen
    - "Boutton Back" qui peut amener l'utilisateur vers la première intent ;
    - "Boutton Url" qui permet l'utilisateur de visiter le site web "http://developer.android.com".

4.4.2.2. Résultat

Le résultat de l'application est comme suit

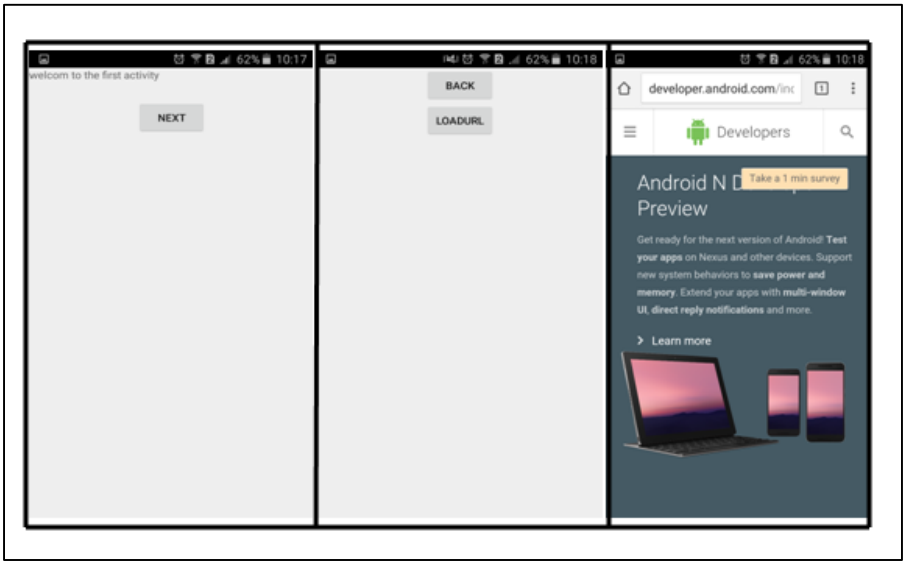

Figure 37 Interface de la première application

```
application HelloDroid => com.example.appsimple
version:1 => "1.0"
sdk:{
    min:7;
    max:14;
}
hello = "Hello Droid!"

screen Main {
    show main
}

layout main {
    # textView text_view string(hello) {
        layout: {
            width: match_parent;
            height: wrap_content;

        }
        width: 50px;
    }

    # button next "Next" {
        onClick:  invoke Second;
    }
}

screen Second {
    # layout second {
        # button back "Back" {
            onClick:  invoke Main;
        }

        # button url "LoadURL" {
            onClick: goTo "http://developer.android.com";
        }
    }
}
```

Figure 38 Modèle PIM de la première application

4.5. Conclusion

À la fin de la réalisation, nous pouvons analyser les résultats obtenus:

- Le générateur de code a donné le code qui peut bien fonctionner pour tous les éléments testés ;
- Le code en forme.xml est bien structuré comme le code qui est réalisé manuellement ;
- Le code en forme.java fait marcher les actions d'application exactement comme le code réalisé manuellement.

Conclusion générale

Nous avions comme objectifs d'utiliser l'ingénierie dirigée par les modèles afin de générer du code source d'applications mobiles dans une forme natif. Nous avons pu illustrer avec succès la faisabilité et le mécanisme avec un exemple de réalisation qui a ciblé la plateforme Android.

Comme perspectives on peut élargir la portée de notre langage pour qu'il prenne en compte encore plus d'attributs de la plateforme Android et ainsi permettre des créé des applications plus complexes. Aussi permettre la génération de code pour d'autres plateformes dominantes comme IOS de Apple.

Références

Baixing, Q., Chen, T., Dai, H., Peng, B., & Wu, M. (2012, June). A Cross-platform Application Development Environment Supported by Cloud Service. In High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on (pp. 1421-1427). IEEE.

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. In Automated Software Engineering, 2001. (ASE 2001). Proceedings. 16th Annual International Conference on (pp. 273-280). IEEE.

Bézivin, J., & Briot, J. P. (2004). Sur les principes de base de l'ingénierie des modèles. L'OBJET, 10(4), 145-157.

Bézivin, J. (2005). On the unification power of models. Software & Systems Modeling, 4(2), 171-188.

Bézivin, J. (2009, September). Advances in Model Driven Engineering. In JISBD (p. 3).

Biap, P., Xiao, K., & Luo, L. (2010, June). Component-based mobile web application of cross-platform. In Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on (pp. 2072-2077). IEEE.

Bettini, L. (2016). Implementing domain-specific languages with Xtext and Xtend. Packt Publishing Ltd.

Blanc, X., & Salvatori, O. (2011). MDA en action: Ingénierie logicielle guidée par les modèles. Editions Eyrolles.

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.

Budinsky, F., Brodsky, S., & Merks, E. (2003). Eclipse Modeling Framework. Retrieved from https://eclipse.org/modeling/emf/.

Bup-Ki, M., Minhyuk, K., Yongjin, S., Seunghak, K., & Hyeon S. K. (2011, November). A UML metamodel for smart device application modeling based on Windows Phone 7 platform. In TENCON 2011-2011 IEEE Region 10 Conference (pp. 201-205). IEEE.

Charland, A., & Leroux, B. (2011). Mobile application development: web vs. native. Communications of the ACM, vol. 54, no 5, pp. 49-5.

Costanich, B. (2011). Developing C# Apps for IPhone and IPad Using MonoTouch. Springer Fachmedien.

Ehringer, D. (2010). The dalvik virtual machine architecture. Techn. report (March 2010), 4, 8.

El-Kassas, W. S., Abdullah, B. A., Yousef, A. H., & Wahba, A. M. (2015). Taxonomy of Cross-Platform Mobile Applications Development Approaches. Ain Shams Engineering Journal.

Fowler, M. (2010). Domain-Specific Languages Addison-Wesley Professional. Boston, MA, USA.

Granter (February, 2016). Gartner Says Worldwide Smartphone Sales Grew 9.7 Percent in Fourth Quarter of 2015. Retrieved from http://www.gartner.com/newsroom/id/3215217 (Accessed on February 12, 2017).

Gartner (March, 2015). Gartner Says Smartphone Sales Surpassed One Billion Units in 2014. Retrieved from

http://www.gartner.com/newsroom/id/2996817 (Accessed on December 3, 2015).
Groenewegen, D. M., & Visser, E. (2013). Integration of data validation and user interface concerns in a DSL for web applications. Software & Systems Modeling, 12(1), 35-52.
Gronback, R. C. (2009). Eclipse modeling project: a domain-specific language (DSL) toolkit. Pearson Education.
Herndon, R. M., & Berzins, V. A. (1988). The realizable benefits of a language prototyping language. IEEE Transactions 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 Computer Programming, 89, 144-161.
Liu, C., Zhu, Q., Holroyd, K. A., & Seng, E. K. (2011). Status and trends of mobile-health applications for iOS devices: A developer's perspective. Journal of Systems and Software, 84(11), 2022-2033.
Mellor, S. J., Clark, T., & Futagami, T. (2003). Model-driven development: guest editors' introduction. IEEE software, 20(5), 14-18.
Minsky, M. (1965). Matter, mind and models. International Federation for Information Processing (IFIP), 1, 45-50.
Mohamed LACHGAR. (2017). Approche mda pour automatiser la generation de code natif pour les applications mobil multiplateformes
Object Management Group (1989). Retrieved from http://www.omg.org.
OMG (2002). Meta Object Facility (MOF) Specification, Version 1.4, April 2002.
Palmier, M., Sing, I., & Cicchetti, A. (2012.) Comparison of cross-platform mobile development tools. Proceedings of the 16th International Conference on Intelligence in Next Generation Networks (ICIN), IEEE Xplore Press, Berlin, pp. 179-186, DOI:10.1109/ICIN.2012.6376023.
Perchat, J., Desertot, M., & Lecomte, S. (2013). Component based framework to create mobile cross-platform applications. Procedia Computer Science, 19, (pp. 1004-1011).
Raj, C. R., & Tolety, S. B. (2012, December). A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In India Conference (INDICON), 2012 Annual IEEE (pp. 625-629). IEEE.
Sambasivan, D., John, N., Udayakumar, S., & Gupta, R. (2011, November). Generic framework for mobile application development. In Internet (AH-ICI), 2011 Second Asian Himalayas International Conference on (pp. 1-5). IEEE.
Selic, B. (2007, May). A systematic approach to domain-specific language design using UML. In Object and Component-Oriented Real-Time Distributed Computing, 2007. ISORC'07. 10th IEEE International Symposium on (pp. 2-9). IEEE.
Serrano, N., Hernantes, J., & Gallardo, G. (2013). Mobile Web Apps. IEEE Software. pp: 22 – 27, DOI:10.1109/MS.2013.111.
Tracy, K. W. (2012). Mobile application development experiences on Apple's iOS and Android OS. Ieee Potentials, 31(4), 30-34.
Vallecillo, A. (2010, June). On the combination of domain specific modeling languages. In European Conference on Modelling Foundations and Applications (pp. 305-320). Springer Berlin Heidelberg.
Xanthopoulos, S., & Xinogalos, S. (2013, September). A comparative analysis of cross-platform development approaches 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 runtime environment for mobile widget-based application. In Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on (pp. 68-71). IEEE