APPROCHE MDA POUR AUTOMATISER LA GENERATION DE CODE NATIF POUR LES APPLICATIONS MOBILES MULTIPLATEFORMES

Mohamed Lachgar

UNIVERSITE CADI AYYAD
FACULTE DES SCIENCES ET TECHNIQUES- MARRAKECH

18/2017

CENTRE D'ETUDES DOCTORALES
« Sciences de l'Ingénieur »

THÈSE

présentée à la Faculté des Sciences et Techniques de Marrakech
pour obtenir le grade de :

Docteur

Formation Doctorale : Génie Informatique, Mathématique Appliquées aux Sciences de l'Ingénieur

Spécialité : Informatique

APPROCHE MDA POUR AUTOMATISER LA GENERATION DE CODE NATIF POUR LES APPLICATIONS MOBILES MULTIPLATEFORMES

par :

Mohamed LACHGAR

(Ingénieur d'Etat: Génie Informatique)

soutenue le 08/07/2017 devant la commission d'examen :

P. Said RAGHAY	PES	FST-Marrakech	Président
P. El Hassan ABDELWAHED	PES	FSS-Marrakech	Rapporteur
P. Aziz SOULHI	PES	ENSM-Rabat	Rapporteur
P. Mostafa EZZIYYANI	PES	FST-Tanger	Rapporteur
P. Said RAKRAK	PES	FST-Marrakech	Examinateur
P. Salah EL-HADAJ	PH	ENCG-Marrakech	Examinateur
P. Abdelmounaïm ABDALI	PES	FST-Marrakech	Directeur de thèse

Mohamed LACHGAR
APPROCHE MDA POUR AUTOMATISER LA GENERATION DE CODE NATIF POUR LES APPLICATIONS MOBILES MULTIPLATEFORMES
Année : 2017 Intitulé de CED : Sciences de l'Ingénieur

Remerciements

Tout d'abord, je tiens à exprimer mes vifs remerciements à **Monsieur Abdelmounaïm ABDALI**, Professeur à la faculté des sciences et techniques de Marrakech, pour la confiance qu'il m'a témoignée en acceptant d'être le directeur de ma thèse. Ses grandes qualités pédagogiques et humaines ainsi que son soutien, sa gentillesse et sa grande disponibilité m'ont permis de mener à bien ce travail.

Je remercie sincèrement les rapporteurs **Monsieur El Hassan ABDELWAHED**, **Monsieur Aziz SOULHI** et **Monsieur Mostafa EZZIYYANI** qui ont accepté de rapporter cette thèse. Je leur exprime toute ma reconnaissance pour l'intérêt porté à ce travail.

Je tiens à remercier **Monsieur Said RAGHAY**, d'avoir accepté d'examiner ce travail et de présider le jury de soutenance.

Je voudrais remercier également les professeurs, **Monsieur Said RAKRAK** et **Monsieur Salah EL-HADAJ** d'avoir accepté d'examiner ce travail.

Je remercie tous les membres de laboratoire et institution : Laboratoire de Mathématiques Appliquées et Informatique (LAMAI), pour leur collaboration, leur gentillesse et leur soutien.

Enfin, je tiens à remercier mes amis, mes proches et ma famille pour leur encouragement.

Merci

Avant-propos

Nom et Prénom de l'auteur: Mohamed LACHGAR

Intitulé du travail: Approche MDA pour automatiser la génération de code natif pour les applications mobiles multiplateformes.

Encadrant :

Nom, prénom et grade : ABDALI Abdelmounaïm, PES

Equipe et institution : Laboratoire et institution : Laboratoire de Mathématiques Appliquées et Informatique (LAMAI), Faculté des Sciences et Techniques Guéliz (FSTG) – Université Cadi Ayyad, Marrakech.

Liste des publications liées à la thèse:

Revues internationales:

- Lachgar, M., & Abdali, A. (2017). "Decision Framework for Mobile Development Methods". International Journal of Advanced Computer Science and Applications (IJACSA), vol. 8, no 2.
- Lachgar, M., & Abdali, A. (2017). "Rapid Mobile Development: Build Rich, Sensor-Based Applications using a MDA approach". International Journal of Computer Science and Network Security (IJCSNS), 17(4), 274.
- Lachgar, M., & Abdali, A. (2016). "Modeling and generating native code for cross-platform mobile applications using DSL". Intelligent Automation & Soft Computing, pp. 1-14.
- Lachgar, M., & Abdali, A. (2015). "Modeling and Generating the User Interface of Mobile Devices and Web Development with DSL". Journal of Theoretical & Applied Information Technology, vol. 72, no 1.

Conférences internationales :

- Lachgar, M., & Abdali, A. (2015, November). "DSL and code generator for accelerating iOs apps development". In the Third World Conference on Complex Systems (WCCS), 2015 (pp. 1-8). IEEE.
- Lachgar, M., & Abdali, A. (2014, October). "Generating Android graphical user interfaces using an MDA approach". In the Third IEEE International Colloquium in Information Science and Technology (CIST), 2014 (pp. 80-85). IEEE.

Conférences nationales :

- Lachgar, M., & Abdali, A. (2015, Novembre). "Survey of mobile development approaches". Dans les journées Doctorales en Systèmes d'Information, Réseaux et Télécommunication (JDSIRT). 12-13 novembre 2015, ENSIAS - Rabat.

- Lachgar, M., & Abdali, A. (2015, Mai). “Generating Native Code for Android and Windows Phone Applications using an MDA Approach”. In the First Spring Conference on Applied Science and Computing. 30-31 Mai 2015, EST d’Essaouira (*Meilleure communication orale*).

- Lachgar, M., & Abdali, A. (2015, Mai). “Étude comparative des approches de génération de code pour les applications mobiles, et convergence vers une modélisation pragmatique”. In the First Spring Conference on Applied Science and Computing. 30-31 Mai 2015, EST d’Essaouira.

- Lachgar, M., & Abdali, A. (2014, Juin). “Modélisation et génération des interfaces graphiques Android et JSF 2.2 selon une approche MDA”. Dans la 6ème édition des journées Doctorales Nationales en TIC (JDTIC’ 14). 19-20 Juin 2014, ENSIAS - Rabat.

Liste des publications qui ont cité notre travail :

- Valiyev, R. A., Galiullin, L. A., & Iliukhin, A. N. (2015). “Design of the Modern Domain Specific Programming Languages”. Int. J. Soft Comput, 10(5), 340-343.

- Benouda, H., Azizi, M., Esbai, R., & Moussaoui, M. (2016). “MDA approach to automate code generation for mobile applications”. In Mobile and Wireless Technologies 2016 (pp. 241-250). Springer Singapore.

- Benouda, H., Azizi, M., Esbai, R., & Moussaoui, M. (2016). “Code generation approach for mobile application using acceleo”. International Review on Computers and Software (IRECOS), 11(2), 160-166.

- Freitas, F., & Maia, P. H. M. (2016, November). “JustModeling: An MDE Approach to Develop Android Business Applications”. In Computing Systems Engineering (SBESC), 2016 VI Brazilian Symposium on (pp. 48-55). IEEE.

- Channonthawat, T., & Limpiyakorn, Y. (2016, May). “Model Driven Development of Android Application Prototypes from Windows Navigation Diagrams”. In Software Networking (ICSN), 2016 International Conference on (pp. 1-4). IEEE.

- Núnez, M., González, M., Cernuzzi, P. D. L., & Aquino, M. S. N. (2016). “Propuesta MDD para la persistencia de datos en aplicaciones nativas orientadas a teléfonos móviles inteligentes”.

- Yizhou, J., Liwei, S., Xin, P., & Wenyun, Z. (2016). “Android code automatic generation based on application description”, Computer applications and software, 33(11), 169-174.

- Akbulut, A., Akbulut, F. P., Köseokur, H., & Çatal, Ç. (2017). “Son Kullanıcı Geliştirme için Otomatik Kod Üretim Aracının Tasarımı ve Gerçeklenmesi”. Journal of Science and Engineering, 19 (55.1).

- Zhao, Y., Cai, H., Zhao, H., & Xie, Y. (2016). "Self-Adaptive foreground extration based on improved Gaussian mixture model", Computer applications and software, 33(11), 161-163.

- Veisi, P., & Stroulia, E. (2017, May). "AHL: Model-Driven Engineering of Android Applications with BLE Peripherals". In International Conference on E-Technologies (pp. 56-74). Springer, Cham.

- Novák, R. (2017). Návrh konceptu a vývoj prototypu univerzálního uživatelského rozhraní.

Résumé

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 de soucis tel que l'efficacité du code, l'interaction avec les périphériques, ainsi que la rapidité d'envahissement du marché. L'Ingénierie Des Modèles (IDM) ou Model-Driven Engineering (MDE) combiné avec UML, comme cela a été déjà adopté dans le génie logiciel, pourrait fournir une abstraction et une automatisation pour les développeurs de logiciels mobiles. Pour appuyer cela, des outils et des approches adéquates sont nécessaires. Ce projet de thèse présente une approche MDE pour le développement des applications mobiles, qui inclut la modélisation avec un langage dédié, la modélisation UML et la génération de code **afin de faciliter et d'accélérer le développement des applications mobiles**.

Cette thèse est découpée en quatre parties principales. Dans la première partie nous avons étudié les différentes approches de développement mobiles, et nous avons proposé un cadre décisionnel permettant d'évaluer les différentes approches vis-à-vis des besoins techniques de l'application à développer.

Dans la deuxième partie nous avons proposé une approche basée sur un langage dédié nommée **GUI DSL**, afin de modéliser les interfaces graphiques des applications web et mobiles basées sur les composants, ensuite, nous avons généré le code source pour Android et le Framework JSF avec les composants Primefaces.

Dans la troisième partie, nous avons étendu notre méta-modèle basé sur le langage dédié, présenté dans la partie précédente et nous l'avons nommé **Mobile DSL**, afin de modéliser les applications mobiles en tenant compte des fonctionnalités natives et les différentes caractéristiques d'une application mobile, ensuite, nous avons proposé un ensemble de transformations vers les plateformes cibles (Android, Windows phone et iOS), et nous avons généré le code ciblant ces trois plateformes.

Dans la dernière partie nous avons proposé **une approche pragmatique** en combinant le langage dédié et le langage UML afin de générer une application mobile qui respecte l'architecture en couche.

Mots clés : MDA, approches de développement mobile, développement mobile multiplateforme, UML, langage dédié, transformation de modèles, générateur de code.

Abstract

The industry of mobile application development nowadays lives a non-stop growth, due to the intensive use of this latter in mobile devices, most of these mobile applications works under Android, iOS and Windows Phone as operating systems.

Nonetheless, the development of applications designed for mobile platforms requires more worries such as code efficiency, interaction with peripherals, as well as the speed of market invasion. Since Model-Driven Engineering (MDE) combined with UML, as already adopted in software engineering, could provide abstraction and automation for mobile software developers. To support this, appropriate tools and approaches are needed

This thesis project presents an MDE approach for the development of mobile applications, which includes modeling with a dedicated language, UML modeling and code generation to facilitate and accelerate the development of mobile applications.

This thesis is divided into four main parts. In the first part we studied the different mobile development approaches and we proposed a decision-making framework to evaluate the different approaches to the technical needs of the application to be developed.

In the second part, we propose an approach based on a dedicated language called DSL GUI, in order to model the graphical interfaces of the web based and mobile applications based on the components, then we generated the source code for Android and the JSF Framework with the Primefaces components.

In the third part, we have extended our meta-model based on a dedicated language, presented in the previous section, in order to model the mobile applications taking into account the native functionalities and the different characteristics of a mobile application. Afterwards, we suggested a set of transformations to target platforms (Android, Windows phone and iOS), and we generated the code targeting these three platforms.

In the last part we proposed a pragmatic approach by combining the dedicated language and the UML in order to generate a mobile application that respects the layered architecture.

Keywords : MDA, Mobile development approaches, cross-platform mobile development, UML, DSL, model transformation, code generator.

ملخص

صناعة وتطوير التطبيقات النقالة في تزايد مستمر بسبب الاستخدام المكثف لهذه الأخيرة في الأجهزة النقالة، ومعظمها تعمل على أنظمة التشغيل Android، iOS وWindows Phone .

ومع ذلك، فإن تطوير تطبيقات مصممة للأنظمة الأساسية للجوال تتطلب المزيد من التطلعات كنجاعة الشيفرة المصدرية، والتفاعل مع الأجهزة، وسرعة اكتساح السوق .هندسة نموذج (IDM) أو (MDE) جنبا إلى جنب مع UML، كما اعتمد بالفعل في هندسة البرمجيات، يمكن أن تقدم نوعا من التجريد و الآلية اللازمين للمطوري البرمجيات المتنقلة .لدعم هذا، هناك حاجة إلى أدوات ومقاربات ملائمة.

مشروع أطروحتنا ينهج مقاربة من نوع MDE لتطوير التطبيقات النقالة، بما في ذلك النمذجة مع لغة معينة، والنمذجة UML وتوليد شيفرة لتسهيل وتسريع عملية تطوير التطبيقات النقالة.

ينقسم هذا البحث إلى أربعة أجزاء رئيسية .في الجزء الأول ألقينا نظرة على مختلف أساليب التنمية النقالة، واقترحنا إطار صنع للقرار, لتقييم مختلف المقاربات بخصوص المتطلبات التقنية للتطبيق المزعم تطويره.

في الجزء الثاني اقترحنا نهج قائم على لغة معينة تحمل اسم *GUI DSL* لنمذجة واجهات المستخدم الرسومية والهاتف المحمول على أساس المكونات الرسومية، ثم، إنشاء شفرة المصدرية من أجل Android وإطار JSF مع المكونات primefaces .

في الجزء الثالث، قمنا بتوسيع ال Meta-model الخاص بنا, الذي يقوم على لغة معينة، أسميناها DSL النقال (Mobile DSL), كما تقدم معنا في الجزء السابق، من أجل نموذج التطبيقات النقالة مع الأخذ بعين الاعتبار الوظائف الأصلية ومختلف مميزات التطبيقات النقالة، ثم اقترحنا مجموعة من التحويلات صوب المنصات المستهدفة (Android وWindows phone وiOS)، وتم إنتاج التعليمات البرمجية الخاصة بالمنصات الثلاث.

في الجزء الأخير اقترحنا نهج عملي من خلال الجمع بين اللغة المقترحة (أي DSL النقال) واللغة UML من أجل إنتاج تطبيق نقال يوافق الهندسة الطبقية.

الكلمات الرئيسية: MDA , مقاربات التنمية النقالة, التنمية النقالة للمنصات المتعددة, UML , DSL , تحويل النماذج ,مولد الشيفرات.

Glossaire

Abréviation	Signification
2D	Deux Dimensions
3D	Trois Dimensions
AGL	Atelier de Génie Logiciel
AMMA	Architecture Atlas Model Management
API	Application Programming Interface
ART	Android Runtime
ASTM	Abstract Syntax Tree Metamodel
ATL	Atlas Transformation Language
BLL	Business Logic Layer
BOL	Business Objects Layer
CIM	Computation Independent Model
CNRS	Centre National de la Recherche Scientifique
COM	Component Object Model
CRUD	Create, Read, Update And Delete
DAL	Data Access Layer
DEX	Dalvik EXecutable
DL	Data Layer
DSL	Domain Specific Language
DSML	Domain-Specific Modeling Language
DSM	Domain-Specific Modeling
DTD	Document Type Definition
EJB	Enterprise Javabeans
EMF	Eclipse Modeling Framework
ETP	Eclipse Tools Project
GPML	Gnu Mathematical Programming Language
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
IDM	Ingénierie Dirigée par les Modèles
IHM	Interface Homme Machine
INRIA	Institut National de Recherche en Informatique et en Automatique
INSA	Institut National des Sciences Appliquées
JDT	Eclipse Java Development Tools
JEE	Java Enterprise Edition
JSAF	Javascript Application Framework
JSF	JavaServer Faces
JSON	JavaScript Object Notation

JSR	Java Specification Requests
KDM	Konwledge Discovery Metamodel
KM3	Kernel Meta Meta Model
M2C	Model To Code
M2M	Model To Model
M2T	Model To Text
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MOF	Meta-Object Facility
MVC	Model–View–Controller
OCL	Object Constraint Langage
OMG	Object Management Group
OPL	Open Programming Language
OS	Operating System
PDM	Plateform Dependant Model
PHP	Hypertext Preprocessor
PIM	Platform Independent Model
PSM	Platform Specific Model
QVT	Query/View/Transformation
RAD	Rapid Application Development
REST	Representational State Transfer
RFP	Request For Proposal
SAL	Service Access Layer
SDK	Software Development Kit
SMS	Short Message Service
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
UGI	Graphic Design And User-Interface
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
VMTS	Visual Modeling and Transformation System
W3C	World Wide Web Consortium
WLAN	Wireless Local Area Network
XMI	Xml Metadata Interchange
XML	Extensible Markup Language
XNU	X is Not Unix
XSLT	Extensible Stylesheet Language Transformations

Liste des Figures

Liste des Tableaux

Sommaire

Introduction générale

1. Contexte générale

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 conséquence, 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 thèse 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.) en répondant aux limitations des solutions existantes. En effet, la plupart des approches de développement mobile existantes basées sur l'approche MDA permettent de générer en gros la couche présentation et la structure de la couche logique d'une application mobile multiplateforme, sans tenir compte de la séparation de couches. Toutefois, aucune des approches, n'offre un méta-modèle pour modéliser une application qui accède aux fonctionnalités natives d'un smartphone et un générateur de code basé sur le méta-modèle proposé. Ainsi, nous résumons les limites des approches existantes ci-dessous :

- Manque d'un méta-modèle générique permettant de modéliser une application mobile complète ;
- Manque d'un générateur de code permettant la génération d'une application mobile respectant la programmation en couche ;
- Manque d'un méta-modèle pour la modélisation des fonctionnalités natives et un générateur de code natif basé sur ce méta-modèle ;
- Manque d'un méta-modèle pour modéliser les transitions entre les écrans ;
- Manque d'une description détaillé des règles de transformations utilisées lors de l'implémentation des générateurs de code ;
- Manque d'un générateur de code pour générer à la fois une application métier orientée données (e.g. application de gestion) et une application multimédia (e.g. Intégration des vidéos et audio) ;
- Manque d'un méta-modèle pour la génération des services distants ;
- Manque d'un méta-modèle pour la génération des bases de données embarquées.

3. Principe de notre approche

L'objectif de la présente thèse est de présenter un socle méthodologique et technique 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 de modèles.

4. Contributions

Nos contributions dans cette thèse portent sur les axes suivants :

- Etude des approches de développement mobiles et la mise en place d'un cadre décisionnel pour ces approches ;

- Modélisation et conception d'un DSL pour la génération des interfaces graphiques des plateformes mobiles et web basées sur les composants, ensuite, implémentation d'un prototype de générateur de code pour la génération des GUIs multiplateformes ;
- Etendre le DSL pour prendre en compte les aspects comportementaux des applications basées sur les composants et aussi pour générer les classes permettant le traitement des données ;
- Etendre le méta-modèle basé sur le DSL pour prendre en charge la modélisation des applications mobiles qui font recours aux fonctionnalités natives et aux capteurs embarqués, ensuite, implémentation d'un prototype de générateur de code natif pour les applications mobiles multiplateformes ;
- Implémentation d'un prototype de générateur de code pour la génération du code natif des applications mobiles multiplateformes respectant une architecture en couche selon une approche MDA, en se basant sur une approche pragmatique qui combine entre une modélisation avec un langage dédié (DSL) et le langage UML.

5. Structure du mémoire

Le reste de ce mémoire s'étale sur les six chapitres suivants:

Chapitre 1 - Généralités sur le développement des applications mobiles. 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 dévoilons notre cadre décisionnel, permettant dans un premier temps, de sélectionner la méthode convenable pour mettre en place une application mobile bien spécifique, dans un deuxième temps, de classer les outils à utiliser pour développer l'application, et cela, en se basant sur un ensemble de critères jugés pertinents.

Chapitre 2 - Etat de l'art. Afin de situer notre travail, nous dressons principalement dans ce chapitre, un état de l'art des travaux qui s'intéressent à notre problématique de modélisation et de génération de code pour les applications mobiles multiplateformes. Premièrement, nous commencerons par une description de modèle de développement multiplateforme et les travaux connexes ; en commençant par les travaux réalisés pour la définition des méta-modèles pour les plateformes mobiles comme Android et Windows phone, ensuite, nous détaillons les différentes approches de modélisation et de génération de

code pour les applications mobiles multiplateformes. Enfin, nous discuterons les limites des propositions actuelles et nous conclurons ce chapitre par une synthèse des travaux présentés.

Chapitre 3 - L'ingénierie dirigée par les modèles – IDM. Ce chapitre introduit l'environnement scientifique sur lequel s'appuie cette thèse. Plus précisément, il introduit les principes généraux de l'ingénierie à base de modèles et présente un tour d'horizon sur les outils et les langages de modélisation et de transformation des modèles.

Chapitre 4 - Modélisation et génération des interfaces graphiques des applications basées sur les composants selon une approche MDA. Dans ce chapitre nous présentons notre approche pour la conception et la génération des interfaces utilisateurs pour les applications mobiles et web basées sur les composants, ensuite, nous appliquons l'approche sur la plateforme Android et le Framework Java Server Faces.

Chapitre 5 - Modélisation et génération du code natif des applications mobiles multiplateformes en utilisant un DSL. La principale contribution de ce chapitre est une approche pilotée par les modèles à base de MDA, afin de générer le code natif pour les applications mobiles. Dans un premier lieu, nous rappelons brièvement quelques préalables sur l'apport de l'approche MDA pour le développement mobile et l'architecture MDA avec le DSL. Ensuite, nous décrivons notre méthodologie proposée pour la génération des applications mobiles natives multiplateformes. Un exemple illustratif est discuté dans ce chapitre, qui montre les différentes étapes suivies au cours de notre approche.

Chapitre 6 - Approche Pragmatique pour la Modélisation et la Génération des Applications Mobiles Multiplateformes. Dans ce chapitre nous suggérons une approche permettant aux développeurs de générer des applications mobiles natives, et cela en combinant le langage UML et le langage dédié (DSL) pour modéliser ces applications et par la suite générer le code natif pour ces systèmes variés selon une architecture en couches.

Conclusion et perspectives. Nous achevons ce mémoire par une conclusion qui synthétise les contributions de notre approche et résume le travail effectué dans cette thèse et les perspectives envisagées, suivi par la liste des références bibliographiques.

Chapitre 1

Généralités sur le développement des applications mobiles

“Le téléphone mobile représente pour les habitants des marchés émergents un portail de savoir, de divertissement et de communication, et souvent le seul. Il est devenu indispensable dans la vie et les affaires des personnes qui l’utilisent"

Nokia

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 rapport à la même période en 2014. Les chiffres et les tendances présentées dans l'étude suivante confirment ce constat (*Gartner, 2015*).

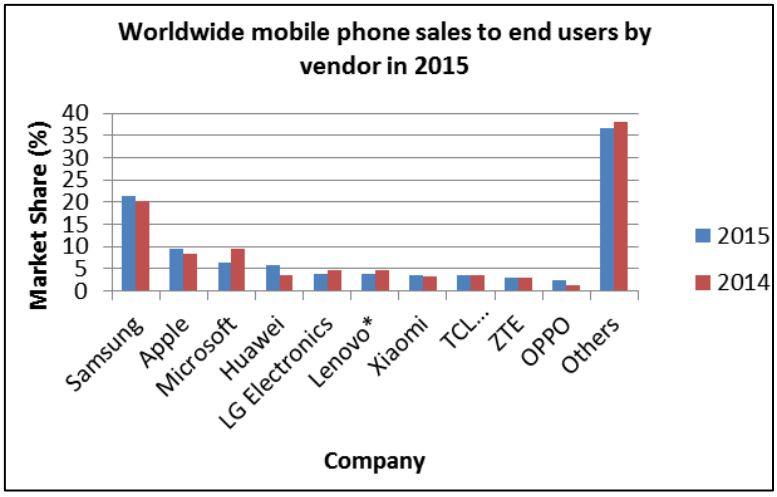


Figure 1. 1 - Les ventes des téléphones mobiles dans le monde pour les utilisateurs finaux par fournisseur

Cette évolution est due à la croissance du marché des smartphones, ce qui pousse les consommateurs à abandonner les téléphones classique 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.

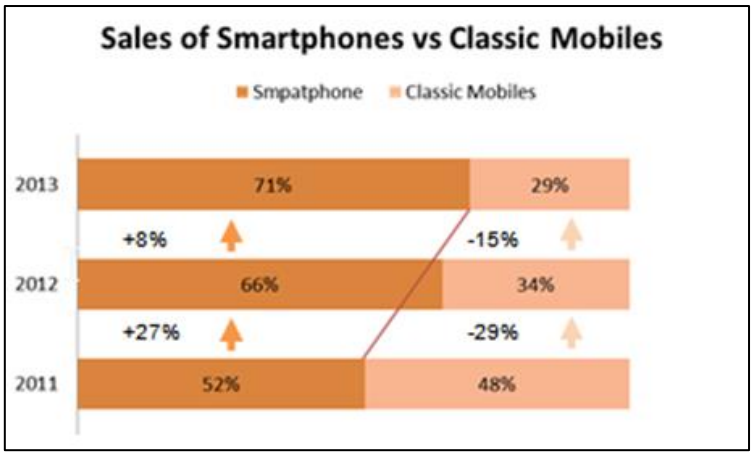


Figure 1. 2 - Les ventes des Smartphones vs mobiles classiques (*Kerensen Consulting, 2015*)

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

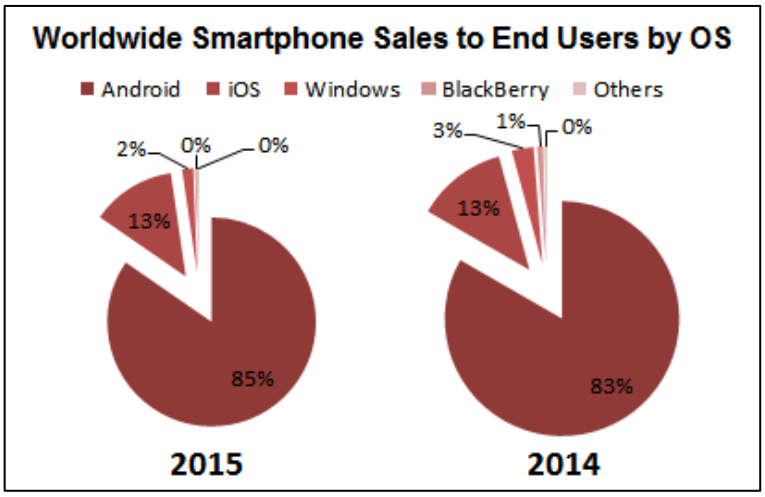


Figure 1. 3 - Les ventes des Smartphones dans le monde pour les utilisateurs finaux par OS (*Gartner, 2015*)

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-dessous 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 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 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 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 (*traduit de la documentation Google*).

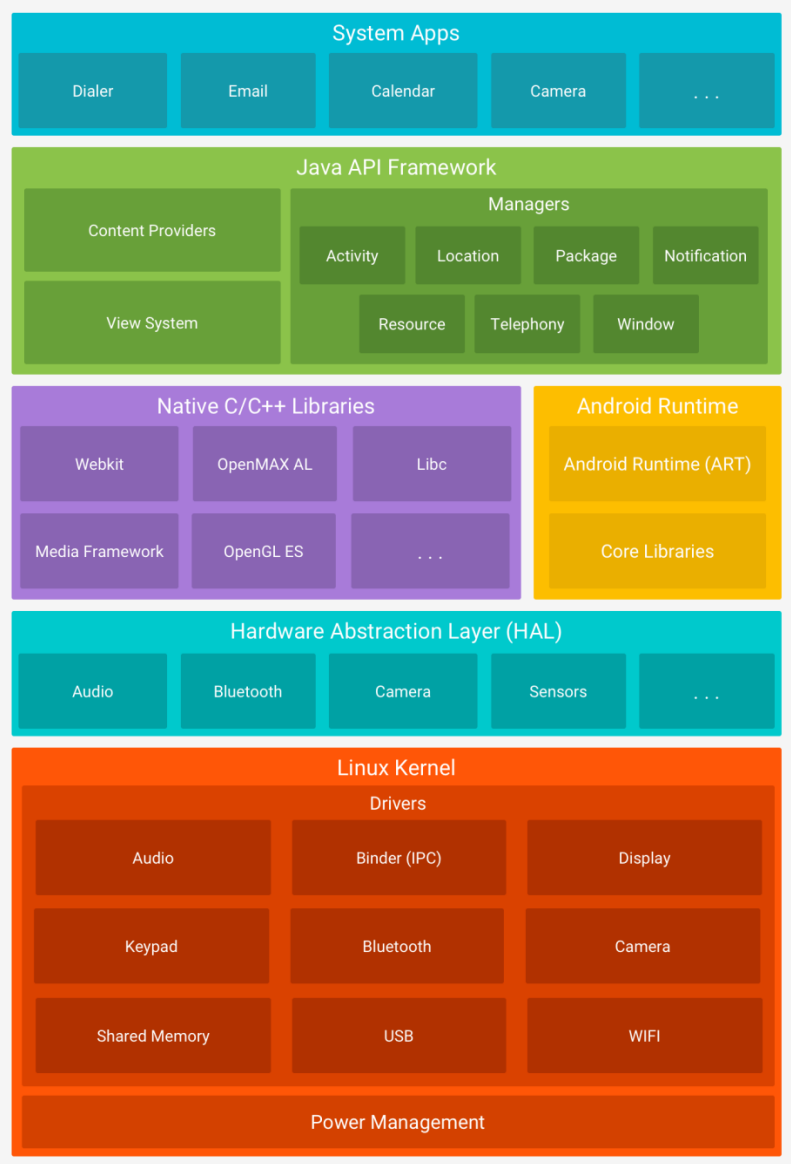


Figure 1. 4 - Architecture Android (*Android, 2017*)

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, etc.

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.

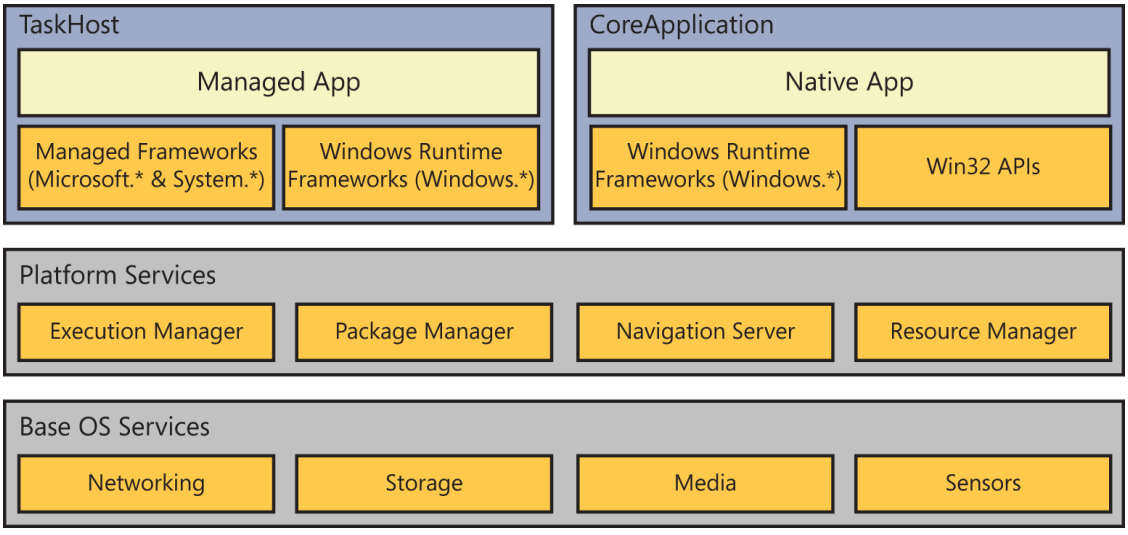


Figure 1. 6 - Architecture Windows Phone

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 conséquence, elle pousse les développeurs à faire un choix sur la plateforme, tout en assurant la plus grande distribution possible.

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

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

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 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 :

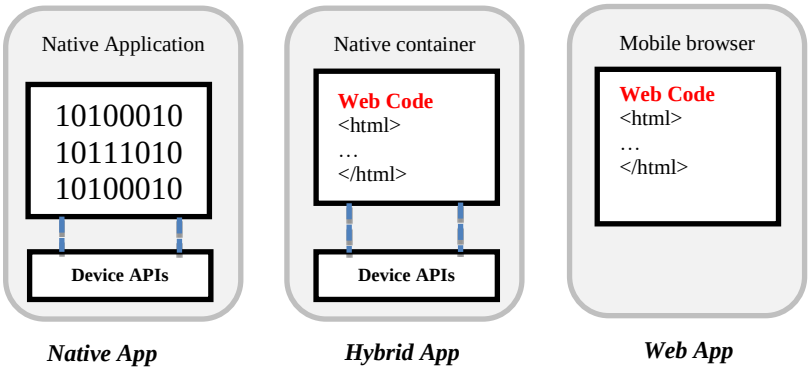


Figure 1. 7 - Méthode de développement des applications mobiles

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 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, 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:

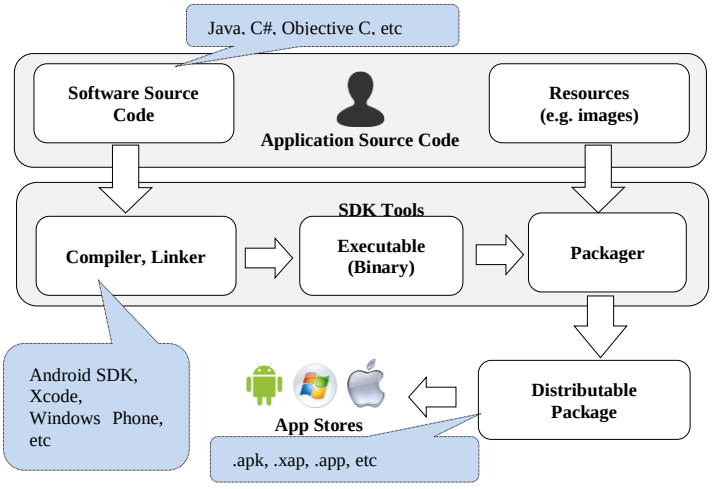


Figure 1. 8 - Architecture d'une application native

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

Tableau 1. 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 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 (*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 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 :

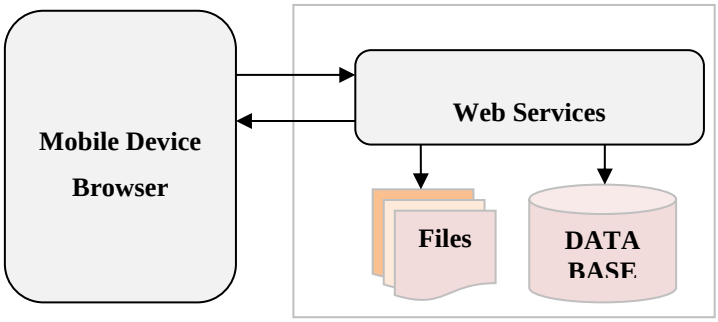


Figure 1. 9 - L'architecture logique d'une application web mobile (*Raj et al, 2012*)

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

Tableau 1. 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 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 ci-dessous :

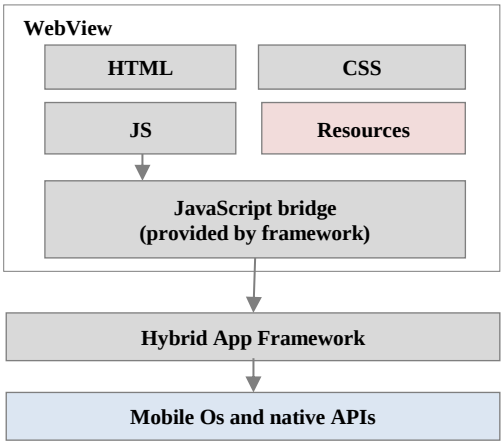


Figure 1. 10 - L'architecture logique d'une application hybride typique

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

Tableau 1. 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 " présenté ci-dessous illustre la tendance de l'approche native par rapport aux facteurs coût et temps des approches de développement multiplateformes.

Tableau 1. 5 - Comparaison des approches de développement des applications mobiles

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 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 development framework (like PhoneGap, Ionic)	Objective-C, Java, C, C++, C#, VB.net

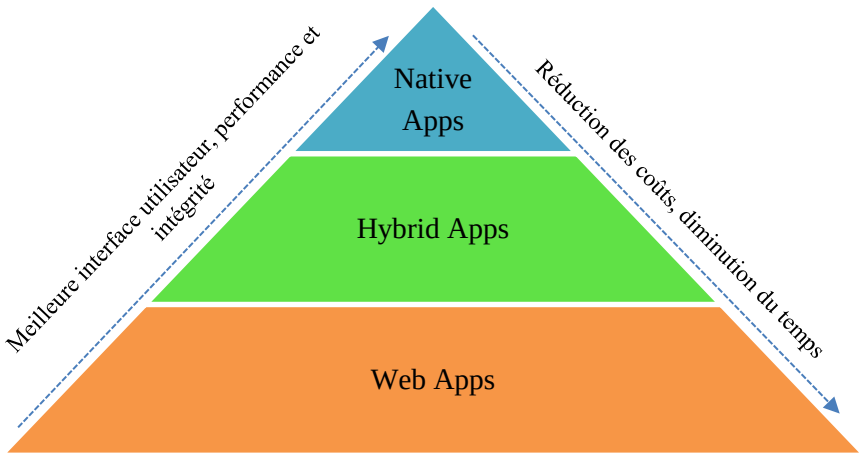


Figure 1. 11 - Approche native vs développement multiplateformes

1.5. Cadre décisionnel pour les méthodes et les outils de développement mobile

Nous avons montré que les trois solutions ont des avantages et des inconvénients. La question maintenant qui se pose est : Quelles sont les approches qui peuvent être adoptées afin de développer une application multiplateforme mobile ? Et quel est l'outil à utiliser pour implémenter une application particulière ?

Pour répondre à ces questions, **nous proposons** un cadre permettant de fournir des réponses fondées sur la nature de l'application à développer. L'architecture de cet outil est présentée dans la figure ci-dessous (*voir la figure 1.12 pour plus de détails*).

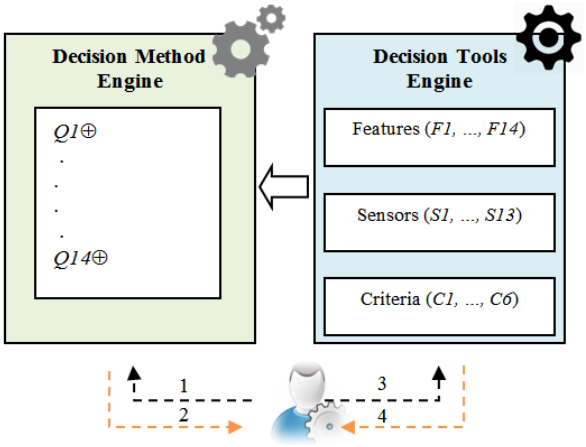


Figure 1. 12 - Architecture de Framework proposée (*Lachgar et al, 2017*)

De façon plus détaillée, le fonctionnement de ce Framework est basé sur quatre étapes clés:

(1) La première étape, consiste à remplir un sondage, puis envoyer les réponses au moteur de décision de la méthode (*Decision Method Engine*) ;

(2) Dans la deuxième étape, le moteur de décision de la méthode analyse les réponses et transmet la méthode de développement mobile approprié au client et détermine également le pourcentage d'achèvement de chaque méthode ;

(3) Dans la troisième étape, selon la méthode reçue, le client doit remplir un questionnaire, puis le transmet au moteur de décision de l'outil (*Decision Tools Engine*);

(4) Dans la dernière étape, le moteur de décision de l'outil analyse les réponses et envoie l'outil adéquat pour mettre en place l'application. Dans le cas hybride, les outils sont classés en fonction des caractéristiques recherchées dans l'application à développer.

Les sous-sections suivantes décrivent la mise en place de chaque bloc.

1.5.1. Moteur de décision de méthode (Decision Method Engine)

Pour mettre en place le moteur de décision de méthode, **nous avons proposé** une série de

questions afin de désigner la bonne approche pour développer une application mobile bien spécifique. Ces questions sont représentées ci-dessous.

Q 1 : L'application devrait être publiée sur les boutiques en ligne ?
Q 2 : L'application fonctionne en mode hors connexion ?
Q 3 : Voulez-vous vendre l'application ?
Q 4 : Est-ce que c'est une simple application ?
Q 5 : L'application sera fréquemment utilisée par l'utilisateur ?
Q 6 : Y a-t-il un besoin immédiat de livrer l'application sur le marché ?
Q 7 : Avez-vous des budgets distincts pour développer l'application pour chaque OS ?
Q 8 : Avez-vous besoin d'un grand nombre de fonctionnalités natives dans l'application ?
Q 9 : Est-ce que la sécurité de l'application présente une haute priorité ?
Q 10 : L'application devrait être très fluide ?
Q 11 : Voulez-vous beaucoup d'animations dans l'application ?
Q 12 : Est-ce que l'application a besoin de faire recours à des algorithmes de calcule ?
Q 13 : Voulez-vous être toujours à jour avec les dernières versions de l'OS ?
Q 14 : Vous voulez avoir la meilleure expérience utilisateur ?

Le tableau ci-dessous donne les réponses à ces questions pour chaque approche de développement mobile (native, hybride et web).

Tableau 1. 6 - Cadre décisionnel des méthodes de développement mobile (*Lachgar et al, 2017*)

	Native	Hybrid	Web
Q 1	✔	✔	✘
Q 2	✔	✔	✘
Q 3	✔	✔	✘
Q 4	✘	✘	✔
Q 5	✔	✔	✘
Q6	✘	✔	
Q 7	✔	✘	
Q 8	✔	✘	
Q9	✔	✘	
Q 10	✔	✘	
Q 11	✔	✘	
Q12	✔	✘	
Q 13	✔	✘	
Q 14	✔	✘	

Dans cette perspective, nous présentons les critères de sélection établis dans un arbre de décision représenté dans la figure ci-dessous.

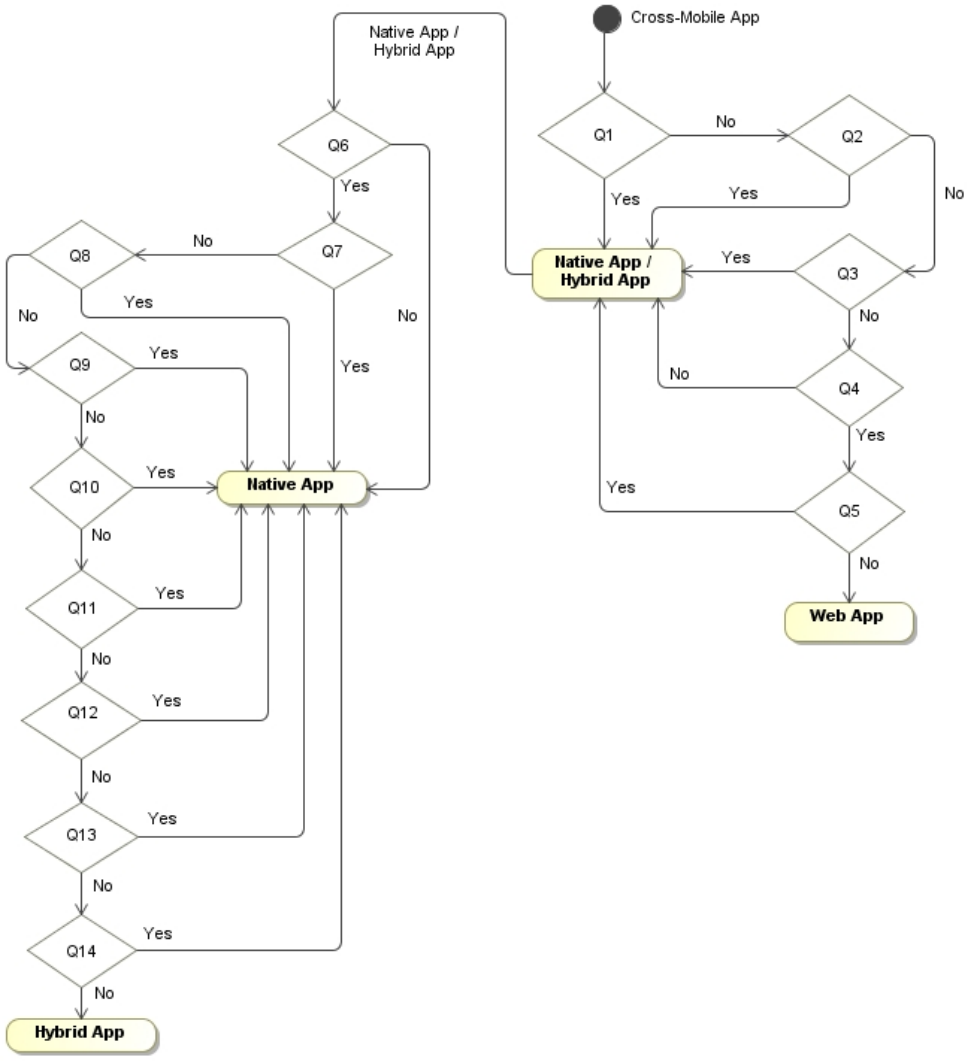


Figure 1. 13 - Arbre de décision pour l'adoption de la méthode de développement approprié (*Lachgar et al, 2017*)

1.5.2. Mise en œuvre de moteur de décision de méthode

L'arbre de décision illustré à la figure ci-dessus (*Figure 1.13*) est utilisé pour déterminer l'approche de développement mobile à adopter au sein d'une situation donnée. Le moteur de méthode de décision permettra également de déterminer le pourcentage d'achèvement de chaque méthode. Pour ce faire, nous avons adopté l'approche suivante.

Nous avons attribué, un facteur de décision où barème de pondération, à chaque question, en fonction de l'importance qui lui est accordée. Les intervalles sélectionnés clarifient ces facteurs:

- 8: Très important.
- 6: Important.
- 4: Peu important.

- 2: Pas du tout important.

Le tableau ci-dessous illustre les poids attribués à chaque question selon son importance:

Tableau 1. 7 - Facteurs attribués aux questions (*Lachgar et al, 2017*)

Question	Facteur (*Poids*)
Q1	8
Q2	4
Q3	6
Q4	2
Q5	2
Q6	6
Q7	8
Q8	6
Q9	6
Q10	4
Q11	4
Q12	6
Q13	6
Q14	6

La justification du choix des différents facteurs est fournie ci-dessous:

- Question 1 est primordiale, car elle permet de décider entre l'approche web et les deux autres approches ;
- Question 2, est moins importante, en particulier l'approche web permet la sauvegarde des données via le mode hors-ligne selon les innovations de HTML 5 ;
- Question 3 dépend de la question 1; une application mobile pour la vendre, doit être publiée dans les boutiques en ligne, donc, nous avons fourni le facteur de décision 6 pour cette question ;
- Question 4 pas tout importante, une application simple peut être développée par les différentes approches ;
- Question 5 n'est pas importante du tout, une simple application développée avec l'approche web, sans avoir recours aux API natives, peut également être utilisé fréquemment par les utilisateurs ;
- Question 6 est importante, de décider entre l'approche native et web. Si l'entreprise dispose de ressources humaines qualifiées pour développer l'application dans les délais souhaités, sera intéressante d'adopter l'approche native, ce qui explique le facteur 6 comme facteur de décision ;
- Question 7 est très importante, car elle permet de choisir entre l'approche native et l'hybride, ce qui explique le facteur 8 en tant que facteur de décision pour ces deux questions ;

- Question 8 peut faire la différence entre les approches natives et hybrides. Afin de mettre en œuvre une application, un accès à plusieurs API natives est nécessaire, il est donc préférable d'utiliser l'approche native ;
- Question 9 est importante, si la sécurité est une priorité, alors il serait préférable d'adopter l'approche native. Par conséquent, nous avons attribué 6 comme facteur de décision pour cette question ;
- Les questions 10 et 11 sont moins importantes, selon l'évolution de l'approche hybride qui prend en charge la mise en œuvre de certaines animations et la fluidité en fonction de l'évolution du Framework JavaScript, donc, pour les deux questions, nous avons attribué le facteur 4 ;
- Chaque fois que l'application nécessite beaucoup de calcul algorithmique, alors il est préférable d'utiliser un langage natif pour tirer profit des méthodes déjà développées. Cela explique 6 comme facteur de décision pour la question 12 ;
- Question 13 est importante, si une application mobile a besoin de plusieurs fonctionnalités natives, elle devrait bénéficier des dernières mises à jour du système d'exploitation, ce qui explique le facteur 6 comme facteur de décision ;
- L'expérience utilisateur est un point fort, pour cela nous avons attribué 6 comme facteur de décision pour la question 14.

Un extrait du diagramme de classe utilisé pour la mise en œuvre est illustré ci-dessous:

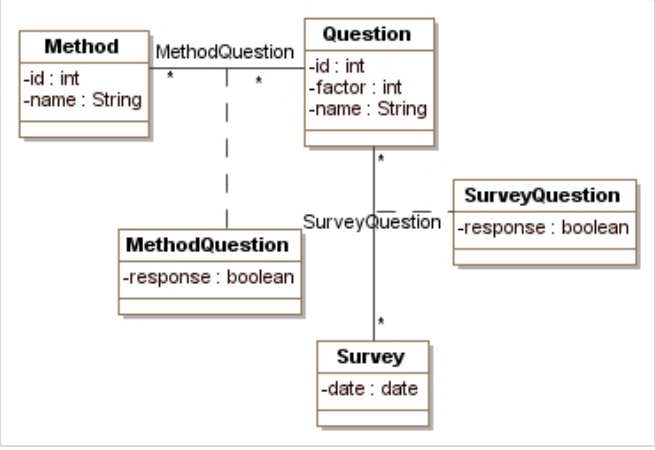


Figure 1. 14 - Extrait du diagramme de classe de moteur de décision de méthode (*Lachgar et al, 2017*)

La précision est donnée par la formule suivante:

$$\text{Precision (en \%)} = \frac{\sum \text{Factor}(Q_P)}{\sum \text{Factor}(Q_E)} * 100$$

Où:

$\sum \text{Factor}(Q_P)$: La somme des facteurs des questions effectuées.

$\sum \text{Factor}(Q_E)$: La somme des facteurs des questions attendues.

Avec : (Q_P) sont les questions réalisées et (Q_E) sont les questions attendues (*selon le tableau 1.6*).

1.5.3. Moteur de décision des outils (Decision Tools Engine)

Une fois que l'approche de développement est déterminée, la prochaine étape sera de définir les outils à utiliser pendant la phase de la mise en œuvre. Pour atteindre cet objectif, nous évaluons les différents besoins de l'application tels que : accès aux capteurs embarqués, accès aux fonctionnalités natives disponibles dans les téléphones mobiles et d'autres critères jugés utiles dans le processus de sélection.

Dans les tableaux ci-dessous une description des différentes fonctionnalités natives et les différents capteurs disponibles dans les smartphones.

Tableau 1. 8 - Les fonctionnalités standards dans un smartphone (*Lachgar et al, 2017*)

Code	Fonctionnalité	Définition
F1	Contacts	Est-ce que la solution prend en charge les fonctionnalités CRUD pour accéder à la liste de contacts ?
F2	Geolocation	Est-ce que la solution peut être capable d'utiliser le GPS ?
F3	Ad hoc Wi-Fi	Est-ce que la solution capable de gérer les connexions Wi-Fiad hoc ?
F4	Storage	Est-ce que la solution support les fonctionnalités CRUD pour le stockage local ?
F5	SMS	Est-ce que la solution possède une API pour envoyer des SMS à partir de l'application ?
F6	Telephony	Est-ce que la solution possède une API pour effectuer des appels à partir de l'application ?
F7	Bluetooth	Est-ce que la solution fournit un accès au périphérique Bluetooth ?
F8	Audio (Recording)	Est-ce que la solution permet la lecture audio dans l'application ?
F9	Audio (Reading)	Est-ce que la solution permet l'enregistrement audio dans l'application ?
F10	Camera (Take photo)	Est-ce que la solution permet de prendre des photos dans l'application ?
F11	Camera (VideoRecording)	Est-ce que la solution permet l'enregistrement de la vidéo dans l'application ?
F12	Vibration	Est-ce que la solution permet faire vibrer l'appareil depuis l'application ?
F13	Multi – touch	Est-ce que la solution peut être capable de capturer les "Gestures" ou "multi-touch" ?
F14	SOAP	Est-ce que la solution possède une API pour gérer le protocole SOAP ?
F15	Push Notification	Est-ce que la solution contient une API pour gérer "les notifications push" ?
F16	SQLite	Est-ce que la solution intègre la fonctionnalité Create, Read, Update et Delete (CRUD) de SQLite ?
F17	Network availability	Est-ce que la solution peut être capable de vérifier la disponibilité du réseau ?
F18	File System	Est- ce que la solution permet d'accéder au système de fichiers de l'appareil ?
F19	Memory management	Est-ce que la solution permet de gérer manuellement la mémoire ?

Tableau 1. 9 - Les capteurs standards dans un smartphone (*Lachgar et al, 2017*)

Code	Capteur	Définition
S1	Accelerometer	Est-ce que la solution permet d'accéder à l'accéléromètre ?
S2	Compass	Est-ce que la solution permet d'accéder au magnétomètre ou a fait une API pour créer une boussole ?
S3	Orientation	Est-ce que la solution permet la détection de la rotation du dispositif ?
S4	Light sensor	Est-ce que la solution permet l'accès au capteur de lumière ?
S5	Gravity	Est-ce que la solution permet l'accès au capteur de gravité ?
S6	Pressure	Est-ce que la solution permet l'accès au capteur de pression ?
S7	Gyroscope	Est-ce que la solution permet l'accès au capteur gyroscopique ?
S8	Proximity	Est-ce que la solution permet l'accès au capteur de proximité ?
S9	Temperature	Est-ce que la solution permet l'accès au capteur de température ?
S10	Ambient Temperature	Est-ce que la solution permet l'accès au capteur de température ambiante ?
S11	LinearAccelerometer	Est-ce que la solution permet l'accès au capteur de l'accéléromètre linéaire ?
S12	Magnetic Field	Est-ce que la solution permet l'accès au capteur de champ magnétique?
S13	Relative Humidity	Est-ce que la solution permet l'accès au capteur d'humidité relative ?

En outre, voici quelques critères qui peuvent être utiles dans le processus de sélection:

Tableau 1. 10 - Critères de sélection supplémentaires (*Lachgar et al, 2017*)

Code	Critère
C1	Taux de développement
C2	Documentation
C3	Look and feel
C4	Popularité
C5	Courbe d'apprentissage
C6	Outil graphique pour GUI

La sous-section suivante décrit et évalue les outils de développement mobiles, à l'égard des différents aspects que nous avons identifiés ci-dessus. Ces outils sont classés en trois catégories: les kits de développement de plateforme spécifique, les outils de développement mobile multiplateforme et les outils web.

1.5.4. Mise en œuvre de moteur de décision d'outils

Dans le cas de l'approche Web, les outils sont définis à savoir HTML5, CSS et JavaScript.

Dans l'approche native, selon les plateformes cibles, nous pouvons définir les outils à utiliser pour chaque plateforme. Dans ce cas, le choix est unique.

Dans le cas de l'approche hybride le moteur de décision des outils doit fournir un score qui sera calculé sur la base du nombre de fonctionnalités natives, capteurs nécessaires dans l'application qui sont pris en charge par l'outil, et l'évaluation des critères supplémentaires.

Maintenant, pour choisir le bon outil pour la mise en œuvre d'une application logicielle mobile, nous avons défini le barème suivant:

Evaluation API native ou capteur:

- 4: Bien pris en charge.
- 2: pris en charge.
- 0: pas soutenu.

Taux de développement:

- 3: très rapide.
- 2: rapide.
- 1: moyen.
- 0: lent.

Documentation:

- 3: très bon.
- 2: bon.
- 1: faible.
- 0: médiocre.

Look and Feel:

- 3: très bon.
- 2: bon.
- 1: faible.
- 0: médiocre.

Popularité:

- 3: très populaire (Very High).
- 2: populaire (High).
- 1: moins populaire (Medium).
- 0: non populaire.

Courbe d'apprentissage:

- 3: très rapide.
- 2: rapide.
- 1: medium.
- 0: long.

Outil graphique pour l'interface graphique:

- 2: bien supporté.
- 1: pris en charge.
- 0: pas pris en charge.

Un extrait du diagramme de classe utilisée pour la mise en œuvre est représenté sur la figure ci-dessous:

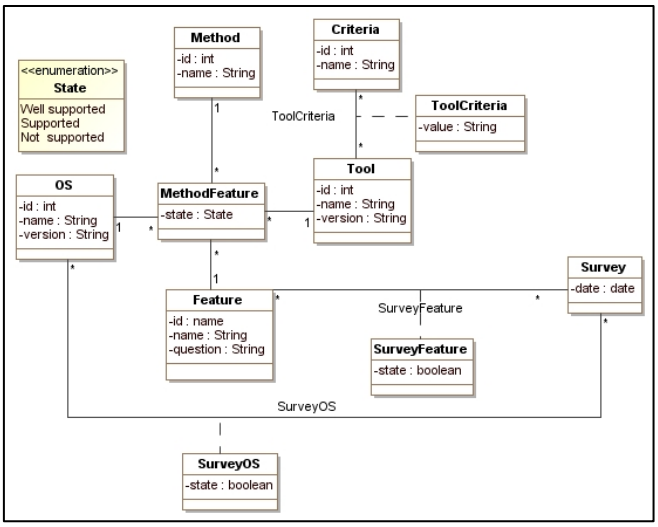


Figure 1. 15 - Extrait du diagramme de classe de moteur de décision d'outils (*Lachgar et al, 2017*)

1.5.5. Exemple illustratif

a. Description de projet à développer

Le but de ce projet est de développer une application mobile pour Android et iOS, basée sur la géolocalisation. Cette application permet de localiser la position des contacts figurant dans l'annuaire téléphonique, situé dans un rayon donné, en utilisant une carte Google Map, elle offre également la possibilité de communiquer avec d'autres personnes connectées au réseau avec la même application, en échangeant des messages texte où des fichiers multimédia (photo, vidéo), et enfin elle permet de prendre des photos et de les transmettre via l'application à d'autres contacts.

b. Besoins

En analysant la description de l'application, nous pouvons déduire les besoins ci-dessous :

- Disponible sur Android et iOS ;
- Accès au réseau ;
- Alerte de notification et vibration ;
- Accès à la caméra et à la vidéo ;
- Développement à bas coût ;
- Déploiement sur les magasins d'applications ;
- Accès aux médias ;
- Accès au GPS du Smartphone ;
- Accès à la liste des contacts ;
- Accès à la téléphonie.

c. Outils de développement proposés

Pour développer cette application, nous proposons d'évaluer les outils ci-dessous :

- F1 : PhoneGap + jQuery Mobile
- F2 : PhoneGap + Sencha Touch
- F3 : PhoneGap + Onsen IU
- F4 : PhoneGap + Angular UI
- F5 : PhoneGap + Ionic
- F6 : Titanium Appcelerator
- F7 : Xamarin
- F8 : Flex + Air

Nous avons réalisé différentes enquêtes (*voir Annexe 1*) pour évaluer les différentes outils présentés ci-dessus. Ensuite, nous avons appliqué notre démarche afin d'obtenu les résultats présentés ci-dessous.

d. Résultats

- Décision de méthode

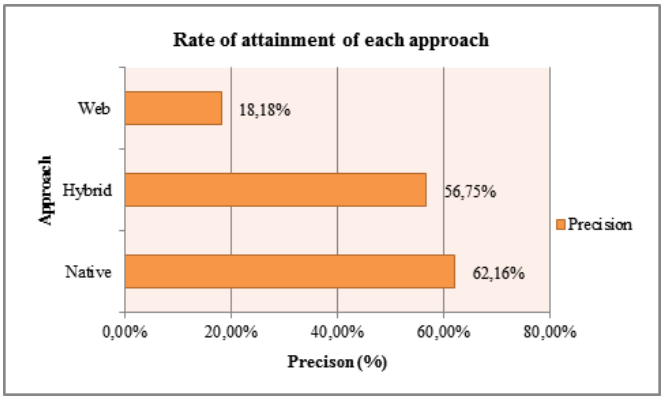


Figure 1. 16 - Taux de réalisation pour chaque méthode (*Lachgar et al, 2017*)

62.16% des besoins nécessitent d'adopter l'approche native ;

56.75% des besoins peuvent être implémentés avec l'approche hybride ;

18.18% des besoins peuvent être développés avec l'approche Web.

- Décision d'outil

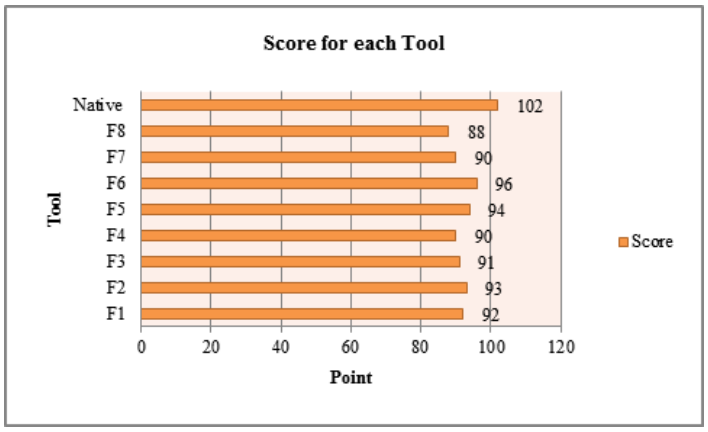


Figure 1. 17 - Score pour chaque outil (*Lachgar et al, 2017*)

Pour cette étude de cas, les kits de développement spécifiques à la plateforme sont parmi les meilleurs, Titanium Appcelerator au milieu suivi par PhoneGap avec le Framework Ionic. Cependant, Sencha Touch, et Flex occupent les dernières places.

1.5.6. Synthèse

Ce travail, présente un cadre permettant de sélectionner la meilleure technologie à utiliser pour le développement d'une application mobile spécifiée dans un contexte donné. Ce cadre se compose de deux étapes principales, la première consiste à déterminer la méthode de développement mobile (*natif, hybride ou web*) avec un pourcentage d'achèvement appelé précision, basée sur une série de questions pertinentes, la deuxième consiste à déterminer l'outil approprié pour la mise en œuvre de l'application basée sur un ensemble de critères pertinents.

Dans un monde idéal, sans contraintes de temps et d'argent, il serait évidemment plus intéressant de passer à une solution native. Le résultat a des avantages en termes d'ergonomie, de rendement et d'intégrité.

Cette étude nous a permis de comprendre dans quel cas, il est intéressant de se tourner vers les solutions web et hybride. Ces dernières, permettent un gain en temps opportun des performances simples et sans contraintes.

Par conséquent, **pour remédier aux faiblesses de l'approche native, nous suggérons la mise en place d'une solution basée sur l'ingénierie des modèles permettant le portage d'une application mobile sur les différentes plateformes**.

1.6. Discussion et travaux connexes

Plusieurs études ont été réalisées sur les méthodes de développement mobiles, où les chercheurs ont présenté les avantages et les inconvénients de chaque approche. Dans (***Dalmasso et al, 2013***), les auteurs ont présenté une étude comparative des outils de développement mobile multiplateforme (PhoneGap, Titanium, Sencha Touch et jQuery Mobile). Tandis que le document (***Raj al, 2012***), a montré les avantages et les inconvénients des différentes méthodes de développement mobile et les technologies proposées pour chaque cas, en fonction des propriétés qualitatives. Cependant, Dans (***Charland et al, 2011***) présentent une comparaison approfondie des applications Natives et du développement d'applications Web. Dans (***Heitkötter et al, 2012***) les auteurs présentent une étude comparative entre certains outils mobiles inter-plateformes basés sur plusieurs facteurs qualitatifs tels que les coûts de licence, le look-and-feel, les plateformes supportées, les environnements de développement,

la maintenabilité et l'évolutivité. Dans cette approche, la perspective multiplateforme n'est pas prise en compte. Dans (***Veldhuis, 2013***) présente une analyse comparative des performances des différents outils de développement mobiles, basée sur un simple calcul numérique. Cependant, dans (***Smutny, 2012***) les auteurs formulent une méthode pour évaluer et sélectionner les meilleurs outils de développement multiplateforme pour un développeur et aussi évaluer les outils multiplateforme en utilisant les critères : le temps, la technologie, la maturité et le coût du développement des applications mobiles. En revanche, ce travail est axé sur les outils de développement multiplateformes et ne présente pas un processus pour évaluer la méthode de développement appropriée à adopter (*natif, hybride ou web*).

Dans ce chapitre, nous avons présenté l'architecture et les caractéristiques de chaque méthode, et nous avons proposé une approche qui pourrait être adoptée pour choisir une méthode et un outil appropriés, afin de développer une application mobile.

Notre cadre se concentre sur l'amélioration de la prise de décision dans le domaine des applications mobiles, en tenant compte de plusieurs facteurs qualitatifs tels que le taux de développement, la documentation, l'apparence, la popularité, la courbe d'apprentissage et l'outil graphique pour la mise en place de l'interface graphique. Le cadre mentionné peut être divisé en deux étapes; La première permet de déduire la méthode de développement mobile alors que la deuxième permet de sélectionner l'outil adéquat pour chaque méthode dont la précision dépasse 50%.

1.6. Conclusion

Au terme de conclusion, nous avons présenté une brève description des plateformes mobiles leader du marché mondiale, les défis de développement mobile et les différentes approches de développement mobile. Ensuite, nous avons proposé un cadre décisionnel, permettant dans un premier temps, de sélectionner la méthode convenable pour mettre en place une application mobile bien spécifique, dans un deuxième temps, de classer les outils à utiliser pour développer l'application, et cela, en se basant sur un ensemble de critères jugés pertinents. Dans le chapitre suivant, nous présentons l'état de l'art sur les recherches réalisées dans le cadre des approches de développement des plateformes mobiles basées sur une approche native.

Chapitre 2

Etat de l’art

2.1. Introduction

Durant la dernière décennie, nous avons pu observer une croissance significative des ventes suite à la forte demande de Smartphones. L'utilisation de ces dispositifs et de leurs services devient de plus en plus populaire. Les gens peuvent facilement accéder à leurs boîtes aux lettres et répondre aux courriels avec leurs téléphones. En plus ces appareils offrent plusieurs services utiles, par exemple, les services de géolocalisation, les services de météorologiques ou des services d'information sur le transport etc. Toutefois, le développement des applications pour ces dispositifs nécessite des connaissances variées à compte tenu de la diversité des plateformes mobiles. Par conséquent, le développement d'une application devient une tâche épuisante. La variété des caractéristiques des périphériques, y compris: GUI, vitesse de traitement, capteurs, GPS, web service et les bases de données, sont d'autres contraintes qui doivent être prises en compte lors du développement des applications mobiles.

Dans ce contexte, les développeurs préfèrent de **mettre en œuvre une application mobile une seule fois et de la déployer pour de nombreuses plateformes avec un minimum d'effort et de temps**.

Dans ce chapitre nous présentons le modèle de développement multiplateforme et les travaux connexes ; en commençant par les travaux réalisés pour la définition des méta-modèles pour les plateformes mobiles comme Android et Windows phone, ensuite, nous détaillons les différentes approches de modélisation et de génération des applications mobiles multiplateformes.

2.2. Modèle de développement multiplateforme

L'objectif de toute entreprise de développement mobile est de cibler autant d'utilisateurs que possible en fournissant la même application pour différentes plateformes. Chaque fournisseur de plateformes fournit aux développeurs différents environnements de développement intégrés (IDE), langages de programmation, API et marché de distribution d'applications (store). Par conséquent, les sociétés de développement mobile doivent choisir l'une des deux alternatives pour développer la même application pour différentes plateformes. La première alternative est que les développeurs travaillent ensemble pour produire une application de qualité en un temps efficace et limiter sa distribution à une plateforme spécifique. Cette alternative soutiendra les différentes plateformes séquentiellement, mais perd beaucoup de temps dans le développement en apprenant et en se familiarisant avec les environnements de développement des différentes plateformes. Une autre alternative est que

les développeurs sont divisés en équipes distinctes et chaque équipe travaille pour une plateforme spécifique. Cette alternative prendra en charge les différentes plateformes en parallèle, mais peut exiger plus de développeurs que la première alternative donc elle est plus coûteuse. Le cycle de développement du logiciel traditionnel est illustré dans la figure ci-dessous.

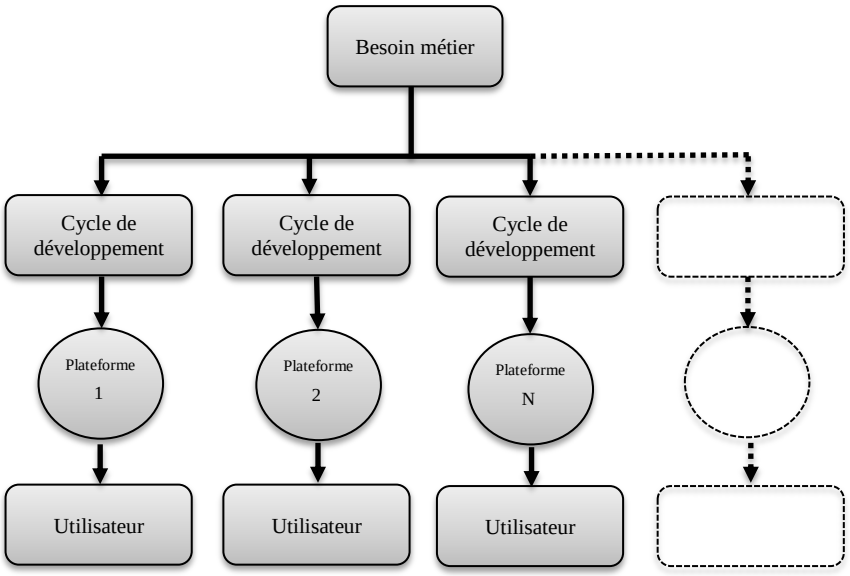


Figure 2. 1 - Modèle traditionnel de développement (*Corral et al, 2012*)

Cependant, il est toujours nécessaire de fournir l'application plus rapidement en économisant le temps et les efforts de développement. Ce problème conduit à l'existence des solutions de développement mobile multiplateforme. Le concept principal des solutions multiplateformes est de développer l'application une fois et de l'exécuter n'importe où.

Les solutions multiplateformes étendent le cycle de développement du logiciel en développant l'application une fois et en la déployant plusieurs fois pour supporter différentes plateformes, comme le montre la figure ci-dessous.

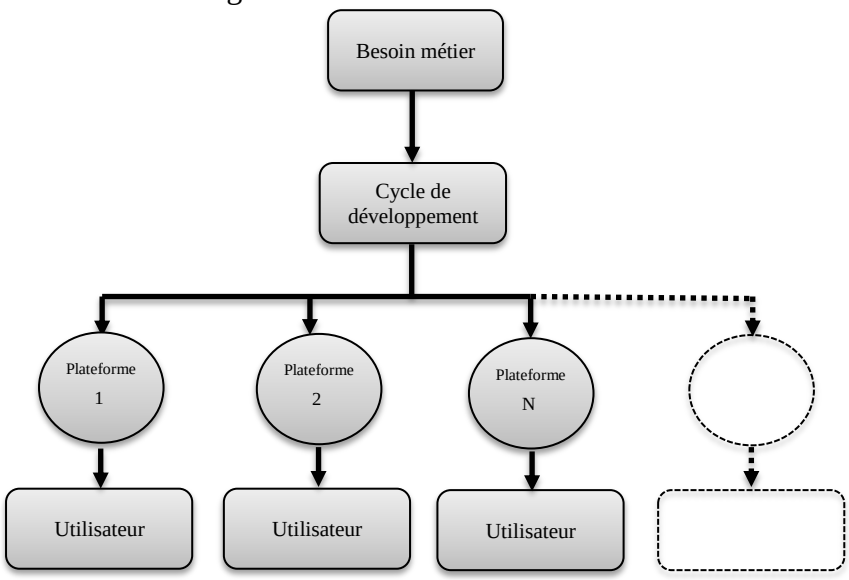


Figure 2. 2- Modèle multiplateformes de développement (*Corral et al, 2012*)

De nombreuses solutions de développement mobile multiplateforme sont toujours en recherche et développement. Certaines solutions sont disponibles dans l'utilisation commerciale mais **jusqu'à présent la solution finale qui résout le problème de développement multiplateforme mobile n'existe pas**.

2.3. Développement d'applications mobiles

Le développement des applications mobiles est un cas particulier du développement du logiciel car les développeurs doivent tenir compte de différents aspects comme le cycle de développement court, les capacités des périphériques mobiles, la mobilité, les spécifications des périphériques mobiles comme la taille de l'écran, la conception de l’interface graphique utilisateur, la navigation entre les écrans de l’application, la sécurité et le respect de la vie privée et les applications nécessitent de marketing pour augmenter leurs popularités. Une multitude d'applications sont disponibles dans les magasins pour aider à offrir une vie basée sur le mobile, ou mLife. Cela comprend le mTourisme, le mEnvironnement, la mEducation, le mBanking, la mHealth, le mEntertainment, etc.

Le cycle de vie du développement d'applications mobiles se compose des éléments suivants:

(1) analyse de l'idée de l'application ;
(2) la conception de l'interface utilisateur ;
(3) le développement de l'application à l'aide des outils et des langages de programmation de la plateforme cible ;
(4) le test de l'application sur différents appareils ; et enfin
(5) publier l'application sur le magasin de la plateforme cible.

De nouvelles fonctions ou mises à jour de l'application sont publiées dans les versions successives de cette application dans le magasin de plateforme cible. Pour développer l'application mobile pour de nombreuses plateformes, le cycle de développement ci-dessus est répété pour chaque plateforme cible à l'exception de la première étape de l'analyse des exigences de l’application.

Si le développeur souhaite développer la même application pour toutes les plateformes, les équipements de laboratoire suivants sont nécessaires: un ordinateur personnel (PC) et un Mac (utilisé pour le développement d'iOS Apps), les outils de développement de chaque plateforme et la configuration appropriée, au moins un dispositif de smartphone et un dispositif de tablette pour chaque plateforme et parfois de plusieurs dispositifs pour la même plateforme comme Android, parce qu'il est soutenu par beaucoup de vendeurs de smartphones

(*Perchat et al, 2013*). Ensuite, il est nécessaire de créer un compte de développeur pour chaque magasin (store) de plateforme pour publier les applications développées.

2.4. Approche de développement mobile multiplateforme basée sur l'ingénierie des modèles

L'approche MDA a su faire ses preuves pour le développement d'applications d'entreprises et peut également apporter beaucoup pour les applications mobiles. L'approche MDA peut nous aider à assurer la pérennité des savoir-faire, le gain en productivité tout en répondant aux problématiques de fragmentation des plateformes mobiles. Dans les sections suivantes, nous présentons quelques travaux connexes inscrits dans cette perspective, en commençant par une présentation des méta-modèles proposés pour modéliser les applications mobiles, suivi, des différentes approches de de génération de code ciblant les plateformes mobiles.

2.4.1. Méta-modélisation des plateformes mobiles

Les auteurs dans (*Madari et al, 2007*) présentent une étude d'une diversité de méta-modèles pour les IHM des plateformes mobiles telles que Symbian, JavaME et .NET, et une contribution à la mise en place d'un méta-modèle commun pour ces différentes plateformes mobiles. D'autre part, les auteurs ont discuté la définition d'une syntaxe abstraite pour les plateformes cibles.

Il existe un flux d'œuvres, qui proposent des méta-modèles pour les plateformes mobiles spécifiques telles que la plateforme Android (*Minhyuk et al, 2012*) et la plateforme Windows phone 7 (*Bup-Ki et al, 2011*). Dans ces œuvres, les auteurs ont contribué à des extensions de méta-modèle UML existant, en proposant des stéréotypes tels que (Activity, Manifest, Service, etc.) pour Android et (Page, Model, View, etc.) pour Windows phone, ces stéréotypes vont servir par la suite pour enrichir des modèles UML tel que le diagramme de classe, pour le rendre spécifique à la plateforme Android ou à la plateforme Windows phone. Ensuite, ce modèle spécifique va servir comme entrée pour un générateur de code comme AndroMDA (*AndroMDA, 2014*).

Le méta-modèle d'une application Android proposé par (*Minhyuk et al, 2012*) est présenté dans la figure ci-dessous :

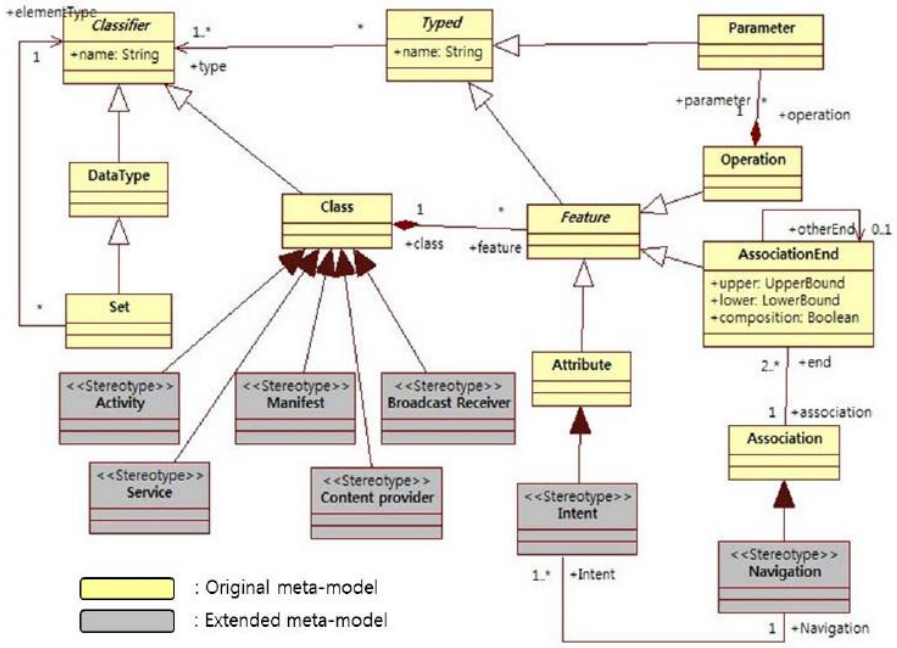


Figure 2. 3 - Extrait de méta-modèle Android (*Minhyuk et al, 2012*)

Le méta-modèle d'une application Windows phone 7 proposé par (***Bup-Ki et al, 2011***) est présenté dans la figure ci-dessous :

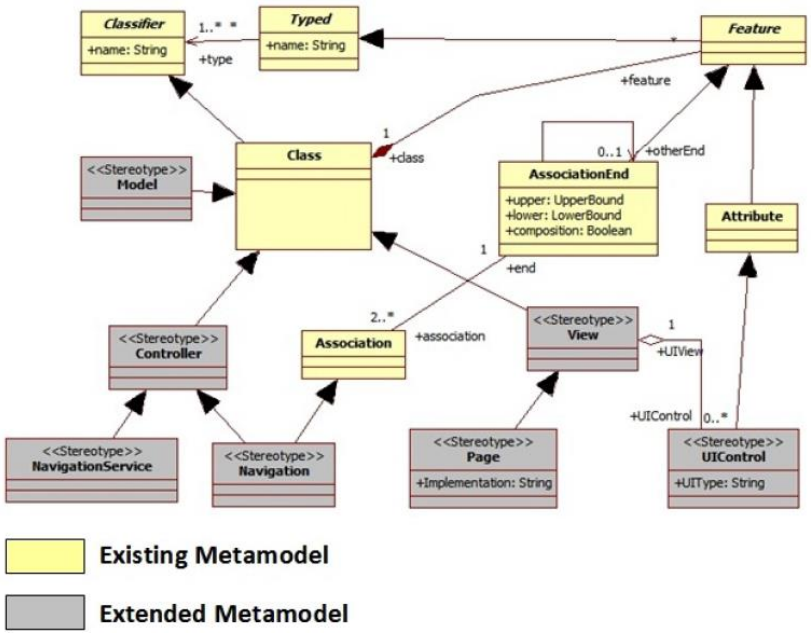


Figure 2. 4 - Extrait de méta-modèle Windows phone 7 (*Bup-Ki et al, 2011*)

Les auteurs dans (***Minhyuk et al, 2012***) ont également proposé des stéréotypes qui vont être utiles, pour modéliser des applications mobiles qui font recours aux fonctionnalités natives telles que la caméra, GPS, le réseau et les capteurs embarqués, etc. (*voir la figure ci-dessous pour plus de détails*).

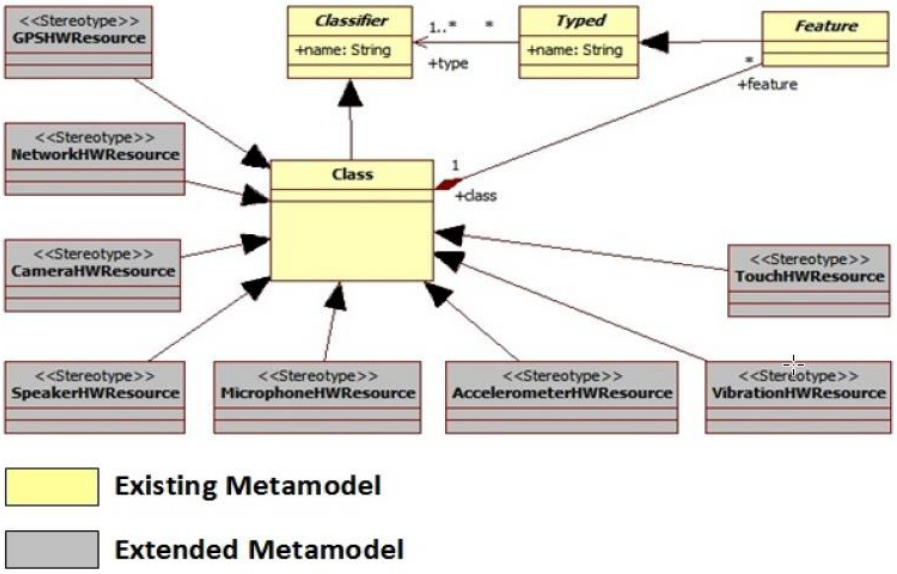


Figure 2. 5 - Méta-modèle pour les ressources matérielles (*Minhyuk et al, 2012*)

Les travaux présentent une étude des plateformes mobiles Android et Windows phone, et une extension du méta-modèle UML pour décrire précisément une application mobile Android ou Windows phone. D'autre part, les auteurs n'ont pas présenté une démarche pour la génération de code à partir de modèle PSM créé, également les auteurs n'ont pas présenté la structure des applications mobiles susceptible d'être générées à partir de modèle PSM (*Diagramme de classe enrichi par les différents stéréotypes*).

Dans (***Sabraoui et al, 2013***) les auteurs ont proposé un méta-modèle générique pour la conception des interfaces graphiques utilisateurs des applications mobiles (*voir la figure ci-dessous pour plus de détails*).

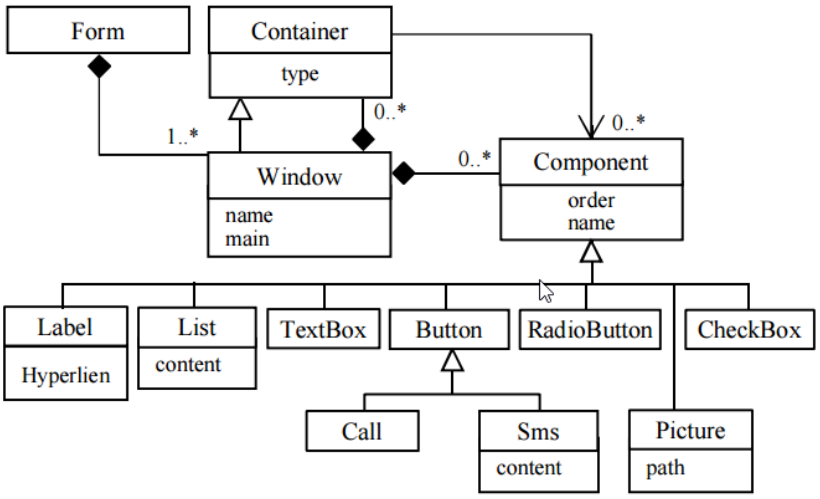


Figure 2. 6 - Méta-modèle indépendant pour les interfaces graphiques utilisateurs des applications mobiles (*Sabraoui et al, 2013*)

Le méta-modèle ci-dessous permet de modéliser l'interface graphique utilisateur d'une application mobile indépendamment de la plateforme cible. Cependant, ce méta-modèle reste

limité, en effet, il permet de concevoir des applications avec des interfaces graphiques simples sans prendre en charge des interfaces plus compliquées à mettre en œuvre et les évènements sur les différents composants.

2.4.2. Approches de modélisation et de génération code

2.4.2.1. Approches basées sur les modèles UML

Plusieurs travaux ont été effectués dans ce sens, les auteurs dans (***Juliano et al, 2011***) ont contribué à l'amélioration de la génération des interfaces graphiques de certaines plateformes telles que JSF et JSTL à l'aide du Framework open source AndroMDA (***AndroMDA, 2014***). L'approche décrite dans leur papier est basée sur l'analyse et la conception du modèle PIM à l'aide des digrammes UML, ensuite enrichi par des stéréotypes pour obtenir le modèle PSM, une fois que la transformation PIM vers le PSM est réalisée, AndroMDA génère le code spécifique de la plateforme cible. L'approche proposée permet de générer les applications selon une approche web.

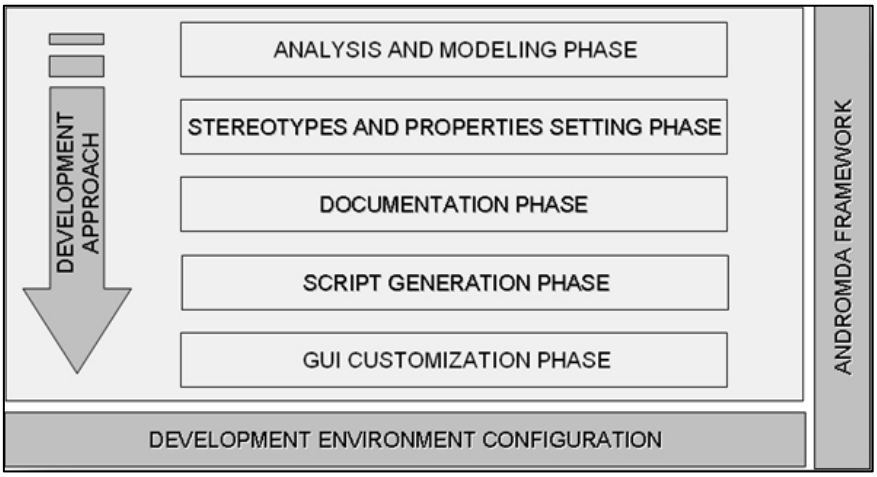


Figure 2. 7 - Approche pour la génération des interfaces graphiques avec AndroMDA (*Juliano et al, 2011*)

D'autres travaux de recherches tels que ceux de (***Link et al, 2008***) proposent des profils UML pour la modélisation des interfaces utilisateurs. Ces chercheurs ont exploité l'approche MDA en définissant un modèle indépendant de la plateforme, enrichi ensuite par des stéréotypes pour aboutir à un modèle spécifique à une plateforme en utilisant des transformations M2M. Ce dernier est transformé vers le code source suite à des transformations M2C (*Model to code*). Pour ce faire, les auteurs ont utilisé les digrammes UML pour définir le PIM et le QVT pour réaliser les différentes transformations. Cette approche est limitée à la génération des interfaces graphiques utilisateurs (*voir la figure ci-dessous pour plus de détails*).

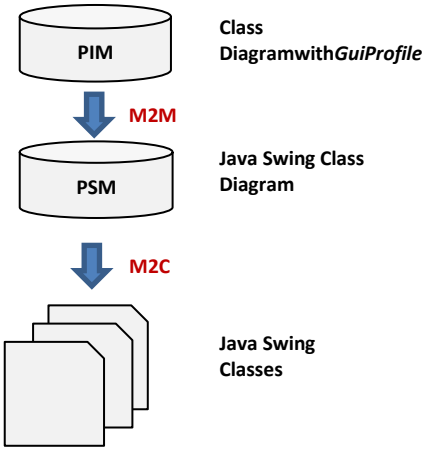


Figure 2. 8 - Approche pour la génération des interfaces graphiques en se basant sur les profils UML

Une approche MDA (***Sabraoui et al, 2013***) a été implémentée dans le but de modéliser et générer les interfaces graphiques des plateformes mobiles. Cette approche consiste en quatre principales étapes :

- Modélisation de l'interface graphique sous UML en utilisant un diagramme d'objet ;
- Transformation des diagrammes obtenus à un simple schéma XMI en utilisant l'API JDOM ;
- Transformation de nouveau modèle XMI vers le modèle spécifique à la plateforme cible ;
- Génération de l'interface graphique sur la base de l'approche MDA, en projetant dans des Template implémentés avec Xpand (*voir la figure ci-dessous pour plus de détails*).

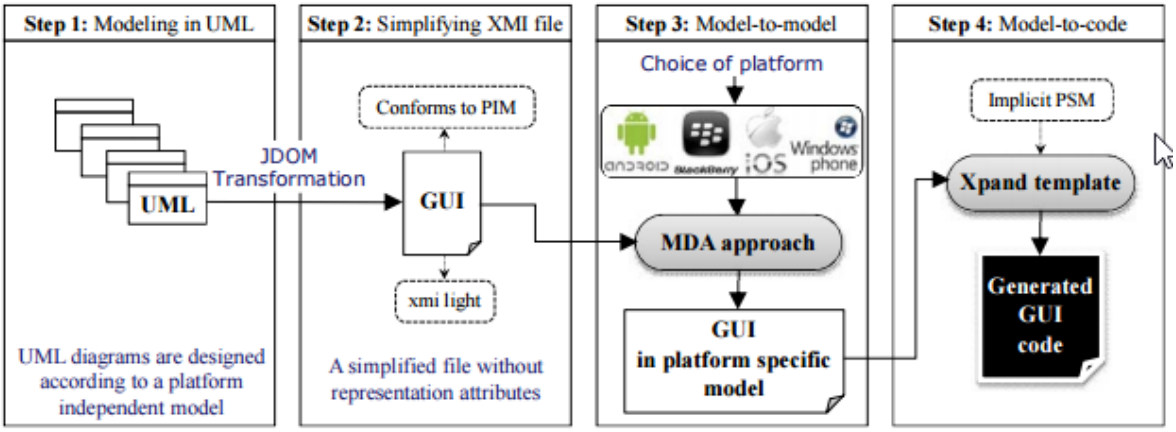


Figure 2. 9 - Approche pour la génération des interfaces graphiques en se basant sur JDOM et ATL (*Sabraoui et al, 2013*)

Cette méthode présente l'avantage de générer automatiquement les interfaces graphiques pour plusieurs plateformes mobiles à partir d'un modèle UML. Cependant, les développeurs dans cette approche ne peuvent pas concevoir l'interface utilisateur d'une manière plus simple et conviviale, en particulier dans le cas, où l'application nécessite plusieurs écrans. De plus,

l'utilisation du diagramme d'objets pour la modélisation de l'interface graphique prend beaucoup de temps. L'approche présentée reste limité à la génération des interfaces graphiques utilisateurs sans tenir compte des fonctionnalités natives offertes par les smartphones (e.g. GPS, caméra, capteurs, etc.), également, ne permet pas la génération des applications selon le principe de séparation de couche.

Les auteurs dans (***Benouda et al, 2016a, 2016b***) ont proposé une approche basée sur l'ingénierie des modèles visant la génération des interfaces graphiques utilisateurs des applications Android. Pour ce faire les auteurs ont utilisé le digramme de classe pour définir le PIM, le QVT pour réaliser les différentes transformations vers le PSM-Android et Acceleo pour la génération de code. Ce travail vise à accélérer et à faciliter le développement des applications Android. Il prend en compte la génération des interfaces graphiques, sans tenir compte de l'accès aux ressources, capteurs embarqués, interfaces graphiques plus compliquées et gestionnaires d'événements, etc. (*voir la figure ci-dessous pour plus de détails*).

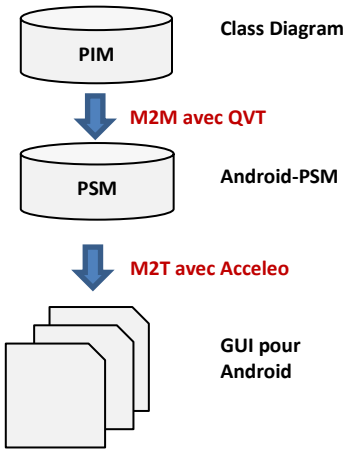


Figure 2. 10 - Approche pour la génération des interfaces graphiques en se basant sur le QVT et Acceleo

Un autre outil nommé AppliDE qui supporte la même approche est présenté dans (***Quinton et al, 2011***) (*voir la figure ci-dessous pour plus de détails*).

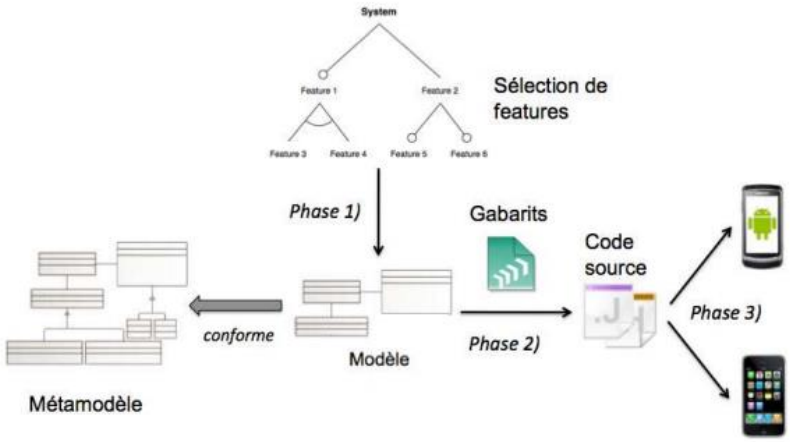


Figure 2. 11 - Processus de développement avec AppliDE

Cette approche est basée sur trois phases principales :

- ***Phase 1*** : Sélection des fonctionnalités, composition et modélisation de l'application. En utilisant des outils d'Eclipse Modeling Framework ;
- ***Phase 2*** : Génération de code. Les générateurs de code ont été développés sous la forme de gabarits Acceleo ;
- ***Phase 3 :*** Déploiement. Les fichiers source natifs générés (e.g. Objective-C pour iPhone, Java pour Android) sont exportés afin d'être utilisés dans leur projets respectifs. Chaque projet est ensuite compilé et exécuté dans son environnement de développement dédié. Les applications peuvent enfin être déployées sur les terminaux mobiles.

Dans ce travail, les auteurs n'ont pas discuté les différentes fonctionnalités supportées dans AppliDE, également, ils n'ont pas présenté, les méta-modèles sources et cibles, les règles de transformations et les différentes Templates utilisées.

Dans une autre étude (***Diep et al, 2013***) ont proposé un environnement MDD en ligne, fournissant aux développeurs une plateforme indépendante pour la conception des interfaces graphiques utilisateurs pour les applications mobiles. Cependant, cette solution permet de générer le code natif, pour la couche présentation, sans tenir compte de la génération des classes de traitement des données et les fichiers de configuration ainsi que des transitions entre les écrans (*voir la figure ci-dessous pour plus de détails*).

Figure 2. 12 - IDE en ligne pour concevoir les interfaces graphiques pour les applications mobiles multiplateformes basé sur MDD

Dans le même contexte, les auteurs dans (***Koji et al, 2014***) ont proposé une méthode MDD basée sur les diagrammes UML pour la génération des interfaces graphiques pour les

plateformes mobiles d'une manière intuitive (*voir figure ci-dessous pour plus de détails*).

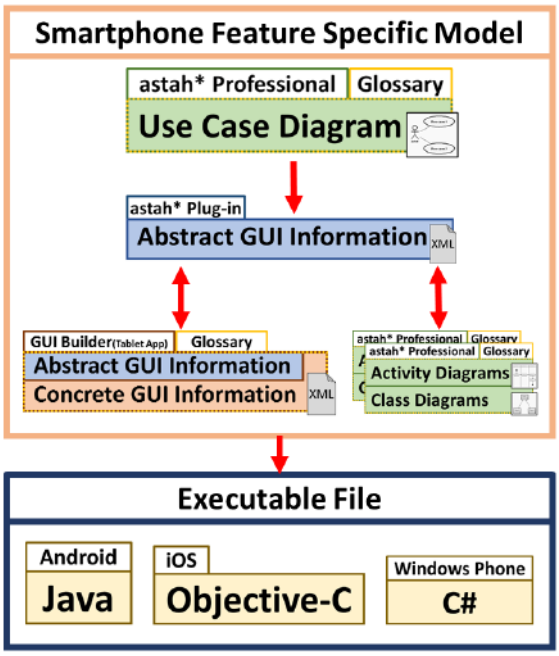


Figure 2. 13 - Présentation de la méthode (*Koji et al, 2014*)

Cette approche permet une flexibilité dans la conception d'une application au niveau de la vue structurelle et de la vue de l'interface utilisateur. Cependant, les auteurs sont toujours dans la phase de développement de générateur de l'interface graphique.

Les auteurs dans (***Parada et al, 2012***) ont proposé une approche pour la modélisation des applications Android selon une approche MDE. Pour ce faire, ils ont basé sur le diagramme d'activité et le diagramme de classe pour concevoir l'application, et le générateur GenCode (***Parada et al, 2011***) pour la génération des classes JAVA. Ainsi, ce travail est limité à la génération des classes JAVA et le code de navigation entre les écrans. Cependant, cet approche n'offre pas un mécanisme pour modéliser une application qui fait recours aux fonctionnalités natives, en outre, ne permet pas la génération des interfaces graphiques utilisateurs.

2.4.2.2. Approches basées sur le DSL

Dans (***Groenewegen et al, 2013***) les auteurs ont proposé un DSL Web pour l'intégration et la validation des données permettant ainsi d'avoir une syntaxe unifiée, des mécanismes de gestion des erreurs et une sémantique pour les contrôles de validation des données. Cela permettra aux développeurs des applications Web d'adopter un modèle centré, en se concentrant sur la conception logique d'une application plutôt que la complexité accidentelle de l'approche technique de mise en œuvre. Cependant, cette approche ne supporte que les applications web.

Les approches basées sur le DSL (***Madari et al, 2007***) et (***Mannadiar et al, 2010***) proposent des langages spécifiques au domaine pour la modélisation des interfaces graphiques des applications mobiles et un modèle graphique pour les règles de transformation pour mapper entre les modèles spécifiques au domaine (DSM) et les applications générées. Ainsi, ces approches nécessitent encore quelques améliorations pour générer automatiquement une application mobile.

Dans la figure ci-dessous le méta-modèle proposé dans (***Mannadiar et al, 2010***) pour la modélisation d'une application mobile en utilisant un DSL.

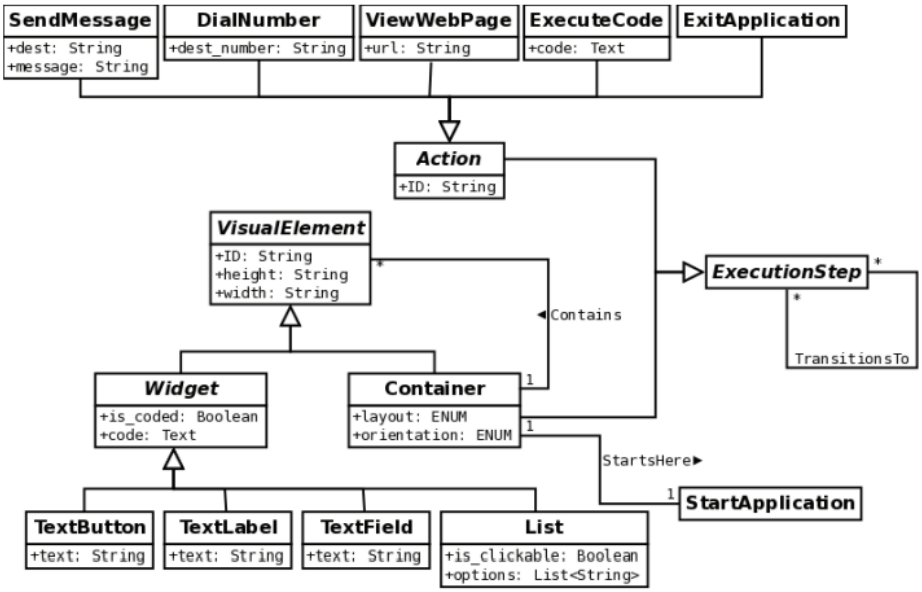


Figure 2. 14 - Méta-modèle des applications mobiles (*Mannadiar et al, 2010*)

Dans (***Raghu et al, 2012***) les auteurs ont proposé le Framework JSAF (JavaScript Application Framework) qui est une plateforme indépendante de modèle (PIM). JSAF permet de développer des applications web mobiles en utilisant les technologies HTML5, CSS 3 et JavaScript. Ce Framework offre un ensemble de widgets nécessaire pour la mise en place d'une application mobile (e.g. Box, Progress Bar, Scroller, Spinner, Menu, etc.). Cependant, cet outil ne supporte pas le développement des applications mobiles natives.

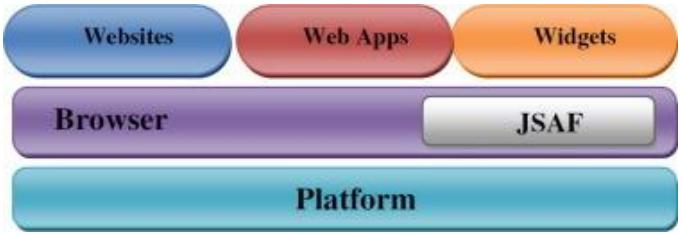


Figure 2. 15 - Architecture de la solution JSAF

En outre, Les auteurs dans (***Heitkötter et al, 2013***) ont proposé le Framework MD2, qui est basé sur un DSL adapté au domaine des applications mobiles. Cet outil permet de

développer des applications en décrivant le modèle d'application en utilisant le DSL, puis un ensemble de transformations sont effectuées pour générer le code source natif spécifique à la plateforme cible. Les applications créées avec MD2 suivent le modèle MVC. Le MD2 permet de:

- Définir les types de données et accéder aux opérations ;
- Définition des CRUD pour la mise à jour de données ;
- Implémenter l'interface utilisateur avec une variété de composants ;
- Définir les validateurs d'entrée sur les données ;
- Accès aux fonctionnalités natives telles que le GPS.

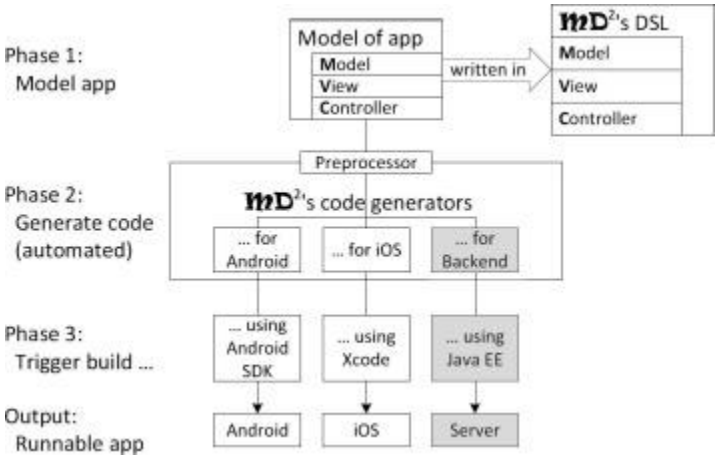


Figure 2. 16 - Architecture de Framework MD2

Les limitations de cette solution sont les suivantes :

- N'est pour le moment qu'un un prototype ;
- Elle se concentre sur une seule catégorie d'applications mobiles: les applications métier orientées données ;
- Elle se concentre sur la génération d'applications mobiles qui ne prennent pas en charge la réutilisation du code source existant ;
- Elle se concentre sur la génération de code pour les tablettes ;
- Les auteurs n'ont pas décrit le méta-modèle de base ;
- Avec le DSL nous ne pouvons pas générer une application mobile complète, ainsi, qu'une application qui respecte la programmation en couche.

2.5. Comparaison entre les approches de génération des applications mobiles

Nous pouvons classer les approches présentées ci-dessous en deux catégories:

(1) Des approches basées sur les modèles UML ;

(2) Et des approches basées sur le DSL, certaines approches permettent de générer le code d'une façon automatique, d'autres d'une façon manuelle ou semi-automatique.

Le tableau ci-dessous illustre une brève comparaison des différentes approches basées sur l'ingénierie dirigée par les modèles, pour la modélisation et la génération de code des applications mobiles.

Tableau 2. 1 – Comparaison entre les approches de génération des applications mobiles (*Lachgar et al, 2016*)

Approche	Plateformes sources	Plateformes cibles	Modélisation	Mappage	Type de construction	Degré d'automatisation	Année
Benouda Approach	Modèle UML	IHM et les classes Java pour Android	UML	QVT, Acceleo	Graphique	Automatique	2016
Koji Approach	Modèle UML	Code natif	Digrammes UML, GUI Builder	-	Graphique	Automatique	2014
Diep Approach	Modèle graphique pour l'IHM	Le code natif pour les IHM	Graphique (GUI)	DOM	Graphique	Automatique	2013
Sabraoui Approach	Modèle UML	IHM et la structure des classes	UML	ATL, DOM, Xpand	Graphique	Automatique	2013
MD2	Syntaxe abstraite	Code natif pour Android et iOs	DSL (Xtext)	Xpand	Textuel	Automatique	2013
Minhyuk Approach	Modèle UML	Code natif pour Android	Profiles UML	-	-	Manuel	2012
Raghu Approach	Modèle abstrait (HTML, CSS, JavaScript)	Code pour IHM et les CRUD pour la mise à jour des données	JSAF	-	Textuel	Automatique	2012
Parada Approach	Modèle UML	Classes Java	Profiles UML	GenCode	Graphique	Automatique	2012
Bup-ki Approach	Modèle UML	Windows phone 7	Profiles UML	-	-	Manuel	2011
Quinton Approach	Modèle UML	Android et iPhone	AppliDE	Acceleo	Graphique	Automatique	2011
Mannadiar Approach	Modèle abstrait	Android	DSL	Série de graphes	Graphique	Semi-automatique	2010
Link Approach	Modèle UML	Code native Pour les IHM	Profiles UML	QVT	Graphique	Automatique	2008
Madari Approach	Syntaxe abstraite	Code natif pour les IHM (JavaME, .Net, etc.)	DSL	VMTS	Textuel	Automatique	2007

2.6. Synthèse sur les approches de développement mobile basée sur l'IDM

La plupart des approches précitées permettent de générer en gros la couche présentation (IHM) et la structure de la couche logique (eg. Activity pour Android) d'une application mobile multiplateforme, sans tenir compte de la séparation de couches (eg. la couche métier, la couche accès aux données, la couche accès aux services réseaux, etc.).

Toutefois, aucune de ces approches, n'offre pas un méta-modèle pour modéliser une application qui accède aux fonctionnalités natives d'un smartphone (eg. GPS, camera, capteurs embarqués, etc.), et un générateur de code basé sur le méta-modèle proposé.

Ainsi, nous résumons les limites de ces approches ci-dessous :

- Manque d'un méta-modèle générique permettant de modéliser une application mobile complète ;
- Manque d'un générateur de code permettant la génération d'une application mobile respectant la programmation en couche ;
- Manque d'un méta-modèle pour la modélisation des fonctionnalités natives et un générateur de code natif basé sur ce méta-modèle ;
- Manque d'un méta-modèle pour modéliser les transitions entre les écrans ;
- Manque d'une description détaillé des règles de transformations utilisées lors de l'implémentation des générateurs de code ;
- Manque d'un générateur de code pour générer à la fois une application métier orientée données (e.g. application de gestion) et une application multimédia (e.g. Intégration des vidéos et audio) ;
- Manque d'un méta-modèle pour la génération des services distants ;
- Manque d'un méta-modèle pour la génération et la manipulation des bases de données embarquées.

Dans notre projet de thèse nous avons essayé **dans un premier lieu** de proposer un méta-modèle pour la modélisation des interfaces graphiques des applications web et mobile basées sur les composantes, et de mettre en place un générateur de code pour les interfaces graphiques. **En deuxième lieu**, nous avons enrichi le méta-modèle pour le rendre spécifique aux plateformes mobiles, ensuite, nous avons amélioré le générateur de code pour générer une application mobile qui fait recours aux fonctionnalités natives. Enfin, nous avons combiné entre la modélisation UML et le DSL pour modéliser une application développée selon une architecture en couche.

2.7. Conclusion

Dans ce chapitre, nous avons commencé par présenter le modèle de développement multiplateforme, les méta-modèles existants pour la modélisation des applications mobiles et les différentes approches existantes pour la génération de code. Ensuite, nous avons classé, ces approches en deux catégories, (1) des approches basées sur UML, et (2) des approches basée sur le DSL. Toutefois, nous avons présenté les limites de ces propositions et l'extension ou les améliorations qui peuvent être apportées par notre démarche.

Dans le chapitre suivant, nous présentons les différents concepts liés à l'ingénierie dirigée des modèles.

Chapitre 3

L’ingénierie dirigée par les modèles – IDM

“Modéliser est le futur, et je pense que les sociétés qui travaillent dans ce domaine ont raison"

B. Gates

3.1. Introduction

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 de modèles (***El Hamlaoui et al, 2015***).

Dans ce chapitre nous présentons dans un premier temps la notion de modèle, 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, suivi, de la démarche à suivre pour mettre en place un méta-modèle à l'aide d'un DSL. Nous sommes focalisés particulièrement sur les trois 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 et le processus de développement dans MDA.

3.2. Ingénierie Dirigée par les modèles *(Model Driven Engineering – MDE)*

L'ingénierie dirigée par les modèles (IDM), ou Model Driven Engineering (MDE) en anglais, terme proposé par (***Kent, 2002***), est une famille d'approches de développement logiciel basé sur l'utilisation de modèles dans la mise en place de logiciels. Elle permet l'exploitation des modèles pour simuler, estimer, comprendre, communiquer et produire du code (***Bézivin, 2005***). De manière plus générale, l'ingénierie dirigée par les modèles offre un cadre méthodologique et technologique qui permet d'unifier différentes façons de faire dans un processus homogène (***Jézéquel et al, 2012***). Il est ainsi possible d'utiliser la technologie la mieux adaptée à chacune des étapes du développement, tout en ayant un processus global de développement qui est unifié dans un paradigme unique (***Favre et al, 2006***).

Les résultats recueillis au cours des dernières années ont montré les avantages du MDE par rapport à l'approche traditionnelle de développement en termes de qualité et de productivité (***El Hamlaoui et al, 2015***):

- *Qualité* : une réduction globale de 1, 2 à 4 fois du nombre des anomalies et une amélioration des anomalies de 3 fois en phase de maintenance d'anomalies. Le coût global de la qualité a également baissé en raison d'une diminution des temps d'inspection et de test ;
- *Productivité* : une amélioration de la productivité de 2 à 8 fois en terme de lignes de code source.

3.2.1. Notion de Modèle

Un modèle est une simplification d'un système intégré avec un objectif visé à l'esprit. Le modèle devrait être en mesure de répondre aux questions à la place du système réel (***Bézivin, 2001***). Dans l'ingénierie des modèles on peut définir un modèle comme une description (*d'une partie*) d'un système écrit dans un langage bien défini (***Kleppe et al, 2003***), qui sera prescrit à l'aide d'un méta-modèle.

La modélisation d'un modèle ne rend pas celui-ci plus formel. Par exemple ce n'est pas en modélisant un avion que l'avion s'en voit mieux connue. Une description (*formelle ou informelle*) d'un avion permet d'obtenir un modèle de cet avion (*qui joue ainsi le rôle de système modélisé*); l'avion jouant à son tour le rôle de modèle par rapport à l'avion réel.

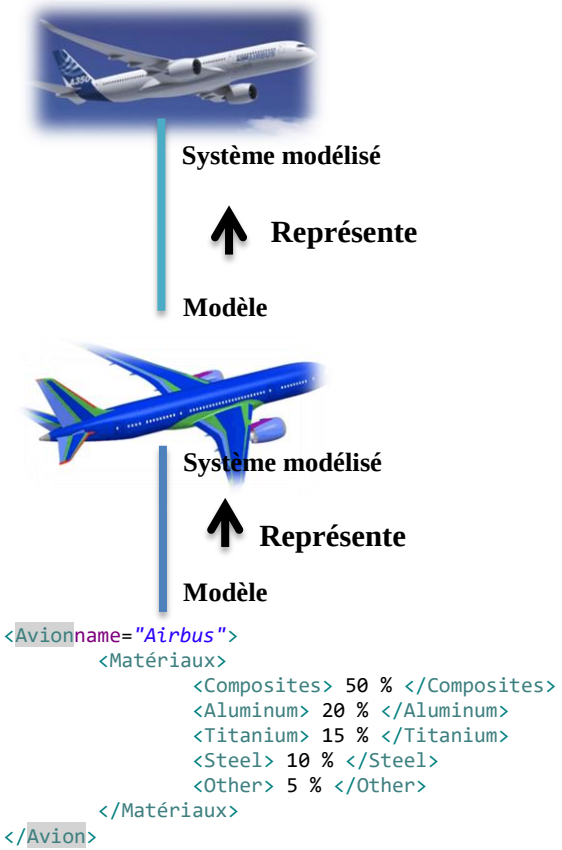


Figure 3. 1 - Modèles et système modélisé

La relation de modélisation a été combinée deux fois, l'avion est un modèle de l'avion réel, et le fichier XML est un modèle de l'avion, pourtant on n'a pas fait intervenir la notion de langage ou de méta-modèle. Le modèle en XML n'est pas un méta-modèle de l'avion. C'est un modèle d'un autre modèle.

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

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 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 :

- *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 ;

- *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 ;
- *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 Frameworks les plus utilisés comme Struts, Spring, Hibernante, .Net, Java, etc. ;
- *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 transformations successives jusqu'au code dans un langage spécifique (e.g. JSF2, EJB3, Struts, etc.).

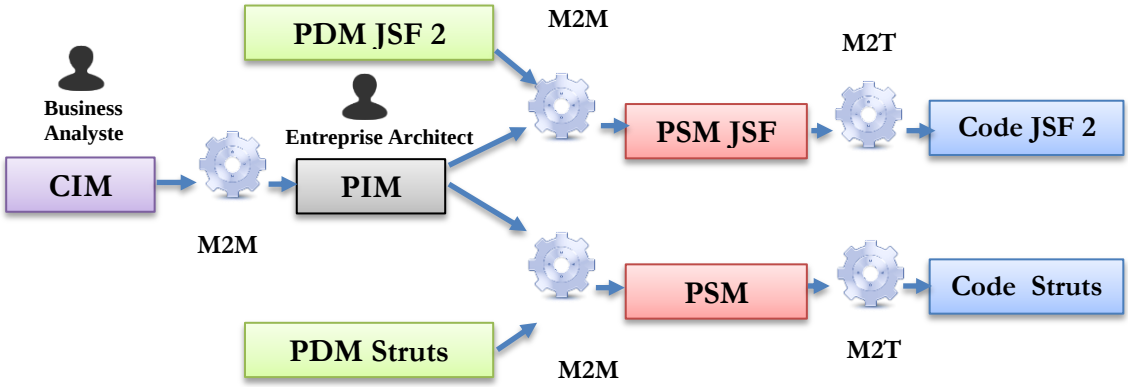


Figure 3. 2 - Exemple d'utilisation des modèles dans l'ingénierie vers l'avant (Forward engineering)

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 et al, 2010***) par exemple, tout d'abord, le modèle CIM est transformé en modèle PIM. Ensuite, 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***).

3.2.3. 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 3. 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 «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-méta-modèle).

Pour assurer une cohérence des niveaux d'abstraction, l'OMG a défini une architecture de modélisation sur quatre niveaux (*voir la figure "Figure 3.4" 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 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.

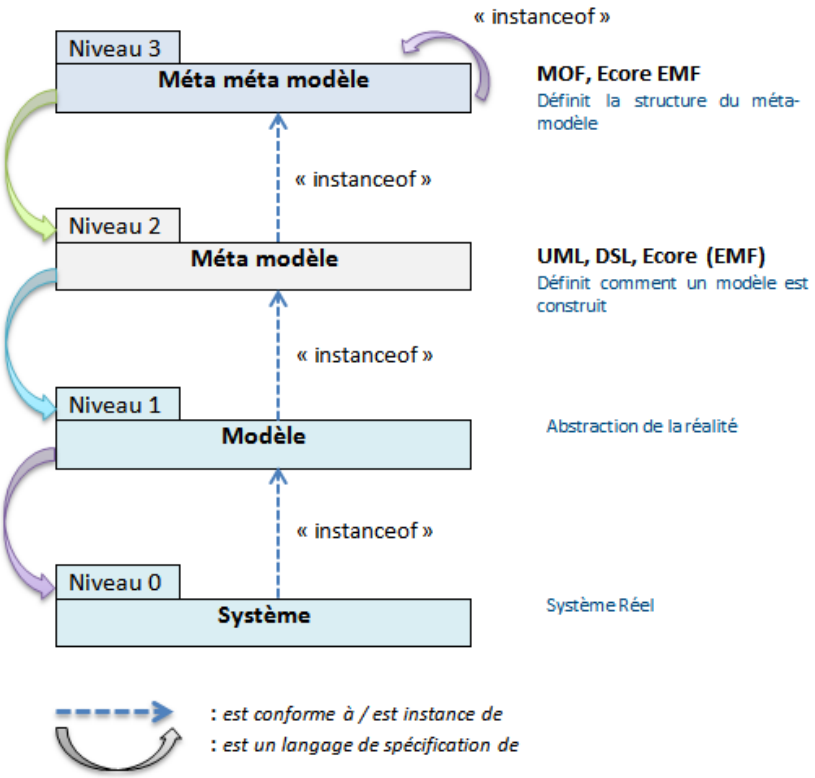


Figure 3. 4 - Architecture à 4 niveaux d'abstraction

3.2.4. 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.

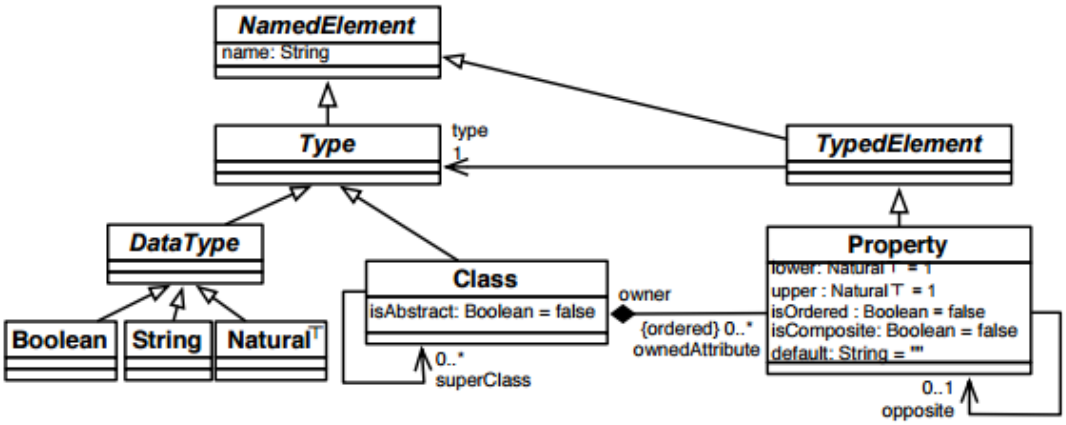


Figure 3. 5 - Représentation de MOF 2.0 sous forme de diagramme de classes

- *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.

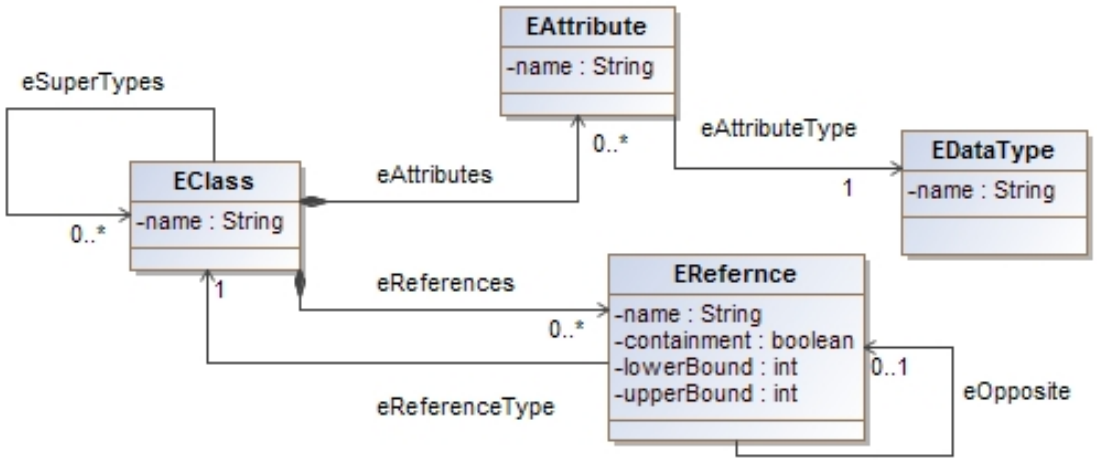


Figure 3. 6 - Extrait du Méta-Modèle Ecore

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

3.2.5. 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.

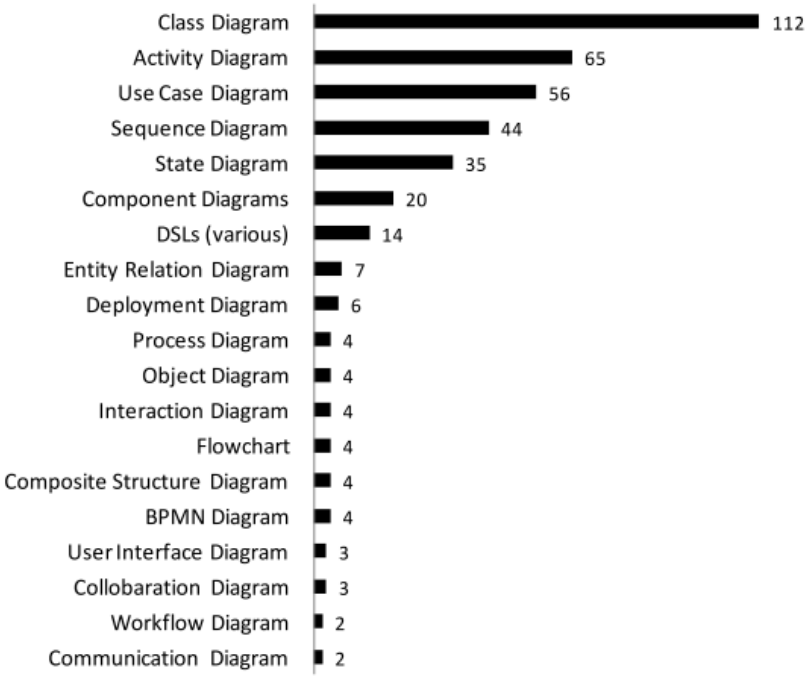


Figure 3. 7 - Liste des diagrammes utilisés régulièrement en UML (*Hutchinson et al, 2014*)

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.

3.2.6. Langage dédié (ou méta modèle)

3.2.6.1. 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.

3.2.6.2. Etapes 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.

a. 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.

b. 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 réécriture et les machines d'état pour la spécification de la sémantique.

c. 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 templates 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 exprimée 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.

d. 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.

3.2.7. Outils pour la définition des DSLs autonomes

3.2.7.1. Xtext

Xtext[1] 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.

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

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.

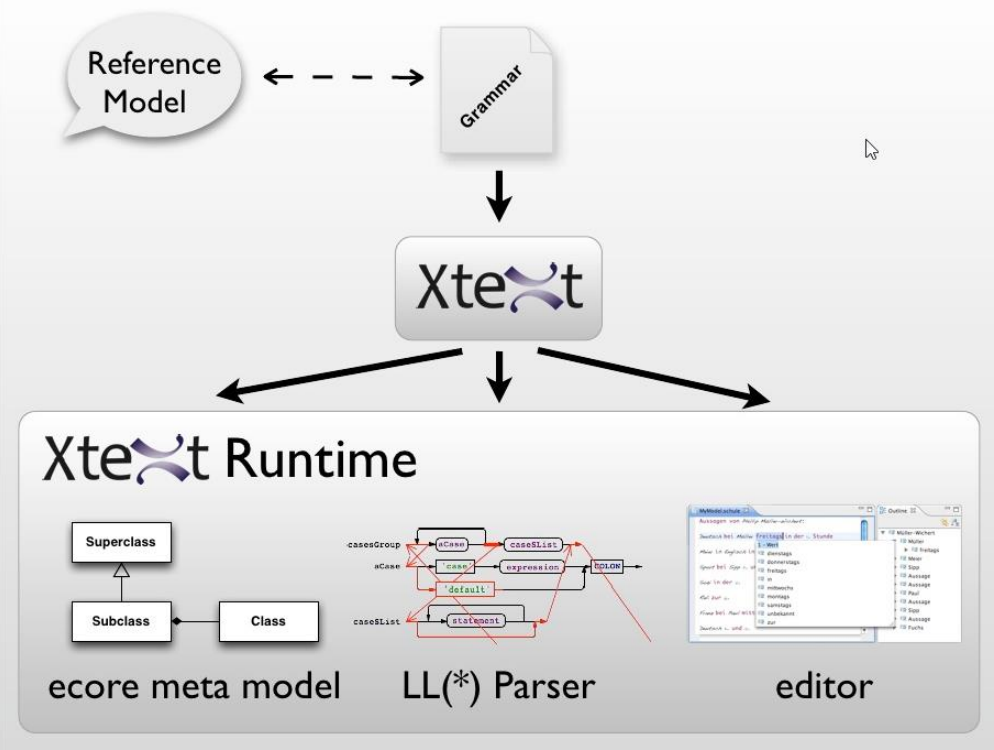


Figure 3. 8 - Vue d'ensemble de l'outil Xtext

3.2.7.2. Spoofax

Spoofax[2] est aussi basé sur Eclipse, mais ne s'appuie pas sur EMF. C'est un système développé par TU Delft[3] 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.

3.2.7.3. JetbrainsMPS

JetBrains MPS[4] 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

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

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.

3.2.8. 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.

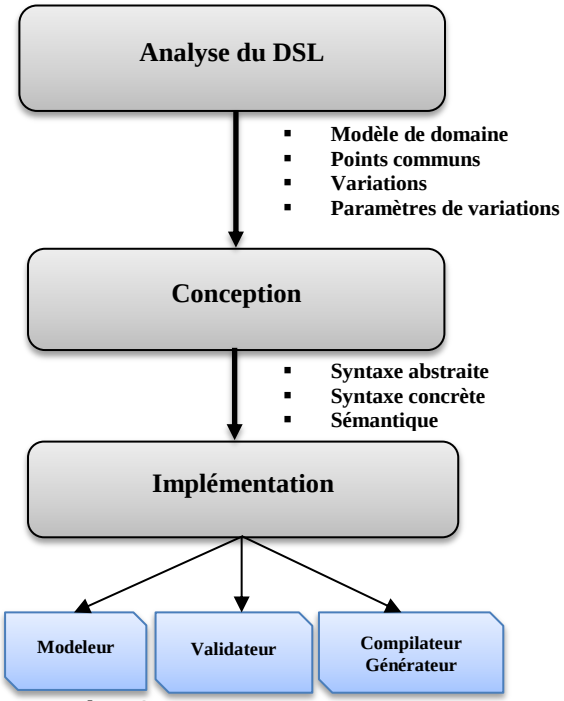


Figure 3. 9 - Processus de développement des DSL

3.2.9. Transformation des modèles

3.2.9.1. Définition

La définition la plus générale et qui fait l'unanimité au sein de la communauté IDM (***Bézivin, 2004***) 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. Dans l'approche par modélisation, cette transformation se fait par l'intermédiaire de règles de transformations qui décrivent la correspondance entre les entités du modèle source et celles du modèle cible. En réalité, la transformation se situe entre les méta-modèles sources et cibles qui décrivent la structure des modèles cibles et sources. Le moteur de transformation de modèles prend en entrée un ou plusieurs modèles sources et crée en sortie un ou plusieurs modèles cibles.

Une transformation des entités du modèle source met en jeu deux étapes. La première étape permet d'identifier les correspondances entre les concepts des modèles source et cible au niveau de leurs méta-modèles, ce qui induit l'existence d'une fonction de transformation applicable à toutes les instances du méta-modèle source. La seconde étape consiste à appliquer la transformation du modèle source afin de générer automatiquement le modèle cible par un programme appelé moteur de transformation ou d'exécution. La figure "Figure 3.10" ci-dessous illustre ces deux étapes d'une transformation de modèles.

Nous distinguons trois types de transformation :

- La transformation d'un modèle vers un autre modèle (Model To Model – M2M) ;
- La transformation d'un modèle vers un texte (Model To Text – M2T) ;
- et la transformation d'un texte vers un modèle (Text To Model – T2M).

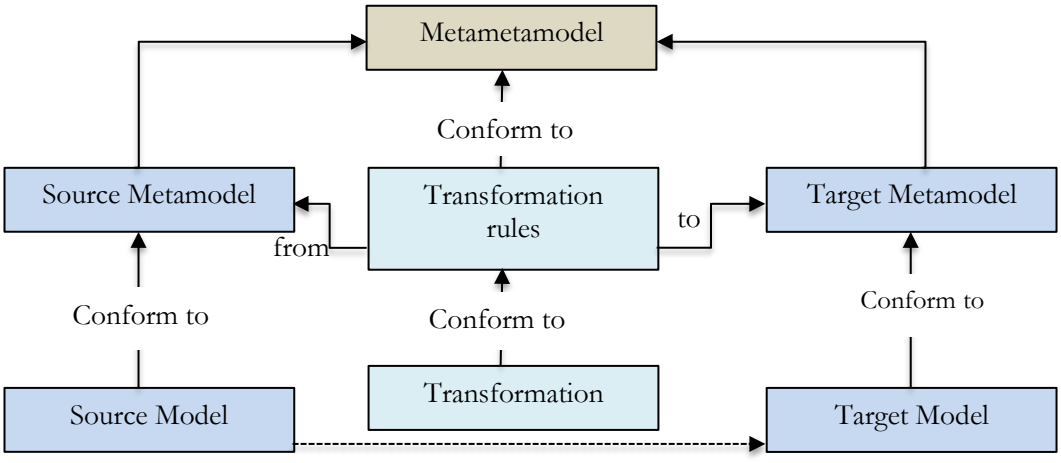


Figure 3. 10 - Schéma de base d'une transformation de modèles (*staff, 2008*)

Parmi les langages de transformation M2M on peut citer : QVT (***OMG, 2015***), ATL

(*Jouault et al, 2008*), VIATRA2 (*Varró et al, 2007*), Xtend (*Bettini, 2016*), etc. Par exemple, le travail de (*Pérez-Martínez et al, 2006*) décrit la transformation d'un modèle d'analyse vers un modèle d'architecture logicielle. Dans les transformations M2M, Mens et al. (*Mens et al, 2006*) distinguent une transformation «endogène» d'une transformation «exogène». Une transformation est endogène quand les méta-modèles des modèles source et cible sont identiques alors qu'elle est exogène quand ils sont différents. Parmi les langages de transformations M2T, on peut citer JET (*Eclipse, 2011*), Xpand (*Klatt, 2007*), Acceleo (*Musset et al, 2006*), Xtend (*Bettini, 2016*), etc. Le travail de (*Albert et al, 2006*) par exemple, décrit la transformation d'un modèle UML vers du code C#. Concernant la transformation T2M, elle est effectuée par une série d'étapes. Le texte est analysé en un modèle conforme au méta-modèle de l'arbre syntaxique abstrait (Abstract Syntax Tree Metamodel - ASTM) (*OMG, 2011a*) puis transformé vers un modèle conforme au méta-modèle de découverte de connaissances (Konwledge Discovery Metamodel - KDM) (*OMG, 2011b*). Ce dernier est transformé par la suite vers le modèle final. Le travail présenté dans (*Barbier et al, 2011*) décrit la transformation d'un code COBOL vers un modèle UML.

3.2.9.2. Principales approches de transformation de modèles

Dans (*Blanc et al, 2011*) trois approches de transformations de modèles sont retenues: l'approche par programmation, l'approche par template et l'approche par modélisation.

- *L'approche par programmation :* consiste à utiliser les langages de programmation en général, et plus particulièrement les langages orientés objet. Dans cette approche, la transformation est décrite sous forme d'un programme informatique à l'image de n'importe quelle application informatique. Cette approche reste très utilisée car elle réutilise l'expérience accumulée et l'outillage des langages existants.
- *L'approche par template :* consiste à définir des canevas des modèles cibles souhaités. Ces canevas sont des modèles cibles paramétrés ou des modèles template. L'exécution d'une transformation consiste à prendre un modèle template et à remplacer ses paramètres par les valeurs d'un modèle source. Cette approche par template est implémentée par exemple dans Softeam MDA Modeler.
- *L'approche par modélisation :* consiste quant à elle à appliquer les concepts de l'ingénierie des modèles aux transformations des modèles elles-mêmes. L'objectif est de modéliser les transformations de modèles et de rendre les modèles de transformation pérennes et productifs, en exprimant leur indépendance vis-à-vis des plateformes d'exécution. Le standard MOF 2.0 QVT de l'OMG a été élaboré dans ce cadre et a

pour but de définir un méta-modèle permettant l'élaboration des modèles de transformation de modèles. A titre d'exemple, cette approche a été choisie par le groupe ATLAS à travers son langage de transformation de modèles ATL.

3.2.9.3. Outils de transformation des modèles

a. Standard XMI

Le standard XML Metadata Interchange (XMI) (***OMG, 2014***) définit le format d'échange et de stockage de modèles via le langage eXtensible Markup Language (XML). XMI repose principalement sur deux autres standards XML du W3C et le MOF de l'OMG.

XMI permet la sérialisation de modèles et la génération de documents DTD et schémas XSD à partir des méta-modèles correspondants. En fait, XMI est une représentation textuelle en XML du modèle selon le méta-méta-modèle MOF. Cette représentation permet d'assurer les échanges de modèles entre les différents outils AGL du marché.

L'utilisation de XMI est incontournable dans les outils AGL de modélisation, malgré que les versions de ce dernier enregistrent un déphasage par rapport aux versions d'UML.

b. Standard QVT

Le QVT (Query/View/Transformation) (***OMG, 2015***) est le langage de transformation utilisé dans l'approche MDA normalisé par l'OMG. Toutefois, dans la spécification initiale du MDA, l'OMG a cherché un standard de transformation de modèle compatible avec la MDA. Pour cette raison, depuis 2002, l'OMG a émis un appel de propositions ayant pour but de normaliser la définition des transformations de modèles autour du standard MOF : le RFP (*Request for Proposal*). Ce RFP a concerné la définition de trois langages :

- Un langage permettant la spécification de règles de transformations de modèles afin de générer un modèle cible à partir de modèle(s) source(s) ;
- Un langage de requêtes qui permet une navigation dans les modèles ;
- Un langage d'élaboration de vues qui permet de construire des vues partielles sur des modèles.

D'autre part, la spécification du standard QVT est hybride (impérative et déclarative). Les parties déclaratives de QVT nommées Relations Language permet la création de spécification déclarative des relations entre des modèles MOF. Dans QVT définir une transformation déclarative revient à spécifier comment un ensemble de modèles peut être transformé en un autre. En plus du langage déclaratif, QVT dispose d'un langage opérationnel pour implémenter des parties impératives aux transformations. En fait, la partie impérative d'une

règle de transformation est une extension du langage OCL. Elle permet d'avoir un style plus procédural et une syntaxe concrète qui ressemble à celle des langages de programmation.

Toutefois, le langage QVT ne fournit pas de spécification précise au langage de vues prescrit dans la RFP. La spécification actuelle de QVT concerne uniquement la définition des transformations modèle-à-modèle. Tandis que les transformations modèle-à-texte est en cours de développement. Par ailleurs, ce langage permet uniquement de réaliser des transformations plusieurs-à-un, ce qui rend ce standard encore peu utilisé. Différents langages de transformation implémentent QVT tels que Medini QVT (***Höbler et al, 2006***), OptimalJ (***Jonkers et al, 2007***) et principalement ATL (***Allilaire et al, 2006***). Ce dernier serait choisi comme langage de transformation qui supportera l'ensemble des transformations de modèles proposées dans notre approche.

c. ATL

L'ATL est développé dans le cadre du projet ATLAS de l'Université INRIA de Nantes par l'équipe de Jean Bézivin (***Allilaire et al, 2006 ; Jouault et el, 2008***). Il fait partie de la plateforme AMMA (Architecture ATLAS Model Management) et il est intégré en tant que plug-in dans Eclipse Modeling Framework (EMF). ATL est inspiré du langage QVT (***OMG, 2015***) et s'appuie sur le formalisme OCL (***OMG, 2006***). En effet, l'ATL est un langage de transformation hybride, il est à la fois déclaratif et impératif. ATL permet de réaliser des transformations entre les modèles source et cible par le biais d'un ensemble de règles de correspondance ou de mappage écrites dans ce langage ATL. Une règle de correspondance décrit la conversion entre les concepts des modèles sources et cibles représentées dans le format XMI ou dans la notation KM3 (***Jouault et al, 2006***). En ATL, nous pouvons créer des modules, des requêtes ou bibliothèques. Un module permet principalement d'effectuer des transformations modèle-à-modèle. Tandis qu'une requête permet de réaliser des transformations modèle-à-texte.

Enfin, le langage ATL donne aussi la possibilité de développer des bibliothèques ATL communes qui peuvent être importées et partagées par différents modules et requêtes de transformation ou par d'autres bibliothèques également.

La structure d'un module ATL se compose de quatre parties: un entête, une section d'importation, les règles de correspondances et les helpers. L'entête est utilisé pour spécifier les modèles source et cible ainsi que leurs méta-modèles respectifs. La section d'importation, qui est facultative, permet aux développeurs de bénéficier de certaines bibliothèques existantes. Les règles de correspondances à leur tour, sont au cœur des modules ATL. En fait,

il existe deux types de règles dans ce langage (***Atlas, 2006***) :

- *Règle de correspondance:* Principalement, elles sont utilisées dans la partie du code déclarative de la règle. Elles décrivent comment les éléments cibles peuvent être générés à partir d'éléments sources et comment leurs propriétés doivent être initialisées. Une règle de correspondante a un nom et un et un seul élément source du modèle(s) en entrée, même si elle peut générer plus qu'un élément cible ;
- *Règle appelée:* Une règle appelée peut être sollicitée par une section du code impératif d'une règle de correspondance ou à partir d'une autre règle appelée. Les règles appelées peuvent être considérées comme un type particulier de helpers. A l'encontre des règles de correspondances, les règles appelées peuvent comporter des paramètres.

```
Module M2M;
create OUT : MMS from IN : MMC;

helper context IN!Element_target def : getNom(): String =…;
rule Element_Target2Element_Source{
      from s:MMC!Element_Source
      to
      t:MMS!Element_cible(t.property←c.property,…)
}
```

Figure 3. 1. Structure basique d'un module en ATL

Enfin, les helpers permettent de créer des méthodes de la même manière qu'en programmation orientée objet. Ils sont définis sur un contexte donné et ils s'appliquent à toute expression ayant comme type ce contexte. Un helper en ATL peut être appelé à différents endroits d'un module ou d'une requête ATL. On peut citer l'exemple de helper consacré à la conversion de la première lettre d'une chaine de caractère en majuscule. Ce helper peut être revendiqué lors de la génération du code d'applications développées en Java, où les noms de classes commencent par une lettre en majuscule conformément à la convention Java Specification Requests (JSR).

d. Kermeta

Le langage Kermeta a été initié par Franck Fleurey en 2005 au sein de l'équipe Triskell de l'IRISA (regroupant des chercheurs de l'INRIA, le CNRS, l'INSA et l'Université de Rennes (***Fleurey et al, 2006***). Le langage Kermeta emprunte des concepts de langues tels que le MOF, OCL et QVT, mais aussi de BasicMTL, un langage de transformation de modèles mis en œuvre en 2004 dans l'équipe Triskell par D. Vojtisek et F. Fondement. Kermeta, et sa plateforme d'exécution sont disponibles sous Eclipse. Il est open-source, sous licence EPL (Eclipse Public Licence).

Il constitue une alternative intéressante, car la nature de son langage permet de définir des transformations complexes tout en bénéficiant de fonctionnalités logicielles avancées (e.g., programmation par aspect) et en conservant une indépendance vis-à-vis de l'implémentation concrète du méta-modèle.

3.2.9.4. Synthèse

Comparant le QVT et ATL, ATL avec sa nature hybride représentée par une programmation déclarative et impérative, ce qui le rend plus expressif et lui accorde la possibilité d'exprimer toute sorte de transformations. En ce qui concerne les performances ATL dans la plupart des cas s'exécute plus vite que QVT grâce à deux raisons principales ; la première : il est plus facile de réduire la mise en correspondance avec la clause WHERE dans les règles ; la deuxième : est le fait que ATL est compilé et exécuté sur une machine virtuelle.

En outre, l'étude menée par (***Farhana Islam et al, 2013***) à l'université d'Ottawa, montre bien que ATL est performant par rapport à Kermeta sur plusieurs axes (*voir le tableau "Tableau 3.1" ci-dessous pour plus de détails*).

Tableau 3. 1 - Comparaison entre le langage ATL et le langage Kermeta (*Farhana Islam et al, 2013*)

Critère	Kermeta	ATL	Résulta
Courbe d'apprentissage	Plus simple à apprendre, sa syntaxe est similaire à JAVA et Eiffel.	Des connaissances sur la syntaxe OCL et la syntaxe ATL sont exigés.	Kermata gagne
Coût	Open source et gratuit.	Open source et gratuit.	Tous les deux gagnent
Portabilité	▪ Disponible pour Windows, Linux and Mac OS ; ▪ Prend en charge 32bit et 64bit.	▪ Disponible pour Windows, Linux and Mac OS ; ▪ Prend en charge 32bit et 64bit.	Tous les deux gagnent
Support technique	▪ Soumettre des rapports de bogues et des demandes de fonctionnalités via la section «*Tracker*» de leur site Web ; ▪ Les forums publics ; ▪ Emails.	▪ Par le biais des forums et des groupes de discussion officiels d'Eclipse ; ▪ Bugzilla ; ▪ Pour un support professionnel: développeurs d'OBEO.	ATL gagne
Documentation	▪ Manuel d'utilisation et guide du développeur, FAQ, articles et documents, tutoriels et cours sont disponibles ; ▪ Des études de cas de transformation de modèles insuffisantes sont fournies.	▪ Manuel d'utilisation et guide du développeur, FAQ, articles et documents, tutoriels et cours sont disponibles ; ▪ Formation professionnelle ; ▪ Nombreux modèles de transformation de modèles sont disponibles.	ATL gagne
Outil de test	Comprend Kunit: Un cadre de test unitaire.	Un très grand nombre d'installation de test sont disponibles (e.g. ATLtest, Octopus etc.).	Tous les deux gagnent

3.2.8. Outils pour la génération de code

Les outils axés sur la génération d'une sortie textuelle à partir d'un modèle (M2T) tel que

Xpand/Xtend et, plus récemment, Xtend2 (***Bettini, 2016***) offrent une spécification flexible et modulaire du code généré à travers la gestion d'imports et d'aspects. De plus, les règles de génération de chaque entité du modèle supportent le polymorphic dispatch[5]. Il s'agit d'une extension du patron de conception visiteur permettant à un objet de visiter la fonction adaptée à son type. Dans le cas du polymorphic dispatch, et à l'inverse du visiteur, aucun artefact intrusif n'est nécessaire dans le code du modèle pour obtenir ce comportement. Ce sont les méthodes visitées elles-mêmes qui définissent le type d'objet qu'elles supportent. Ceci est particulièrement utile dans un compilateur où une représentation intermédiaire est souvent décrite par un arbre de syntaxe abstraite dont les nœuds sont des spécialisations d'une unique définition abstraite.

3.3. 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 le processus de développement MDA, les techniques et langages de modélisation, méta-modélisation, transformation des modèles ainsi que les différentes phases pour la réalisation d'un DSL.

Dans le chapitre suivant, **nous présentons notre contribution** pour la conception de l'interface utilisateur des applications basées sur les composants web (eg. JSF, Asp.net, etc.) et mobile (Android, iOs, etc.). A cet effet, nous exploitons l'approche MDA pour proposer un modèle indépendant de la plateforme sous format textuel et des transformations M2M et M2T sont appliquées pour générer l'interface graphique pour une plateforme spécifique. Pour ce faire, nous utilisons Xtext pour définir le DSL et Xtend 2 pour réaliser les différentes transformations.

[5] http://dslmeinte.wordpress.com/2012/08/03/polymorphic-dispatch-in-xtend/

Chapitre 4

Modélisation et génération des interfaces graphiques des applications basées sur les composants selon une approche MDA

4.1. Introduction

A cause du grand nombre et de la variété des technologies mobiles (e.g. Android, iOS, Windows Phone, etc.) et web (e.g. Java Server Faces, Asp.net, etc.), le développement de la même application pour ces différentes plateformes, devient une tâche épuisante. L'approche MDA (*Model-Driven Architecture*) se propose d'apporter une solution pratique facile et efficace à une telle problématique, en développant une application multiplateforme.

Dans ce travail, nous **proposons une nouvelle approche** pour **la conception de l'interface utilisateur** pour les applications mobiles et web, ensuite, nous appliquons l'approche sur la plateforme Android et le Framework Java Server Faces. Cette approche sera **généralisée par la suite** pour toutes les plateformes mobiles et web basées sur les composants, en **définissant un langage** à part entière pour le développement des interfaces graphiques, un DSL (*Domain Specific Language*) de la neutralité technologique destiné à être compilé de manière croisée pour produire du code natif pour une variété de plateformes.

4.2. Interface graphique utilisateur et interaction utilisateur

La création de l'interface graphique utilisateur est une activité qui demande du temps et de la patience. Il est déterminant de bien y réfléchir pour avoir l'impact désiré auprès des utilisateurs. Pour une même application, plusieurs choix différents pourraient amener à de surprenantes réalisations.

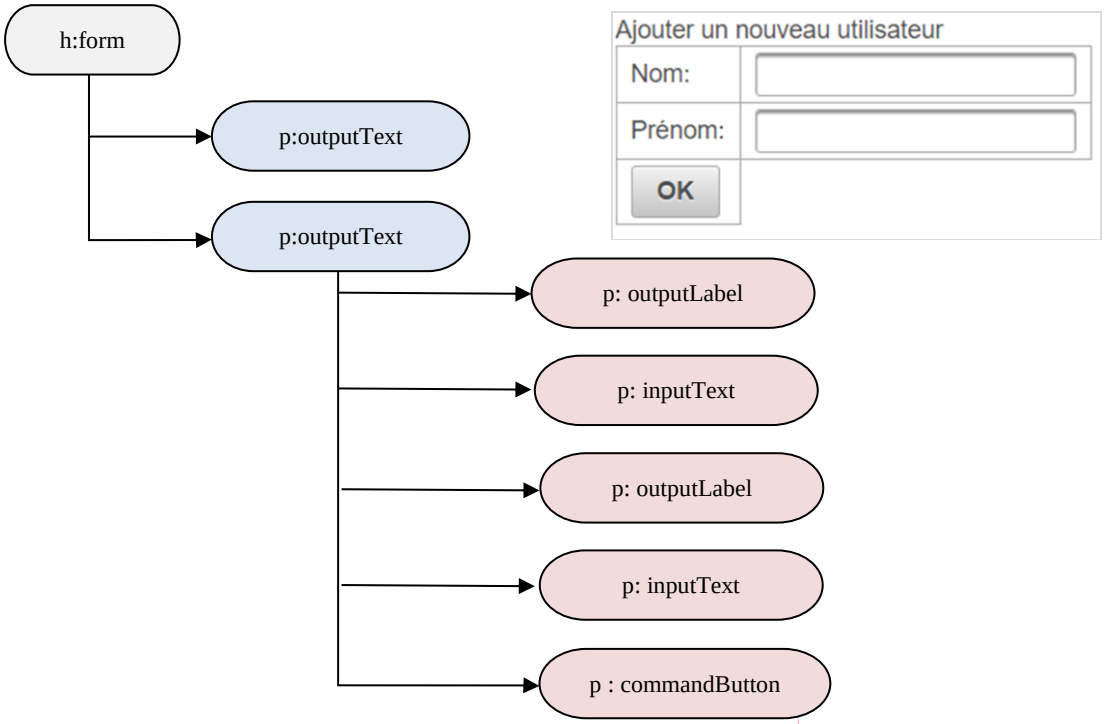
Une application offre différents moyens pour permettre à l'utilisateur d'interagir avec le système informatique. Ces moyens doivent être clairs pour permettre aux utilisateurs de comprendre rapidement comment exécuter les tâches et ils doivent être simples d'utilisation pour que la tâche s'effectue rapidement et sans erreur. Dans les sous-sections ci-dessous, nous présentons quelques caractéristiques des interfaces graphiques des plateformes web et mobile.

4.2.1. IGU et interaction utilisateur dans les applications web

La tendance durant la dernière décennie est de construire des solutions web basées sur un modèle de développement orienté événementiel, l'image des applications standalones ou client/serveurs lourds avec des outils tels que Delphi de Borland, Visual Basic de Microsoft ou Swing avec Java. Ces solutions, se composent d'un ensemble d'API servant, notamment, à représenter les composants, à gérer les états et les événements et à proposer des dispositifs de validation. D'où la naissance de Java Server Faces (***Bergsten, 2004***) et Asp.net (***Rongge, 2002***), qui sont des technologies dont le but est de proposer des Framework qui facilitent et

standardisent le développement des applications web avec Java et .Net. Ces outils permettent un développement de type RAD pour les applications web et ainsi faciliter le développement des applications de ce type. L'adoption du RAD pour le développement web trouve notamment sa justification dans le coût élevé de développement de l'IHM à la main et souvent par copier/coller d'un mélange de plusieurs technologies (HTML, JavaScript, ...), rendant fastidieux et peu fiable le développement de ces applications.

De ce fait, une vue d'une application web basée sur les composants est structurée sous la forme d'une hiérarchie de composants (e.g. JSF, Asp.net, WebObjects, etc.). La figure ci-dessous illustre un exemple d'une hiérarchie de composants dans une application JSF.




```
<h:form>
    <h:outputText value="Ajouter un nouveau utilisateur"/>
    <p:panelGrid columns="2">
        <p:outputLabel value="Nom:"  />
        <p:inputText id="nom" value="#{personneBean.personne.nom}" />
        <p:outputLabel value="Prénom:" />
        <p:inputText id="prenom" value="#{personneBean.personne.prenom}" />
        <p:commandButton value="OK" action="#{personneBean.onCreate()}"/>
    </p:panelGrid>
</h:form>
```

Figure 4. 1 - Hiérarchie de composants dans une application web (JSF)

En outre, les applications basées sur les composants peuvent être créées à l'aide des éditeurs de texte simples, ou à l'aide des IDE (*Integrated Development Environment*) comme Visual Studio .NET de Microsoft ou Java Studio Creator de Sun. Ces IDE permettent de faire glisser et déposer les composants d'une palette sur la page et de personnaliser leurs comportements et leurs apparences à travers les éditeurs de propriété. La figure ci-dessous

illustre les palettes de composants offertes par Visual Studio.NET pour Asp.net et Java Studio Creator pour JSF.

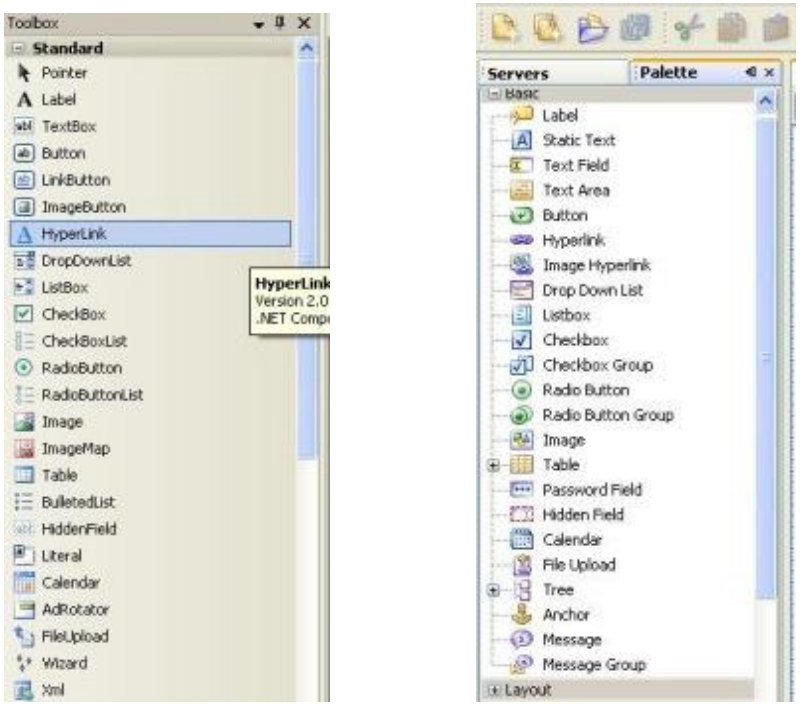


Figure 4. 2 - Palettes de composants dans Visual Studio et Java Studio Creator

4.2.2. IGU et interaction utilisateur dans les applications mobiles

Habituellement, une application mobile est constituée d'un certain nombre d'écrans, composés par un ensemble de widgets, avec lesquels les utilisateurs peuvent interagir afin d'accomplir plusieurs tâches.

Les écrans sont la façon dont l'application présente ses propres fonctionnalités. L'un d'entre eux est spécifié comme l'écran principal, qui est présenté à l'utilisateur quand il lance l'application pour la première fois. Chaque écran peut ensuite démarrer un autre écran pour effectuer différentes actions.

Les widgets sont les éléments de base d'une interface utilisateur graphique, comme les contrôles utilisateur dans l'interface graphique traditionnelle de l'ordinateur de bureau. Ils sont très standardisés, et un utilisateur expérimenté sur une plateforme particulière peut facilement basculer sur une plateforme concurrente. Le look and feel peut être différent et personnalisé, mais les widgets sont les mêmes ainsi que la façon d'interagir avec eux.

La forme la plus simple d'interaction est le toucher de l'écran avec un doigt. Le geste pour interface tactile multipoint (*Multi-touchgesture en anglais*) est un très bon exemple d'interaction avancée, et se réfère à une fonctionnalité utilisée par presque tous les appareils à écran tactile pour effectuer diverses actions:

- Un balayage d'un doigt est utilisé pour déplacer un objet entre deux points ;
- Une pincée se réfère à pincer le pouce et le doigt ensemble, et elle est utilisée pour zoomer sur une image ;
- Et plein d'autres.

Un utilisateur qui essaie n'importe quel smartphone s'attend à interagir de cette façon, car il est très intuitif et couramment utilisé.

La figure ci-dessous illustre les palettes des composants offertes par Microsoft Visual studio Express pour le développement des applications mobile pour Windows, Android Studio pour le développement des applications mobiles pour Android et XCode pour le développement des applications pour iOS.

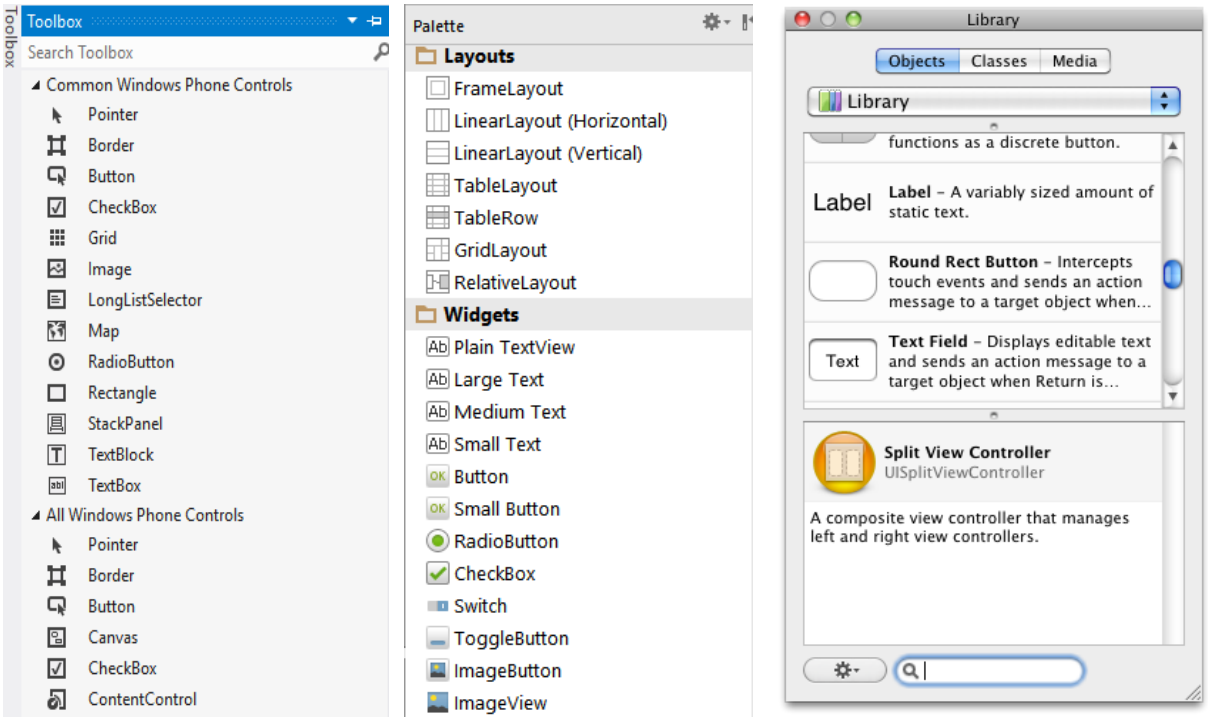


Figure 4. 3 - Palettes de composants dans Visual Studio Express, Android Studio et XCode

4.3. Génération des interfaces graphiques Android

Dans un premier temps, nous avons conçu un DSL pour la modélisation des interfaces graphiques d'une application Android. Ce DSL va nous permettre de créer des modèles textuels représentant une application sans tenir compte des préoccupations techniques. Ensuite, des transformations M2T sont appliquées afin de générer le code natif pour cette plateforme. Aussi, nous avons projeté vers une génération de code natif pour les différentes plateformes mobiles à partir d'un modèle Android, en appliquant des transformations M2M depuis le modèle source ***GUI Android*** vers le modèle de la plateforme cible, ensuite des transformations M2T pour gérer le code.

La figure ci-dessous illustre l'approche proposée.

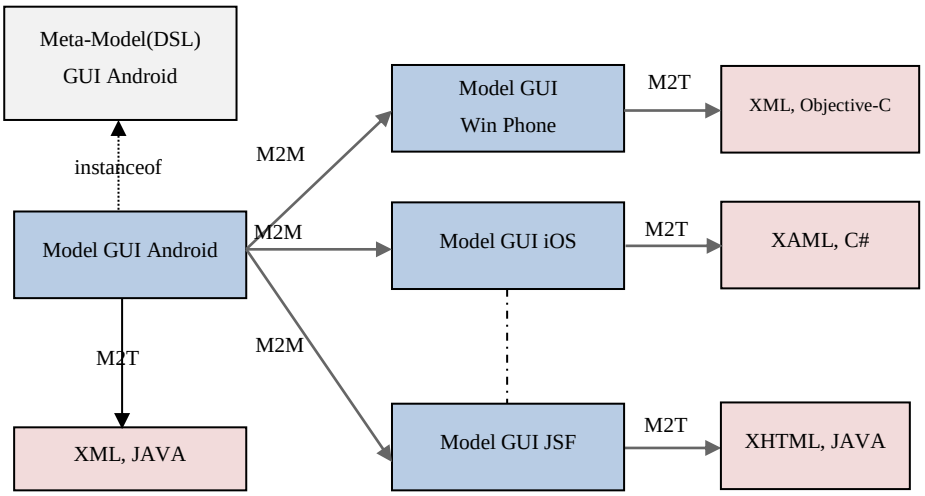


Figure 4. 4 - Architecture proposée pour la génération des interfaces graphiques à partir du modèle GUI Android (*Lachgar et al, 2014a*)

4.3.1. DSL GUI Android

Une interface graphique utilisateur est une collection de composants graphiques, qui peuvent être des boutons, du texte, mais aussi des regroupements d'autres composants graphiques, pour lesquels nous pouvons définir des attributs communs (taille, positionnement, etc.). Un extrait de notre DSL est représenté ci-dessous.

```
Screen:
        'Screen'
        'title' titre=STRING
        OPEN
                views+=View*
        CLOSE;
View:
        Widget | Layout;
Widget:
        Button | EditText | TextView ;
Layout:
        RelativeLayout | LinearLayout;
Button:
        'Button'
        attribut=Attribut;
```

Figure 4. 5 - Extrait de DSL GUI Android (*Lachgar et al, 2014a*)

Le composant graphique élémentaire de la plateforme Android est la vue: tous les éléments graphiques (boutons, images, cases à cocher, etc.) dans Android héritent de la classe View. Android offre la possibilité de regrouper plusieurs vues dans une arborescence à l'aide de la classe ViewGroup (*voir un exemple dans la figure ci-dessous*).

```
LinearLayout:
       'LinearLayout'
       attribut=Attribut
       (OPEN views+=View (NEXT views+=View)* CLOSE)?;
```

Figure 4. 6 - Extrait de DSL GUI Android (Layout) (*Lachgar et al, 2014a*)

Un layout ou modèle dans la documentation officielle d'Android est une extension de la classe ViewGroup. Il s'agit en fait d'un conteneur qui aide à positionner les objets, qu'ils s'agissent bien de widgets ou d'autres modèles au sein d'une interface.

4.3.2. Validation et transformation de modèle

La validation de modèle se fera en arrière-plan tandis que l'utilisateur de DSL est en train de taper dans l'éditeur, de sorte qu'une rétroaction immédiate est fournie. La plupart des contrôles pour le DSL seront mis en œuvre, selon la sémantique que nous souhaitons que notre DSL respecte. Ces contrôles supplémentaires sont implémentés à l'aide de Xtend.

Le générateur de stub est automatiquement créé par Xtext. Xtend offre de nombreuses fonctionnalités permettant d'écrire un code plus propre et plus lisible. Comme il est entièrement intégré à Java, toutes les bibliothèques Java sont accessibles depuis Xtend. De plus, Xtend IDE (*Integrated Development Environment*) lui-même est essentiellement le même que celui de JDT (*Java Developemnt Tools*). Pour toutes les raisons mentionnées ci-dessus, Xtext encourage l'utilisation de Xtend pour développer tous les aspects de la mise en œuvre d'un DSL.

Xtend fournit des expressions de modèle multi-lignes (en effet, toutes les chaînes sont multi-lignes dans Xtend). Un extrait de code utilisé pour la génération de code est présenté dans la figure ci-dessous:

```
List <Screen>screens = new ArrayList<Screen>
List <View>views = new ArrayList<View>
vartitre = null
varintid = 0
overridevoid doGenerate(Resource resource, IFileSystemAccess fsa) {
        var application = resource.contents.headas Application
        screens = application.screens
        titre = application.screens.get(0).titre
        for(Screen screen : screens){
                id = screens.indexOf(screen) + 1
                views = screen.views
                for (View v : views) {
                        fsa.generateFile(application.root+"res/"+screen.titre+".xml", v.compile)
                }
        }
        fsa.generateFile(application.root+'values/string' + ".xml", gen)
}
```

Figure 4. 7 - Extrait de code utilisé pour la génération des UGI (*Lachgar et al, 2014a*)

Un extrait du code généré pour l'interface graphique Android à partir d'un modèle basé sur notre DSL est illustré à la figure ci-dessous (*voir figure 4.8 pour plus de détails*):

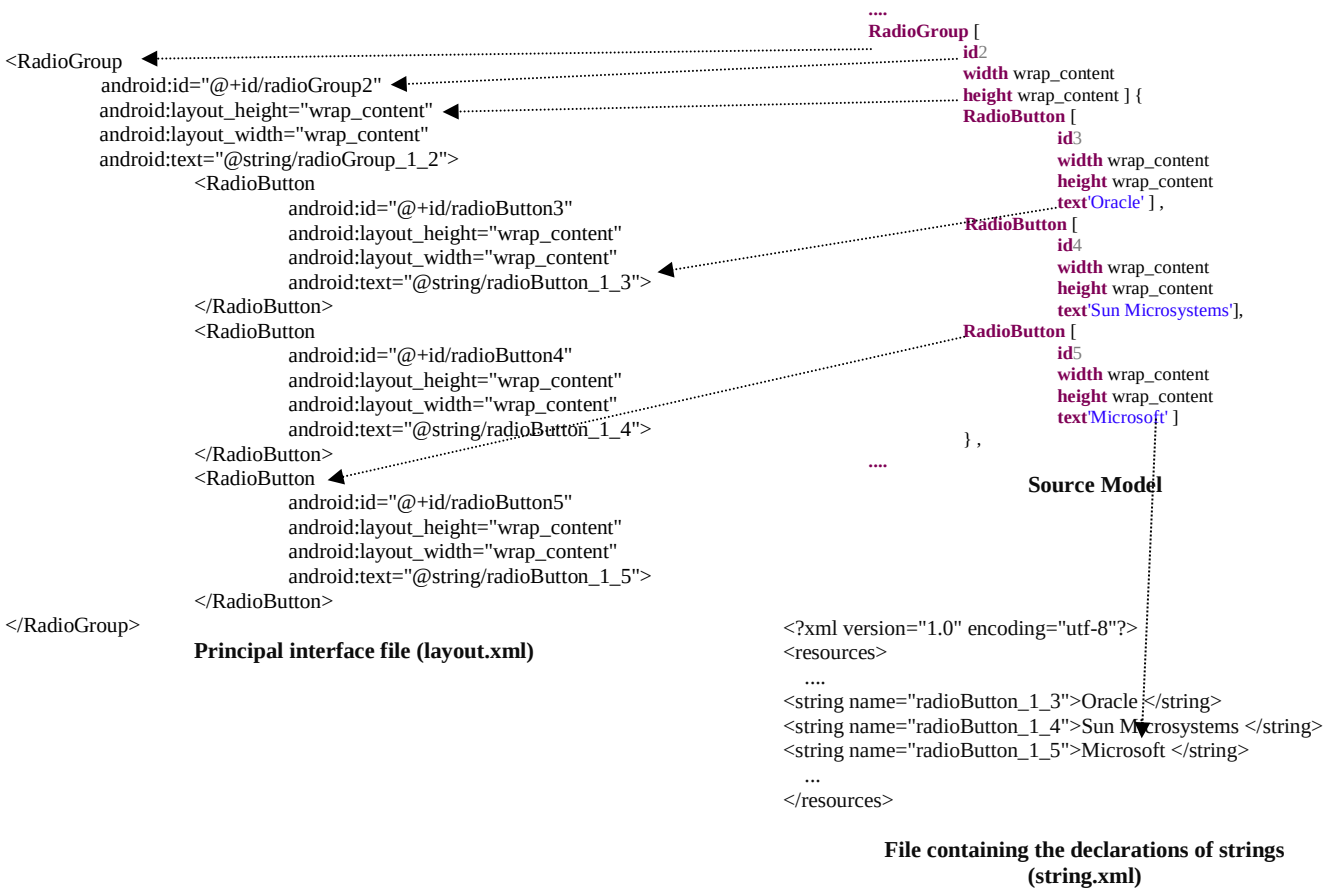


Figure 4. 8 - Code généré pour l'interface graphique Android (*Lachgar et al, 2014a*)

4.3.3. Synthèse

Le diagramme de transformation peut être résumé comme suit:

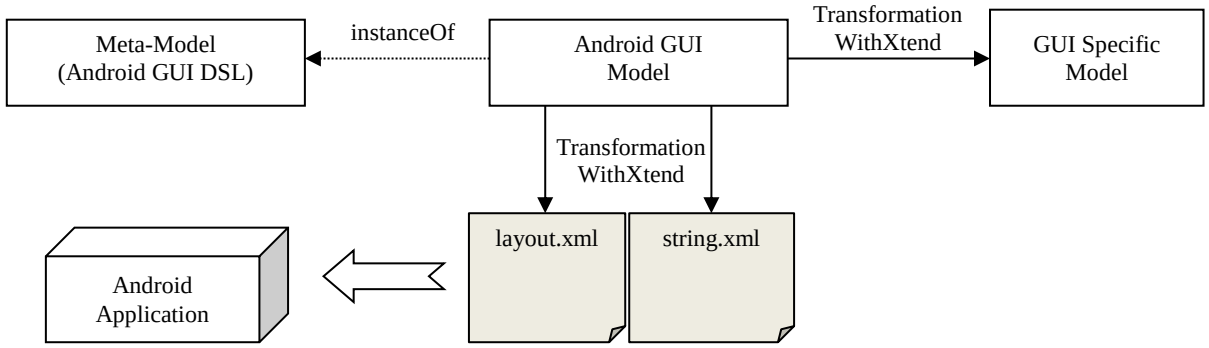


Figure 4. 9 - Diagramme de transformation (*Lachgar et al, 2014a*)

Le diagramme ci-dessous illustre la structure de projet réalisé avec les différentes briques logiciels :

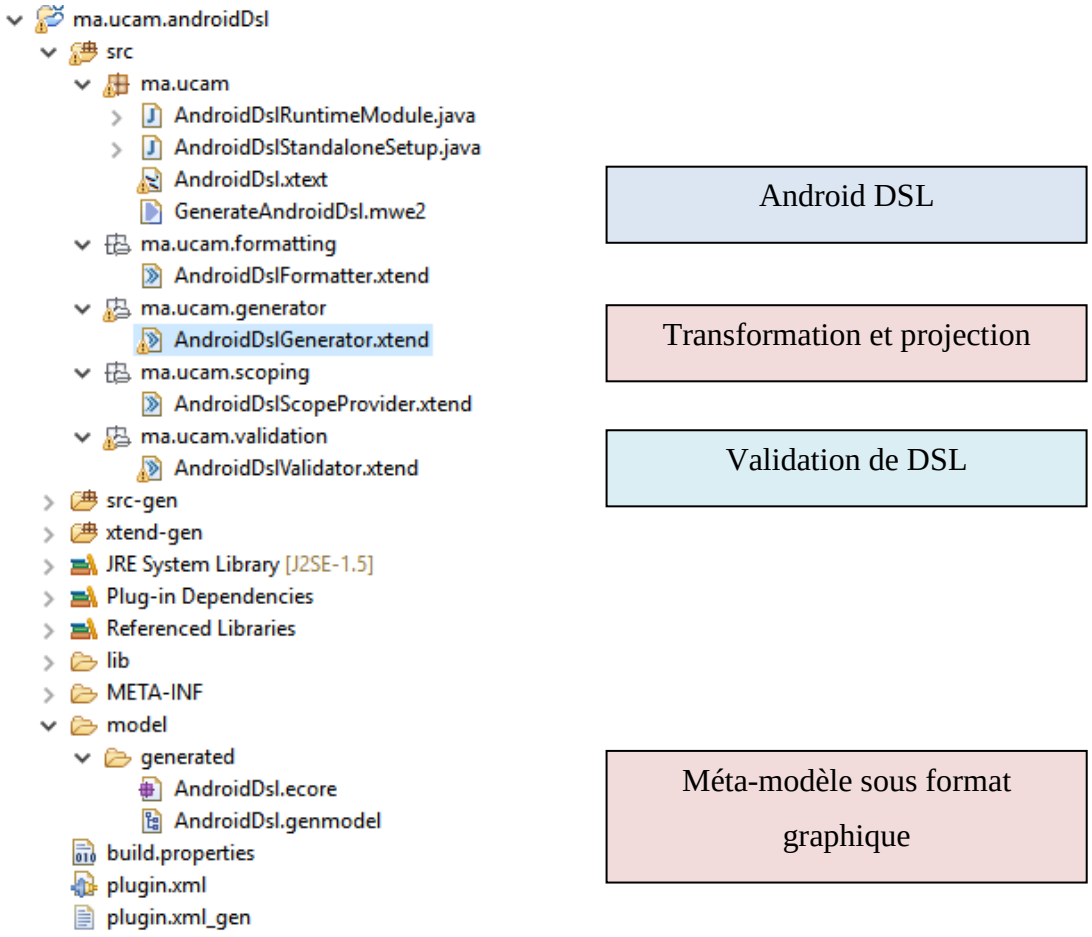


Figure 4. 10 - Structure de projet sous Eclipse Xtext

4.3.4. Exemple illustratif : E-Exam Application

Dans cette section, nous présentons un exemple illustratif de notre approche, afin de générer automatiquement l'interface graphique pour une application Android.

Nous considérons une application de test à choix multiples nommé E-Exam. Dans ce test, l'utilisateur doit répondre à des questions qui sont présentées dans différentes écrans. L'application calcule automatiquement le score et fournit le résultat immédiatement à l'utilisateur.

4.3.4.1. Analyse et création de modèle

L'objectif de cette étape est de concevoir les différentes interfaces graphiques de l'application en utilisant un modèle correspondant à notre DSL (*voir la figure 4.11 ci-dessous pour plus de détails*).

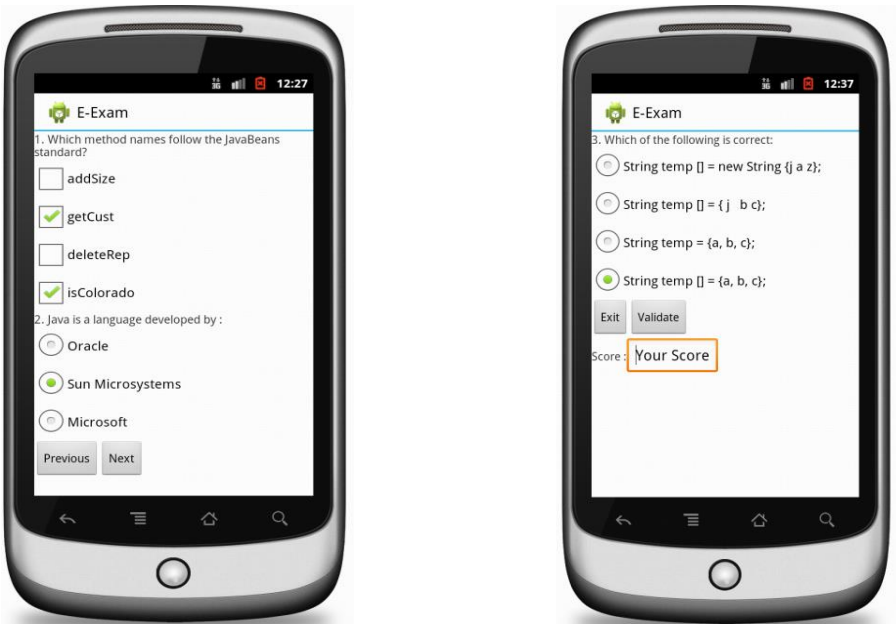
Dans ce cas, notre application est constituée de deux écrans nommés respectivement "Screen-1" et "Screen-2" qui contiennent une collection de composants.

```
Application root "F:\\demo\\" title "E-Exam",

Screen title "Screen-1" {
    RelativeLayout [ id 1 width wrap_content height wrap_content orientation 'vertical' ] {
          LinearLayout [ id 2 width wrap_content height wrap_content  orientation 'vertical'] {
               TextView [ id 1 width wrap_content height wrap_content text '1. Which method names follow the JavaBeans standard? ' ] ,
               CheckBox [ id 1 width wrap_content height wrap_content text 'addSize' ] ,
               CheckBox [ id 2 width wrap_content height wrap_content text 'getCust' ] ,
               CheckBox [ id 3 width wrap_content height wrap_content text 'deleteRep' ] ,
               CheckBox [ id 4 width wrap_content height wrap_content text 'isColorado' ] ,
               TextView [ id 2 width wrap_content height wrap_content text '2. Java is a language developed by : ' ] ,
               RadioGroup [ id 1 width wrap_content height wrap_content ] {
                   RadioButton [ id 1 width wrap_content height wrap_content text 'Oracle' ] ,
                   RadioButton [ id 2 width wrap_content height wrap_content text 'Sun Microsystems' ],
                   RadioButton [ id 3 width wrap_content height wrap_content text 'Microsoft' ]
               } ,
               LinearLayout [ id 3 width wrap_content height wrap_content orientation 'horizontal' ] {
                   Button [ id 1 width wrap_content height wrap_content text 'Previous' ] ,
                   Button [ id 2 width wrap_content height wrap_content text 'Next' ]
               }
          }
     }
}

Screen title "Screen-2"{
    RelativeLayout [ id 1 width wrap_content height wrap_content orientation 'vertical' ] {
         LinearLayout [ id 2 width wrap_content height wrap_content  orientation 'vertical'] {
            TextView [ id 1 width wrap_content height wrap_content text '3. Which of the following is correct:  ' ] ,
            RadioGroup [ id 1 width wrap_content height wrap_content ] {
               RadioButton [ id 1 width wrap_content height wrap_content text 'String temp [] = new String {"j" "a" "z"};' ] ,
               RadioButton [ id 2 width wrap_content height wrap_content text 'String temp [] = { "j " " b" "c"};' ],
               RadioButton [ id 3 width wrap_content height wrap_content text 'String temp = {"a", "b", "c"};' ],
               RadioButton [ id 4 width wrap_content height wrap_content text 'String temp [] = {"a", "b", "c"};' ]
            } ,
            LinearLayout [ id 3 width wrap_content height wrap_content orientation 'horizontal' ] {
               Button [ id 1 width wrap_content height wrap_content text 'Exit' ] ,
               Button [ id 2 width wrap_content height wrap_content text 'Validate' ]
            },
            LinearLayout [ id 4 width wrap_content height wrap_content orientation 'horizontal' ] {
                    TextView [ id 2 width wrap_content height wrap_content text 'Score ' ] ,
                    EditText [ id 1 width wrap_content height wrap_content text 'Your Score' ]
            }
     }
  }
}
```

Figure 4. 11 - Modèle de l'application E-exam (*Lachgar et al, 2014a*)

4.3.4.2. GUI Généré ciblant la plateforme Android

Nous pouvons automatiquement générer le code source dans un projet Android à partir de modèle de l'application E-exam. La figure ci-dessous illustre le résultat obtenu.



Screen_1 Screen_2

Figure 4. 12 - Application E-exam (*Lachgar et al, 2014a*)

Par conséquence, cette approche, nous permettra de développer une application de qualité en temps-efficace sans détails techniques inutiles.

Dans la section suivante, nous allons généraliser notre approche pour prendre en considération les plateformes mobiles et web basées sur les composants graphiques.

4.4. Conception de l'interface graphique utilisateur pour une application basée sur les composants

Pour couvrir l'aspect structurel, **nous proposons un DSL** pour décrire et concevoir les interfaces graphiques des applications basées sur les composants. **Notre approche propose un méta-modèle pour concevoir un modèle indépendant de la plateforme de l'interface graphique d'une application**. Ensuite, des transformations M2M (Model To Model) et M2T (Model to Text) sont appliquées pour générer le code de l'interface graphique ciblant une plateforme spécifique. La figure ci-dessous présente les différentes étapes qui caractérisent notre approche.

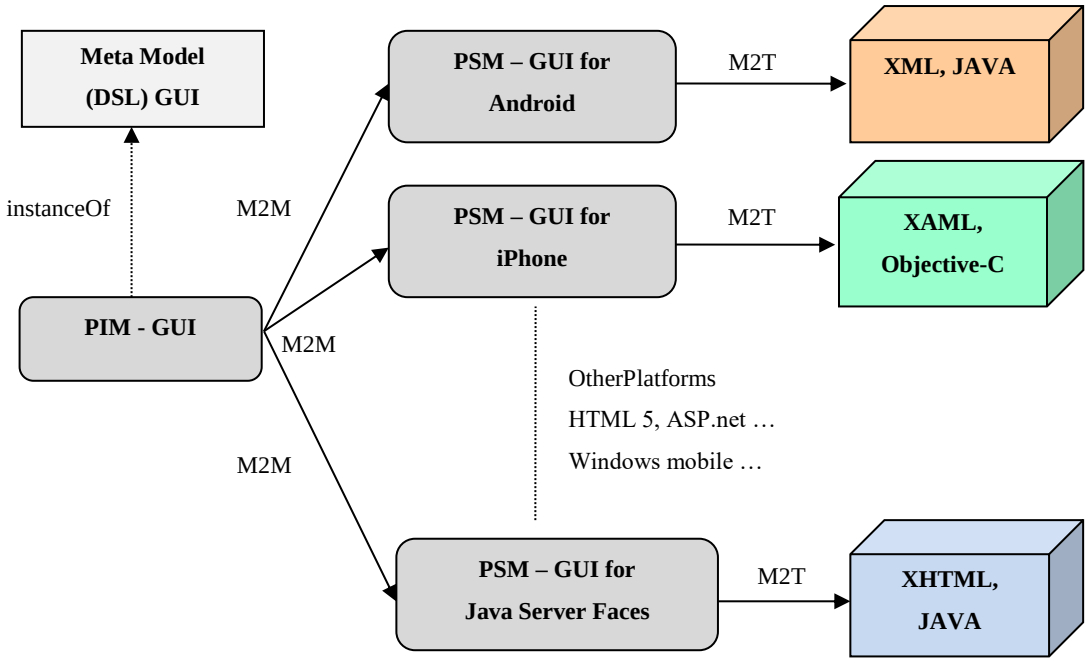


Figure 4. 13 - Architecture proposée pour la génération des interfaces graphiques à partir d'un modèle indépendant de la plateforme (PIM-GUI) (*Lachgar et al, 2015*)

4.4.1. Méta-modèle GUI DSL

Le méta-modèle proposé est décrit par le diagramme de classes suivant :

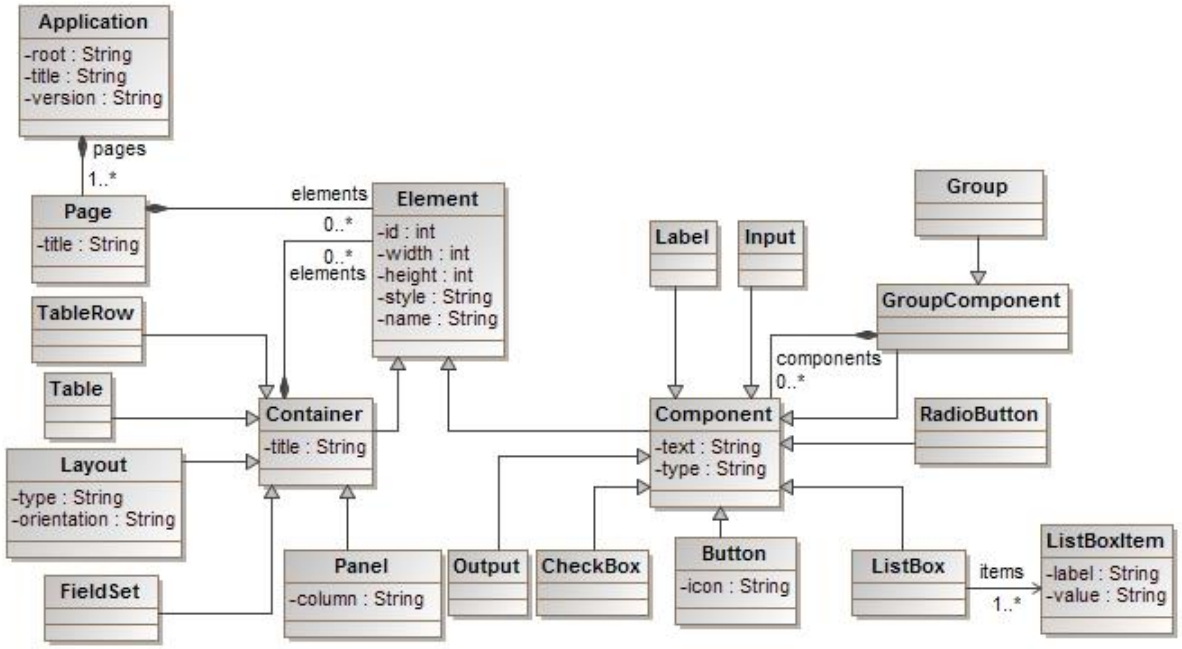


Figure 4. 14 - Le méta-modèle proposé GUI DSL (*Lachgar et al, 2015*)

4.4.2. GUI DSL

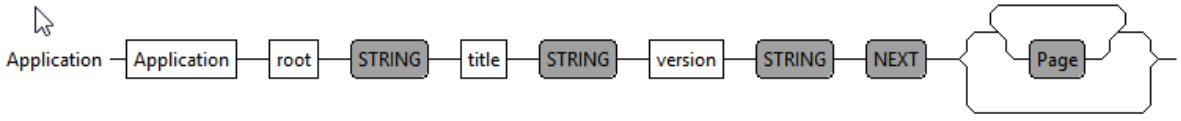
Garder le code "propre" est important pour le développement de logiciels (et cela inclut la modularité, la lisibilité et la maintenabilité). Xtext (***Bettini, 2016***) offre de nombreuses fonctionnalités pour garder l'application DSL propre et modulaire, grâce à sa décomposition en de nombreux aspects variés. Xtext est la solution envisagée pour la création de notre DSL.

4.4.2.1. Description de GUI DSL

Une application se compose de plusieurs pages et chaque page contient un ensemble de composants graphiques, qui peuvent être des boutons, du texte, mais aussi des groupements d'autres composants graphiques, pour lesquels nous pouvons définir des attributs communs (taille, positionnement, etc.). L'attribut "type" permet de distinguer le type d'éléments (e.g. date, email, texte, image, etc.) lors de la transformation.

Un extrait de GUI DSL est représenté ci-dessous.

```
Application:
      'Application'
      'root' root=STRING
      'title' title=STRING
      'version' version=STRING
      NEXT
      pages+=Page*;
```



```
Page:
      'Page'
      'title' title=STRING
      OPEN
            (elements+=Element*)
      CLOSE;

Element:
      Container | Component;
Container:
      FieldSet | Table | Panel;

Component:
      Input | Label | RadioButton | CheckBox | GroupComponent | ListBox;
FiledSet:
      'filedset'
      'legend' title=STRING
      OPENATT attribut=Attribut CLOSEATT
      OPEN
            (elements+=Element (NEXT elements+=Element)*)?
      CLOSE;
Panel:
      'panel'
      OPENATT attribut=Attribut
      ('header' header=STRING)?
      ('columns' column=STRING)? CLOSEATT
      OPEN
            (elements+=Element (NEXT elements+=Element)*)?
      CLOSE;
Input:
      'input'
      OPENATT attribut=Attribut CLOSEATT;
Label:
      'label'
      OPENATT attribut=Attribut CLOSEATT;
RadioButton:
      'radio'
      OPENATT attribut=Attribut CLOSEATT;
CheckBox:
      'checkBox'
      OPENATT attribut=Attribut CLOSEATT;
Button:
      'button'
      OPENATT attribut=Attribut CLOSEATT;
Groupe:
      'groupe'
      OPENATT attribut=Attribut CLOSEATT
      OPEN
             (components+=Component (NEXT components+=Component)*)?
      CLOSE;
ListBoxItem:
      'item'
      OPENATT
      'label' label=STRING
      'value' value=STRING
      CLOSEATT;
ListBox:
      'list'
      OPENATT attribut=Attribut CLOSEATT
      OPEN
            (items+=ListBoxItem (NEXT items+=ListBoxItem)*)?
      CLOSE;
```

Figure 4. 15 - Extrait de méta-modèle textuel GUI DSL (*Lachgar et al, 2015*)

4.4.2.2. Validation de modèle

La plupart des contrôles pour le DSL devront être mis en œuvre, conformément à la sémantique que nous souhaitons que notre DSL respecte. Ces contrôles supplémentaires peuvent être mis en œuvre à l'aide de Xtend 2 (***Bettini, 2016***).

La figure suivante illustre un exemple d'un validateur permettant d'assurer qu'il n'y a pas une duplication des identifiants des composants de la même classe.

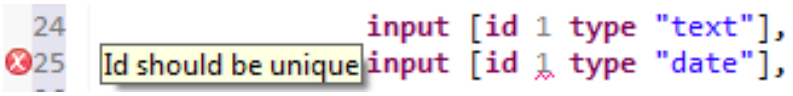


Figure 4. 16 - Exemple d'un validateur

Un extrait de la méthode d'implémentation de ce validateur est présenté ci-dessous:

```
Public static val DUPLICAT_ID = 'duplicat id'

@Check
def checkDuplicatIdForSameView(Page page) {
        var map = new HashMap<String, Element>()
        for (Element e : page.elements) {
                check(e, map)
        }
}

def check(Element e, Map map) {
        if (e instanceof Component) {
                if (map.containsKey(e.attribut.id + "" + e.class.name)) {
                        error('Id should be unique', e.attribut,
                        GUIPackage.Literals.ATTRIBUT__ID, DUPLICAT_ID)
                } else {
                        map.put(e.attribut.id + "" + e.class.name, e)
                }
                if (e instanceof GroupComponent)
                        for (Element ee : e.components)
                                check(ee, map)
        } elseif (e instanceof Container) {
                if (map.containsKey(e.attribut.id + "container")) {
                        error('Id should be unique', e.attribut,
                        GUIPackage.Literals.ATTRIBUT__ID, DUPLICAT_ID)
                } else {
                        map.put(e.attribut.id + "container", e)
                }
                for (Element w : e.elements)
                        check(w, map);
        }
}
```

Figure 4. 17 - Extrait de code pour implémenter un validateur de modèle PIM

4.4.2.3. Transformation

La transformation PIM vers PSM a été réalisée avec Xtend 2. Pour chaque élément de notre PIM est associé un ou plusieurs éléments de modèle PSM, souvent cette distinction est assurée par l'attribut "type". Cependant, nous avons utilisé les composants de la bibliothèque prime faces pour la mise en place de l'interface graphique JSF, cette bibliothèque offre des composants variés et conviviales. Certaines transformations sont représentées dans le tableau ci-dessous.

Tableau 4. 1 - Quelques règles de transformation (*Lachgar et al, 2015*)

Composant	PIM	PSM Android	PSM JSF
Ab	Label	TextView	outPutLabel
Plain Text Date Multiline Text E-mail	Input ▪ type = text ▪ type = date ▪ type = multiline ▪ type = email	EditText ▪ inputType = text ▪ inputType = date ▪ inputType = textMultiLine ▪ inputType = textEmailAddress	▪ inputText ▪ calendar ▪ inputTextarea ▪ inputText[a]
Ab	Output ▪ type = text	▪ TextView	▪ outputText
OK	Button ▪ type = image ▪ type = simple	▪ ImageButton ▪ Button	▪ commandButton[b] ▪ commandButton
	CheckBox	CheckBox	selectBooleanCheckbox
	RadioButton	RadioButton	selectItem
	Group ▪ type = radio	▪ RadioGroupe	▪ selectOneRadio
	ListBox	Spinner	selectOneMenu
	ListBoxItem	Item	selectItem
	TableRow	TableRow	Columns
	FieldSet	RelativeLayout & TextView	Fieldset
	▪ Panel ▪ Panel with columns attribute	▪ LinearLayout (orinetation = vertical) ▪ TableLayout	▪ Panel ▪ panelGrid

[a] inputTextavec validateur

[b] commandButton avec l'attribut icon

4.4.2.4. Projection dans les Template

Une fois que les transformations M2M sont réalisées. Une 2ème étape consiste à générer le code source à partir du PSM obtenu, ce que nous appelons la projection. Le Template est développé avec Xtend 2. Une partie du Template est représentée dans la figure ci-dessous.

```
override doGenerate(Resource input, IFileSystemAccess fsa) {
      var application = input.contents.headas Application
      pages = application.pages
      title = application.title
      for (Page page : pages) {
         id = pages.indexOf(page) + 1
         elements = page.elements
         titlePage = page.title
         for (Element e : elements) {
          fsa.generateFile (application.root + "android/" +  titlePage + ".xml", e.compile)
          fsa.generateFile (application.root + "jsf/" + titlePage +   ".xhtml", e.compileJsf)
         }
      }
      fsa.generateFile (application.root + 'values/string' + ".xml", gen)
}
def gen()'''
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">«titre»</string>
<string name="action_settings">Settings</string>
«FOR m : pages»
         «VARint id = pages.indexOf(m) + 1»
         «FOR v : m.elements»
               «genString(v, id)»
         «ENDFOR»
«ENDFOR»
</resources>
'''
```

Figure 4. 18 - Extrait de Template (*Lachgar et al, 2015*)

4.4.2.5. Synthèse

Pour résumer, notre approche repose sur quatre étapes principales:

- Analyse et modélisation de l'interface graphique avec un modèle textuel conforme à notre DSL présenté dans la figure 4.15;
- Validation du modèle par Xtend 2 ;
- Transformation de PIM vers le PSM à l'aide de Xtend 2 ;
- Génération de code source natif en projetant dans les Template des plateformes cibles.

Dans ce qui suit, nous présentons les diagrammes pour la génération des interfaces graphiques pour la plateforme Android et le Framework Java Server Faces (JSF).

a. La plateforme Android

Les différentes Templates utilisés sont les suivant :

- **string.xml** : Le fichier contenant la déclaration des constantes ;
- **AndroidManifest.xml** : Le fichier de configuration Android ;
- **layout.xml** : Le fichier qui présente l'interface graphique, ce fichier contient la déclaration des composants graphiques.

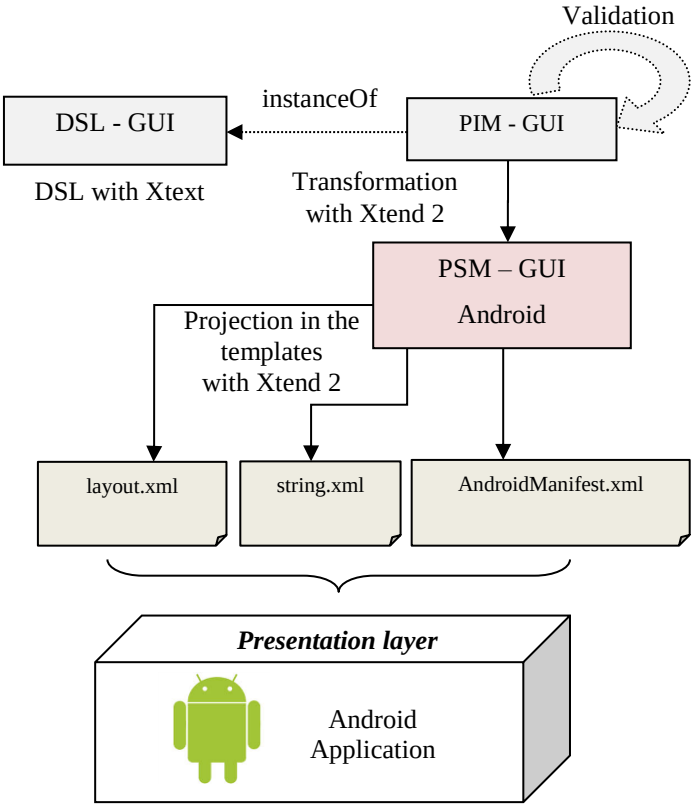


Figure 4. 19 - Diagramme de transformation et de génération pour Android-GUI (*Lachgar et al, 2015*)

b. Le Framework JSF

Les différents Templates utilisés sont les suivants :

- **web.xml** : Le fichier de configuration d'une application web entreprise (cas de JSF) ;
- **page.xhtml** : Le fichier qui présente l'interface graphique. Ce fichier contient la déclaration des composants graphiques ;
- **message.properties** : Le fichier contenant la déclaration des constantes.

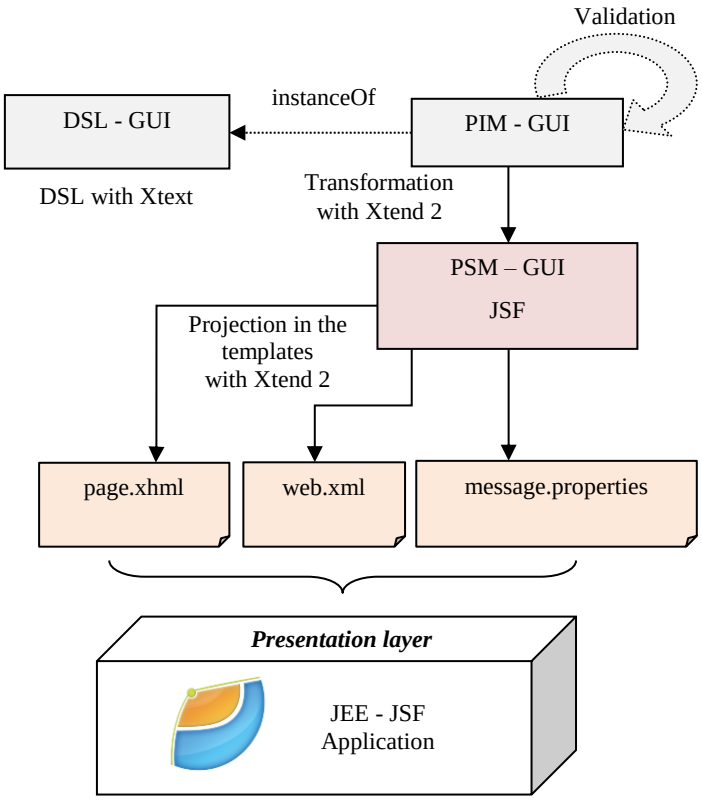


Figure 4. 20 - Diagramme de transformation et de génération pour JSF – GUI (*Lachgar et al, 2015*)

4.5. Exemple illustratif : E-CERTIFICATES

Dans cette section, nous présentons un exemple qui permet d'illustrer notre approche, qui génère automatiquement l'interface graphique pour une application Android et une application JSF. Nous considérons une application de test à choix multiples nommée E-certificates. Dans ce test l'utilisateur doit répondre aux questions qui sont structurées dans des fenêtres différentes (GUI). L'application évalue automatiquement le quiz, et donne le résultat immédiatement à l'utilisateur.

4.5.1. Analyse et création de modèle

L'objectif de cette étape est de concevoir les différentes interfaces graphiques pour notre application en utilisant un modèle conforme à notre ***GUI DSL*** (*voir la figure ci-dessous pour plus de détails*).

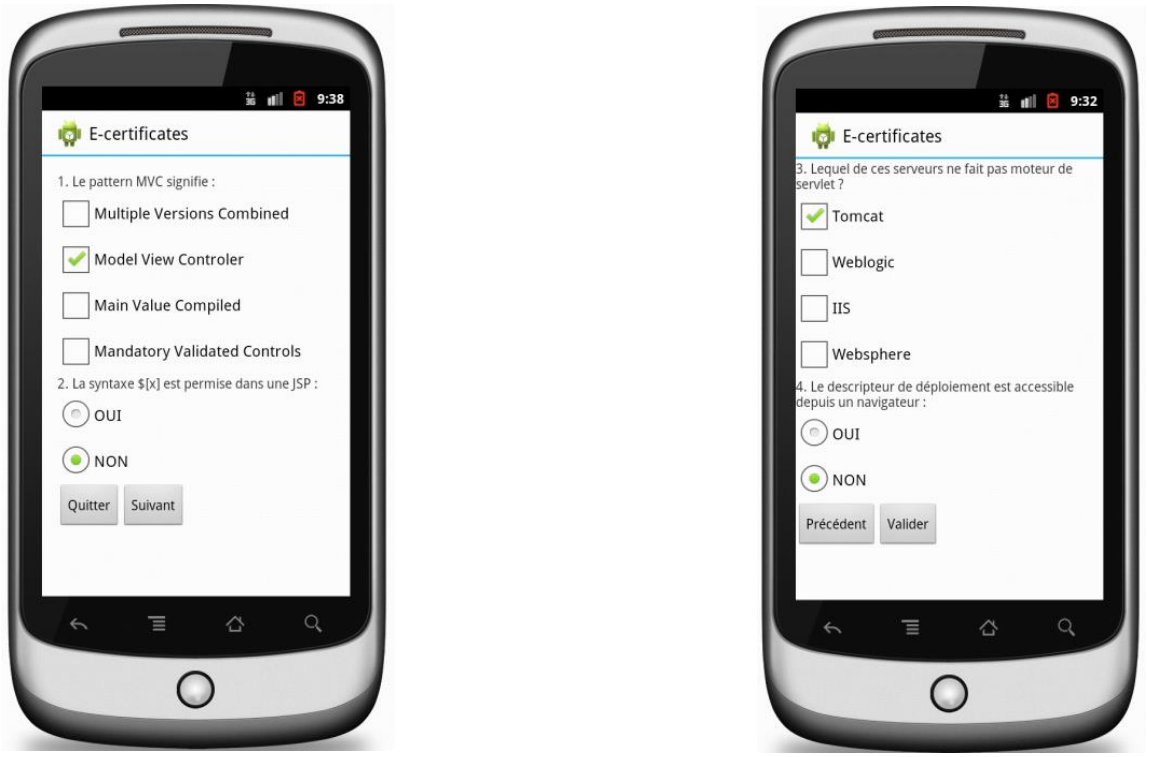
```
Application root "F:\\" title "E-Certificates" version "V3",

Page title "page-1" {
   panel [ id 1 type "simple" width 1 height 1 orientation 'horizontal' header "E-certificates" ] {
        panel [ id 2 type "grid" width 1 height 1   orientation 'vertical'  column "1"] {
            label [ id 1  text '1. Le pattern MVC signifie : ' ] ,
            groupe [ id 2 type "check"  orientation "vertical" ] {
               checkBox [ id 1  text 'Multiple Versions Combined' ] ,
               checkBox [ id 2  text 'Model View Controler' ],
               checkBox [ id 3  text 'Main Value Compiled' ],
               checkBox [ id 4  text 'Mandatory Validated Controls' ]
             } ,
            label [ id 3 text '2. La syntaxe $[x] est permise dans une JSP : '] ,
            groupe [ id 3 type "radio"  orientation "vertical" ] {
               radio [ id 1  text 'OUI' ] ,
               radio [ id 2  text 'NON' ]
             } ,
             panel [ id 3 type "simple" orientation 'horizontal' column "2" ] {
               button [ id 1  text 'Suivant' ] ,
               button [ id 2  text 'Précédent' ]
             }
        }
    }
}

Page title "page-2" {
   panel [ id 1 type "simple" width 1 height 1 orientation 'horizontal' header "E-certificates" ] {
        panel [ id 2 type "grid" width 1 height 1   orientation 'vertical'  column "1"] {
            label [ id 1  text '3. Lequel de ces serveurs ne fait pas moteur de servlet ? ' ] ,
            groupe [ id 2 type "check" orientation "vertical" ] {
               checkBox [ id 1  text 'Tomcat' ] ,
               checkBox [ id 2  text 'Weblogic' ],
               checkBox [ id 3  text 'IIS' ],
               checkBox [ id 4  text 'Websphere' ]
             } ,
            label [ id 3 text '4. Le descripteur de déploiement est accessible depuis un navigateur : '] ,
            groupe [ id 3 type "radio"  orientation "vertical" ] {
               radio [ id 1  text 'OUI' ] ,
               radio [ id 2  text 'NON' ]
             } ,
             panel [ id 3 type "simple" orientation 'horizontal' column "2" ] {
               button [ id 2  text 'Précédent' ] ,
               button [ id 3  text 'Valider' ]
             }
        }
    }
}
```

Figure 4. 21 - Modèle de l'application E-certificates (*Lachgar et al, 2014b*)

Notre application est constituée de deux pages nommées respectivement “Page-1” et “Page-2” et chaque page regroupe un ensemble de composants.

4.5.2. Code graphique généré ciblant la plateforme Android

Nous pouvons générer automatiquement les fichiers sources dans un projet Android à partir d’un modèle conforme à notre DSL. Le résultat obtenu est représenté dans la figure ci-dessous.

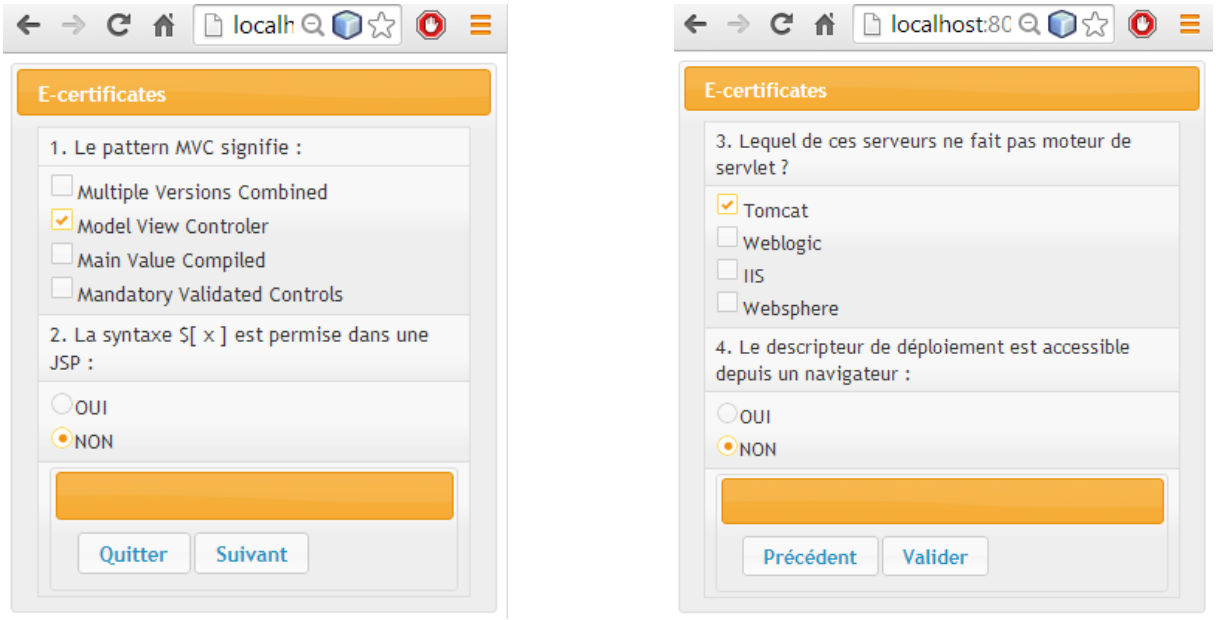


Page-1 Page-2

Figure 4. 22 - E-certificates avec Android (*Lachgar et al, 2014b*)

4.5.3. Code graphique généré ciblant le Framework JSF

Nous pouvons générer automatiquement les fichiers sources dans un projet Web Java à partir d'un modèle conforme à notre DSL. Le résultat obtenu est représenté dans la figure ci-dessous.



Page-1 Page-2

Figure 4. 23 - E-certificates avec JSF (*Lachgar et al, 2014b*)

Ainsi, cette démarche permettra de développer une application de qualité en un temps efficace sans préoccupations techniques.

4.6. 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.

Dans ce chapitre, nous avons présenté une étude des interfaces graphiques de la plateforme Android et le Framework Java Server Faces, et nous avons adopté l'approche MDA pour générer automatiquement l'interface graphique pour ces deux plateformes à partir d'un modèle conforme à notre ***GUI DSL***. Cette approche sera généralisée pour toutes les plateformes mobiles (e.g. iPhone, Windows mobile, etc.) et web (e.g. Asp.net, JSF, etc.) basées sur les composants.

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 GUI 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 Xtend 2 ;
- Transformation PIM vers le PSM à l'aide de Xtend 2 ;
- Génération de code en projetant dans les Templates des plateformes cibles.

Dans le chapitre suivant, nous présentons une approche pour la modélisation et la génération de code natif pour des applications mobiles multiplateformes.

Chapitre 5

Modélisation et génération du code natif des applications mobiles multiplateformes en utilisant un DSL

5.1. Introduction

Durant la dernière décennie, les technologies de développement mobile ont connu une croissance trop rapide et représentent une tendance émergente. En outre, différents smartphones utilisent des systèmes d'exploitation diversifiés, qui prennent en charge différents langages de programmation. Par conséquent, le développement d'applications natives individuellement pour chaque plateforme se révèle être un effort ardu et coûteux à entreprendre. Par conséquence, le concept «write once, deploy everywhere» réduira considérablement le coût de création, de maintenance et de contrôle des versions des applications mobiles.

Le travail présenté dans ce chapitre **est axé sur l'étude d'une approche dirigée par les modèles pour la conception des applications mobiles multiplateformes**, et la définition d'un méta-modèle textuel pour la création d'un nouveau modèle indépendant de la plateforme cible, selon l'architecture pilotée par modèle. De plus, Il donne également une description détaillée des transformations de modèle, pour obtenir le modèle spécifique à la plateforme (PSM) pour chaque plateforme mobile cible. Cependant, le code généré concerne non seulement la signature des méthodes et la structure des classes, mais aussi les blocs de code qui implémentent des fonctionnalités spécifiques telles que les transitions entre les écrans, l'accès aux capteurs embarqués, l'accès aux ressources (e.g. appareil photo, téléphone, sms, etc.), ces blocs peuvent être générés en paramétrant des modèles de code (*Template*). Pour ce faire, nous avons choisi un langage spécifique au domaine (DSL), afin d'accroître la productivité des ingénieurs logiciels en termes d'abstraction de code de bas niveau standard, d'apporter un gain de clarté imminent, ainsi qu'une simplicité d'utilisation. Ce travail présente encore quelques limites; Les applications générées ne respectent pas une architecture en couches et les fichiers de configuration de la base de données sont partiellement générés.

Pour la mise en place de notre DSL, nous avons opté pour le Framework Xtext (***Bettini, 2016 ; The Eclipse Foundation, 2013a***), ce dernier couvre tous les aspects d'un IDE moderne (e.g. l'analyseur, le compilateur, l'interpréteur et une intégration complète dans l'environnement de développement Eclipse). D'autre part, pour la réalisation des différentes transformations, nous avons utilisé le langage "Xtend 2" (***Bettini, 2016 ; The Eclipse Foundation, 2014***), qui étend de langage Java, et utilise : les clusters, les méthodes d'extension, le répartiteur multiple, la surcharge des opérateurs, un switch () plus puissant que celui du langage C, les chaînes multi-lignes et un système de Template permettant une indentation plus précise des Templates, ainsi que celui du texte généré.

Ce chapitre est structuré comme suit: La première section rappelle brièvement quelques préalables sur l'apport de l'approche MDA pour le développement des applications mobiles et l'architecture MDA avec le DSL. La deuxième section décrit la méthodologie proposée, aussi, présente le méta-modèle proposé, quelques règles de transformation et les différents templates utilisés dans la génération du code. La troisième section montre l'applicabilité de notre approche à travers une étude de cas. Finalement, nous présentons les limitations de notre approche basée sur le DSL.

5.2. L'apport de l'approche MDA pour le développement des applications mobiles multiplateformes

Le développement des applications pour les smartphones est une tâche compliquée, en raison de la variété des plateformes mobiles et la diversité des périphériques qui doivent être pris en charge. L'approche MDA a fait ses preuves pour le développement des applications d'entreprise (*AndroMDA, 2014; Mtsweni, 2012*), et peut également apporter beaucoup pour le développement des applications mobiles.

Nous verrons dans cette section pourquoi et comment l'approche MDA peut nous aider à assurer la continuité de l'expertise et à garantir une certaine amélioration de la productivité, tout en répondant aux problématiques de la fragmentation des plateformes mobiles (*Kleppe et al, 2003 ; Florent et al, 2013*).

La figure ci-dessous résume les différents points réponses à cette problématique :

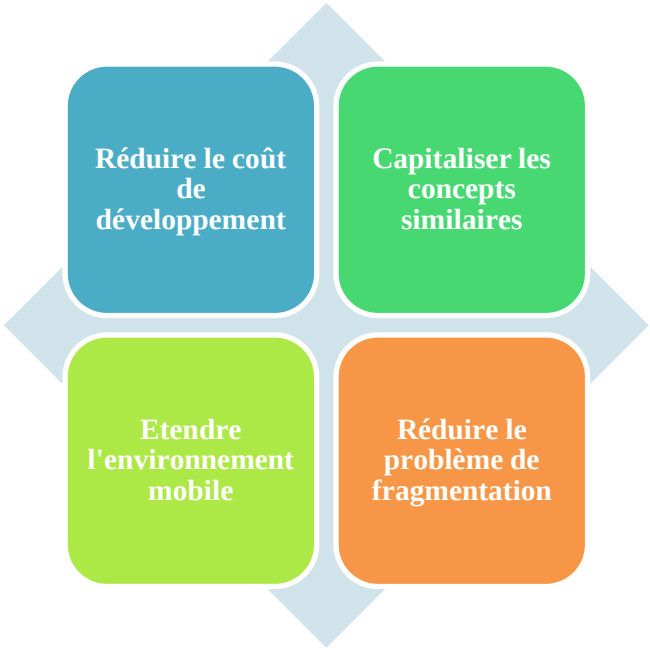


Figure 5. 1 - Apport de l'approche MDA pour le développement des applications Mobiles

(1) *Réduire le coût de développement:* Le grand défi des entreprises consiste à diminuer le coût de la mise en œuvre de leurs applications, suite aux budgets de plus en plus serrés. Ainsi, les générateurs de code et le DSL ont une grande valeur vis-à-vis de capitalisation et de la réutilisabilité. Par conséquent, ils réduisent le coût de la mise en œuvre, y compris les coûts initiaux d'investissement et de maintenance. En effet, on note :

- Une conception facilitée par le DSL adapté, cadrant les besoins spécifiques des plateformes mobiles ;
- Un générateur de code disponible et utilisable permet de stimuler rapidement le démarrage d'un projet ;
- Améliore la productivité de l'équipe de développement (e.g. les modèles basés sur le DSL peuvent être utilisés pour une génération de code automatique ou semi-automatique) (***Presso et al, 2005; Trask et al, 2006***) ;
- Le code ou les modèles complexes susceptibles d'erreurs sont générés automatiquement (e.g. La couche accès aux données, etc.) ;
- Un code généré de bonne qualité qui suit les meilleures pratiques, assure un meilleur contrôle et réduit les coûts d'entretien (***Trask et al, 2006; Weigert et al, 2006***) ;
- Les outils basés sur l'approche MDA réduisent les coûts du projet graduellement (***Kleppe et al, 2003***).

(2) *Réduire les problèmes de fragmentation:* La diversité des systèmes d'exploitation mobiles rend la mise en œuvre des applications multiplateformes plus onéreuse et fastidieuse, et par conséquent les entreprises font face à une alternative:

- Développer une application selon l'approche native: pour obtenir une application de qualité en temps efficace, mais, sa distribution est limitée à une plateforme spécifique (***El-Kassas et al, 2014***) ;
- Développer une application selon l'approche hybride ou web: pour obtenir une application avec des fonctionnalités limitées, cependant, elle est destinée par de nombreuses plateformes et aussi par un large panel d'utilisateurs (***El-Kassas et al, 2014***).

  Il existe de nombreuses solutions hybrides telles que PhoneGap (***Tian et al, 2013***), ou Adobe Flex, qui aident à développer des applications multiplateformes, mais leur appui est encore limité. En fait, une application native sera la plus appropriée pour

plusieurs raisons (***El-Kassas et al, 2015; Selvarajah et al, 2013 ; Lachgar et al, 2017***):

- Les applications natives ont les meilleures performances, et utilisent les dernières ressources matérielles disponibles pour améliorer les performances ;
- Les applications fonctionnent en mode hors-ligne ;
- Les applications sont distribuées à travers les boutiques en ligne, offrant une meilleure visibilité aux utilisateurs potentiels ;
- L'application aura accès aux dernières API natives publiées d'une plateforme ;
- Les applications sont développées avec des langages natifs, et ont accès aux IDE fournissant les meilleurs outils pour développer, déboguer le projet rapidement. Par exemple, une application Android peut être construite en Java sur Android Studio, et une application iOS peut être construite en Objective C ou Swift sur XCode, qui disposent de tous les outils pour déboguer, concevoir l'interface et vérifier la performance au moyen des instruments, et analogues etc. ;

L'approche MDA peut aborder ces problèmes et décrire les exigences fonctionnelles d'une application quelle que soit la plateforme d'exécution, et se conformer aux spécifications techniques de chaque plateforme. Ainsi, l'approche MDA permettra de génère le code sans valeur ajoutée spécifique à chaque plateforme. Cela garantit une productivité plus élevée, même sur les applications natives.

(3) *Capitaliser sur des concepts similaires:* Les concepts généraux sont standards et communs dans les applications mobiles, malgré que chaque système d'exploitation mobile utilise ses propres langages de programmation et ses propres fonctionnalités, ainsi que ses propres API. De ce fait, nous pouvons capitaliser sur les concepts communs des plateformes pour modéliser une application générique, y compris par exemple:

- Composants graphiques et évènements associées: un composant est un objet graphique par lequel l'utilisateur interagit avec l'application, ce qui provoque une action (ou évènement) liée au composant; Nous trouvons la majorité des composants sur tous les systèmes d'exploitations mobiles (e.g. bouton, champ de texte, étiquette, menu, bouton radio, case à cocher, liste, etc.) (***Tschernuth et al, 2011***) ;
- Activité: correspond à un écran de l'application, c'est un concept commun dans tous les systèmes d'exploitation mobile. Une application est composée d'une ou plusieurs activités et chaque activité contient un ensemble de composants graphiques (***Tschernuth et al, 2011***) ;

- Gestion des données d'application: la mise en place d'une application selon une architecture en couche (e.g. séparer le code de l'interface graphique et le code fonctionnel, etc.) ;
- Gestion des événements: une application doit prendre en charge certains événements qui peuvent apparaître pendant l'exécution (e.g. appels téléphoniques, des messages texte, etc.) (***Lettner et al, 2011***) ;
- Fonctionnalités natives: chaque système d'exploitation mobile offre un ensemble de fonctionnalités et d'applications natives qui sont communes à tous ces systèmes d'exploitation (e.g. calendrier, GPS, alarmes, microphones, capteurs embarqués, contacts, le calendrier, etc.).

  L'approche MDA décompose la conception d'une application en deux aspects (***OMG, 2003***):

- *Aspect fonctionnel*: il est indépendant des détails liés à la plateforme d'exécution. Dans cette section, nous modélisons des concepts abstraits tels que des écrans, des boutons, des objets métiers, les objets d'accès aux données (DAO) et nous restons indépendants de leur implémentation technique. Le modèle fonctionnel de l'application assure **la durabilité de l'expertise** ;
- *Aspect technique*: qui décrit l'architecture de la plateforme d'exécution. Dans cette section, nous spécifions comment le concept fonctionnel est généré, c'est-à-dire quel langage (e.g. C #, Swift, Java, etc.), quelle version (e.g. Android 5.0, etc.) et quelle implémentation de la base de données (e.g. SQLite, etc.). Ensuite, nous modélisons l'architecture technique en tenant compte de la plateforme d'exécution.

(4) *Étendre L'environnement mobile:* L'environnement mobile est réduit par rapport à d'autres, comme Java EE et .Net. En effet, l'utilisation de bibliothèques tierces (***Costanich, 2011; Sun et al, 2014***) dans les applications mobiles est limitée par différents systèmes d'exploitation pour plusieurs raisons:

- L'utilisation massive des bibliothèques n'est pas une bonne pratique pour le développement mobile, aussi, la grande taille des applications peut être un obstacle pour l'installer dans l'appareil mobile (problème d'espace mémoire) ;
- Le manque de visibilité des librairies de développement. En effet, les bibliothèques mobiles existantes sont presque exclusivement développées par des développeurs tiers. La surveillance et l'entretien ne sont pas toujours assurés ;

- Les éditeurs Google, Apple, Microsoft ont maintenu la main dans l'évolution des SDK et ne garantissent pas la compatibilité des bibliothèques tierces avec de nouvelles versions de plateformes.

  Pour assurer la durabilité du code et élargir son périmètre de compatibilité pour plusieurs versions, il est préférable d'être indépendant des bibliothèques tierces et d'avoir un code standard. L'approche MDA réduit le temps de développement en générant du code standard au SDK, selon les bonnes pratiques d'architecture et de qualité.

Pour résumer, au lieu d'investir dans la technologie qui héberge une application, il vaut mieux investir sur les techniques de mise en œuvre, ce qui permet de capitaliser les fonctionnalités d'une application, quelle que soit sa technologie. Grâce à une méthode qui intègre l'évolutivité (***Mohagheghi et al, 2008***) et la durabilité (***Palyart et al, 2011***) dans son processus de développement comme dans l'approche MDA, qui représente une proposition pour répondre à cette problématique.

5.3. DSL et l'approche MDA

Un langage spécifique au domaine (DSL) est un langage dédié à un domaine particulier, ainsi, permet d'avoir une autonomie dans la description du langage qui assure une parfaite convenance au domaine. Par opposition aux langages de programmation généraux tels que Java, C#, Swift, etc. et les langages de modélisation généraux comme UML. Le DSL apparaît comme un langage (***Kahlaoui et al, 2014 ; Laine, 2013***) :

- *Spécifique*: Un DSL est un langage, de transformation, de modélisation, de programmation ou d'interrogation, selon ce pourquoi il a été conçu. Il est spécifique au domaine, contrairement à un langage général tel que Java, qui peut traiter pratiquement n'importe quel problème ;
- *Doté d'un vocabulaire précis et concis*: un DSL est facile à utiliser sans ambiguïté. Son expressivité est basée sur un vocabulaire spécifique, propre au domaine métier ;
- *Personnalisé*: la syntaxe DSL est facile à personnaliser, elle est basée sur un vocabulaire et des règles syntaxiques entièrement définies par son concepteur ;
- *Formulé à un problème ou à un domaine particulier*: Avec sa syntaxe personnalisable exprimant des concepts communs à son domaine d'application, le DSL est configurable pour le métier, à une communauté ou à un projet. Facile à interpréter, c'est un outil de communication entre les experts du domaine et le développeur; Il permet aux experts de participer à la conception fonctionnelle de l'application.

Toutefois, les DSL sophistiqués peuvent nécessiter des fonctionnalités avancées de vérification de type. Ceci s'explique par le fait qu'elles incluent généralement des expressions, des types et la notion de conformité de type, avec le Framework Xtext, la construction de langages spécifiques au domaine (DSL), intégrée à l'Eclipse IDE, est devenue de plus en plus populaire et viable même pour des domaines non triviaux.

Nous suggérons dans le diagramme ci-dessous, l'architecture utilisée dans l'approche MDA basée sur le DSL (*voir la figure ci-dessous pour plus de détails*):

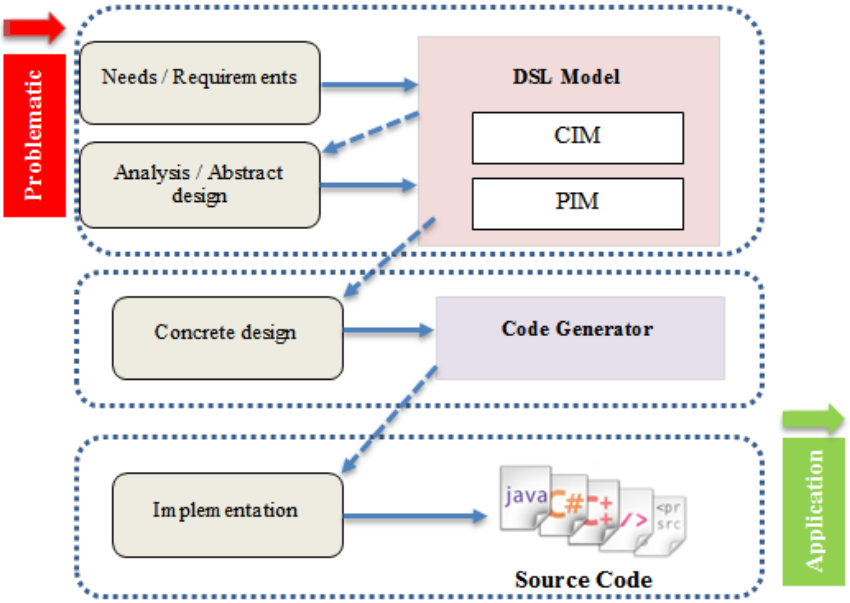


Figure 5. 2 - Architecture MDA basée sur le DSL

Dans cette définition de l'approche MDA, il s'agit toujours de séparer les préoccupations. En effet, les spécifications techniques sont définies par la DSL, le CIM et le PIM sont regroupés dans le modèle DSL. Le PSM est remplacé par le code lui-même, car avec le DSL, les templates conçus dans le générateur de code permettent d'exprimer l'architecture de la plateforme cible. Ainsi, les DSL rendant le processus plus flexible et plus rapide.

5.4. La méthodologie proposée

Dans cette section, **nous présenterons notre méthodologie** pour mettre en œuvre l'approche de conception pilotée par les modèles pour développer des applications mobiles multiplateformes. Ainsi, l'objectif est de concevoir l'aspect structurel des applications, comme les interfaces graphiques utilisateurs et les ressources auxquelles les applications doivent accéder.

La conception doit être effectuée indépendamment d'une plateforme logicielle particulière, en raisonnant avec les éléments et les caractéristiques des applications mobiles. Ensuite, la transformation du modèle abstrait vers les différents modèles spécifiques de chaque

plateforme cible doit être mise en œuvre par un outil automatisé. Enfin, cet outil doit être également capable de générer le code de l'application (e.g. classes, signature des méthodes et d'autres blocs de code), évitant ainsi aux développeurs d'écrire du code répétitif. La méthodologie proposée se réfère au développement des applications mobiles les plus usuelles (*voir la figure ci-dessous pour plus de détails*).

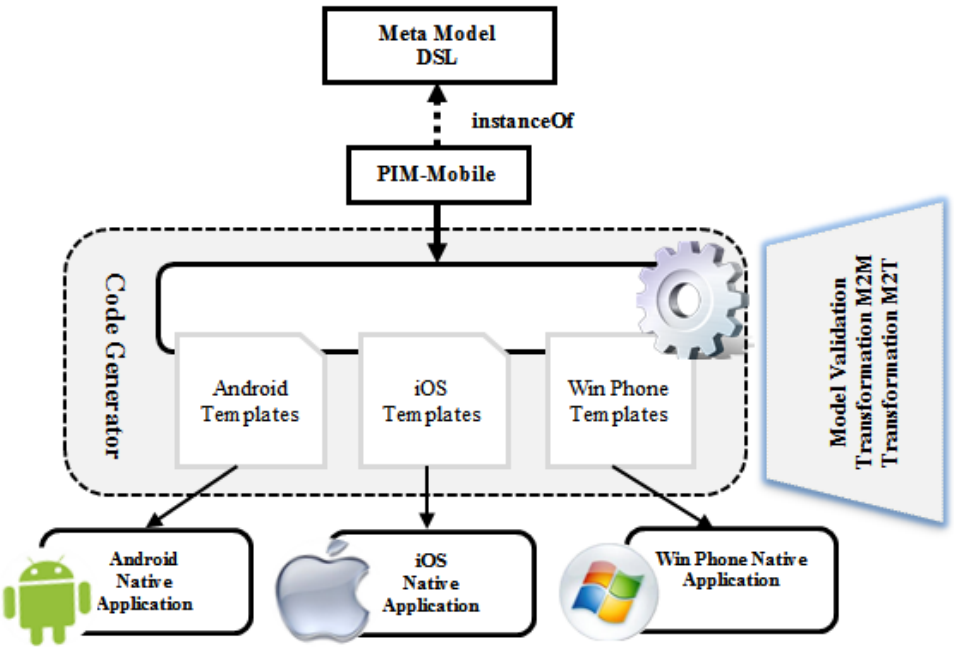


Figure 5. 3 - Schéma fonctionnel de la méthodologie proposée (*Lachgar et al, 2016*)

Cette architecture est structurée en trois niveaux principaux:

- **Niveau 1: Mobile Meta-model**

  Nous avons choisi un DSL pour mettre en œuvre notre méta-modèle; Le DSL présente un vocabulaire précis et concis pour un problème spécifique. Dans un DSL, tout est centré sur un problème technique ou un métier particulier. Toutefois, le langage UML est assez riche (*il couvre toutes les étapes d'un cycle de développement*) pour représenter le modèle, bien que, il peut offrir des fonctionnalités inutiles. Par conséquent, le processus MDA devient lourd à travers l'utilisation des modèles UML à chaque étape; Ainsi, la modélisation par DSL rend le processus plus pragmatique et plus simple, par rapport à une approche MDA basée sur UML.

  Ce méta-modèle prend en compte la modélisation des interfaces graphiques utilisateurs, la navigation entre les écrans, les menus, les composants multimédias, les événements, les types de composants (e.g. date, texte, email, numéro, etc.), les conteneurs, les fichiers de configuration et les ressources nécessaires pour l'application.

- **Niveau 2: PIM-Mobile**

  Après avoir défini le méta-modèle, nous pouvons ensuite créer un modèle textuel qui représentera l'application mobile. Dans cette étape, nous avons mis en œuvre des validateurs pour le modèle (par exemple en contrôlant à éviter la duplication des identificateurs des composants d'une même classe, les valeurs des attributs *width* et *height*, etc.), et nous avons également exploité les fonctionnalités par défaut pour la coloration syntaxique, auto-complétion de code, explorateur de classes, vue Plan et la détection des erreurs. Ainsi, l'utilisation d'un modèle textuel rend le travail d'un développeur plus facile, comparé à l'utilisation des diagrammes UML (par exemple, le diagramme d'objet pour modéliser des interfaces graphiques utilisateurs compliquées).

- **Niveau 3 : Code Generator**

  Le générateur de code est constitué de deux blocs principaux:

  - *Bloc de transformation*: La transformation PIM-Mobile vers le PSM a été effectuée avec Xtend 2. Pour chaque élément de notre PIM-Mobile, nous associons un ou plusieurs éléments du PSM des plateformes cibles ;
  - *Bloc de projection*: Une fois les transformations M2M sont effectuées, la deuxième étape consiste à générer le code source à partir du PSM obtenu, ce que nous appelons la projection. Les Template sont développés avec Xtend 2; Ce dernier fournit une amélioration par rapport à Xpand / Xtend, déjà utilisé dans certains travaux connexes.

  Dans la suite de ce chapitre, nous présenterons en détail chaque niveau.

5.4.1. Modèle indépendant de la plateforme

La première étape consiste à définir le méta-modèle décrivant le PIM mobile. Une option serait de réaliser un méta-modèle spécifique, qui présente les éléments et les concepts du domaine d'une application mobile; Pour ce faire nous avons adopté le méta-modèle Ecore (***Budinsky et al, 2003***) et le cadre logiciel Xtext, pour créer un langage textuel pour décrire le méta-modèle.

5.4.1.1. Description du méta-modèle avec Xtext

a. DSL pour modéliser les ressources d'une application

Une application mobile nécessite, entre autre la liste des fonctionnalités suivantes :

- *Gestionnaire d'informations personnelles (e.g. messages, contacts personnels, calendrier, etc.)*
- *Téléphone (e.g. appels téléphoniques sortants et entrants, SMS, etc.)*

- *Appareil photo (e.g. caméra intégrée permettant à l'utilisateur de prendre des photos et d'enregistrer des vidéos)*
- *Communication réseau (e.g. les piles de protocole, telles que le GPS, IP ou Bluetooth)*
- *Stockage externe (e.g. cartes mémoire supplémentaires pour augmenter la capacité de stockage du périphérique).*

```
grammar ma.ucam.MobileDsl with org.eclipse.xtext.common.Terminals
generate mobileDsl http://www.ucam.ma/MobileDsl
Application returns Application:
{Application}
'Application' name=EString
'{'
        ('root' root=EString)?
        ('icon' icon=EString)?
        ('author' author=EString)?
        ('version' version=EString)?
        ('author' author=EString)?
        ('backgroundImage' backgroundImage = STRING)?
        ('title' title=STRING)?
        ('ressources' '('
              ressources+= [Resource|EString] ( "," ressources+=[Resource|EString])* ')' )?
        ('dataBases' '('
              dataBase += [DataBase |EString] ( "," dataBase +=[DataBase |EString])* ')' )?

'}';
```

Voici un extrait du méta-modèle mobile, en termes de ressources, illustrant les caractéristiques ci-dessus (voir la figure ci-dessous pour plus de détails):

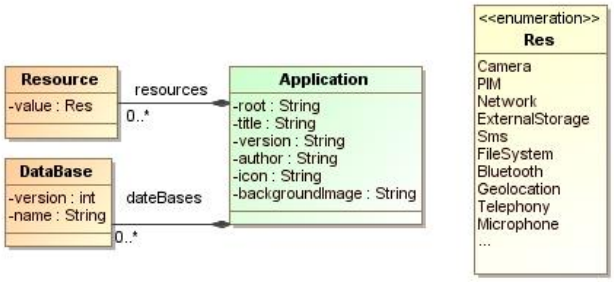


Figure 5. 4 - Un extrait du méta-modèle mobile: Ressources (*Lachgar et al, 2016*)

```
Resource returns Resource:
{Resource}
        ('resource' resource=Res)?
;
enum Res returns Res:
        Camera = 'Camera' | PIM = 'PIM' | Network = 'Network' | Sms = 'Sms' | Telephony =
        'Telephony':
```

b. DSL pour modéliser les écrans et les transitions

Les principaux concepts que nous voulons capturer sont les écrans, les menus, les éléments de menu et les transitions.

```
Application returns Application:
{Application}
'Application' name=EString
'{'
        ('root' root=EString)?
        ('icon' icon=EString)?
        ('author' author=EString)?
        ('version' version=EString)?
        ('author' author=EString)?
        ('backgroundImage' backgroundImage = STRING)?
        ('title' title=STRING)?
        ('screens' '('
                screens+=[Screen|EString] ( ","
                screens+=[Screen|EString])* ')' )?
'}';
```

```
Screen returns Screen:
{Screen}
'Screen'
'{'
        ('title' title=EString)?
        ('orientation' orientation=EString)?
        ('menu' menu=[Menu|EString])?
'}';
```

Un menu est composé d'un ensemble des sous menus :

```
Menu returns Menu:
{Menu}
'{'
        ( 'items' '(' subMenus+=SubMenu ("," subMenus+=SubMenu)* ')' )?
'}';
```

Un sous menu est caractérisé par un titre, une icône, et l'écran cible :

```
SubMenu returns SubMenu:
{SubMenu}
        ('title' title=STRING)?
        ('icon' icon=STRING)?
        ('target' target=STRING)?
;
```

c. DSL pour modéliser les composants de l'écran

Les principaux concepts que nous voulons capturer sont les différentes composants graphiques qui constituent un écran d'une application mobile (Layouts et Widgets), qui peuvent être des boutons, des champs de texte, et des regroupements des autres composants graphiques, pour lesquels nous pouvons définir des attributs communs (e.g. taille, positionnement, etc.).

```
Screen returns Screen:
{Screen}
'Screen'
'{'
        ('title' title=EString)?
        ('orientation' orientation=EString)?
        ('views' '('
                  views+=[View|EString] ( ","
                  views+=[View|EString])* ')' )?
'}';
```

Une vue peut être un Layout, un Widget ou Media.

```
View returns View:
        Layout | Widget | Media;
Widget returns Widget:
        Label | Input | Button | CheckBox | RadioButton | ListBox | WidgetGroup;
Layout returns Layout:
        RelativeLayout | LinearLayout | TableLayout | ScrollView;
```

Un composant de type *Label* est caractérisé par un ensemble d'attributs (width, height, marginTop, etc.).

```
Label returns Label:
{Label}
'Label'
'{'
        ('text' text=EString)?
        attribut= Attribut
'}';
```

Un champ de type *Input* peut être de type : texte, email, password, time, number, etc.

```
Input returns Input:
{Input}
'Input'
'{'
        ('text' text=EString)?
        ('type' inputType=InputType)?
        attribut= Attribut
'}';

enum InputType returns InputType:
          date = 'date' | text = 'text' |email = 'email' |password = 'password' |
          multiline = 'multiline' |time = 'time';
```

Un élément 'Bouton' est caractérisé par un ensemble d'attributs, parmi ces attributs, on trouve une icône et l'écran cible suite à un évènement sur le bouton.

```
Button returns Button:
{Button}
'Button'
'{'
        ('text' text=EString)?
        ('icon' icon=EString)?
        attribut= Attribut
        ('target' target=[View|EString])?
'}';
```

Un *CheckBox* et *RadioButton* sont représentés comme suit.

```
CkechBox returns CkechBox:
{CkechBox}
(checked?='checked')?
'CkechBox'
'{'
        ('text' text=EString)?
        attribut= Attribut
'}';

RadioButton returns RadioButton:
{RadioButton}
(checked?='checked')?
RadioButton'
'{'
        ('text' text=EString)?
        attribut= Attribut
'}';
```

Un groupe de widgets est une collection de composants tels qu'un groupe de boutons radio.

```
WidgetGroup returns WidgetGroup:
{WidgetGroup}
'WidgetGroup'
'{'
        ('text' text=EString)?
        attribut= Attribut
        ('widgets' '('
                    widgets+=[Widget|EString] ( ","
                    widgets+=[Widget|EString])* ')' )?
'}';
```

Un élément de type *ListBox* contient un ensemble des sous-éléments, chaque sous-élément est caractérisé par un titre, l'écran cible et le type d'événement appliqué.

```
ListBox returns ListBox:
{ListBox}
'ListBox'
'{'
        ('text' text=EString)?
        attribut= Attribut
        ('items' '('
                    items+=[ListBoxItem|EString] ( ","
                    items+=[ListBoxItem|EString])* ')' )?
'}'

ListBoxItem returns ListBoxItem:
{ListBoxItem}
'ListBoxItem'
'{'
        ('value' value=EString)?
        ('event' event=Event)?
        ('view' view=[View|EString])?
'}';
```

d. DSL pour modéliser les composants multimédia

La couche multimédia contient les technologies graphiques, audio et vidéo.

```
Video returns Video:
{Video}
'Video'
'{'
        attribut= Attribut
'}';

Controller returns Controller:
{Controller}
'Controller'
'{'
        attribut= Attribut
'}';
```

Chaque composant possède un attribut *event* permettant de spécifier l'évènement à appliquer sur le composant (e.g. click, touch, etc.):

```
enum Event {
        onClick, onTouch, keyUp ,keyDown
}
```

e. DSL pour modéliser une application qui utilise les capteurs embarqués

Une application mobile peut également utiliser les données renvoyées par les capteurs embarqués (e.g. accéléromètre, gravité, gyroscope, etc.), ces données seront envoyées dans

un délai bien déterminé.

Voici un extrait du méta-modèle mobile, en termes de capteurs, illustrant les caractéristiques ci-dessus (voir la figure ci-dessous pour plus de détails):

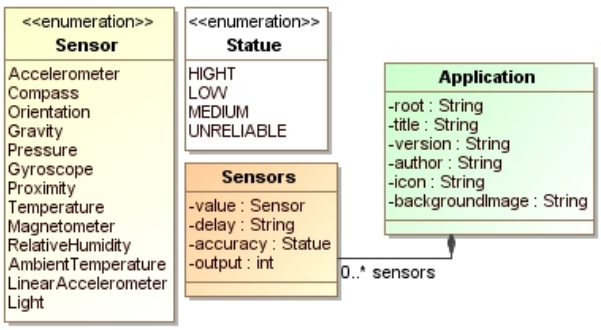


Figure 5. 5 - Un extrait du méta-modèle mobile: Capteurs (*Lachgar et al, 2016*)

```
Application returns Application:
{Application}
'Application' name=EString
'{'
        ('root' root=EString)?
        ('icon' icon=EString)?
        ('author' author=EString)?
        ('version' version=EString)?
        ('author' author=EString)?
        ('backgroundImage' backgroundImage = STRING)?
        ('title' title=STRING)?
        ('sensors' '(' sensors+=[Sensor|STRING] ( ","
              sensors+=[Sensor|STRING])* ')' )?
'}';

enum   Sensor :
    Accelerometer | Gravity | Gyroscope | Light | Orientation | Pressure | Proximity |
    Temperature;

Sensors:
    'Sensor' value = Sensor
    'Accuracy' accuracy = Statue
    'Delay' delay = STRING
    'Output output = INT;
```

5.4.1.2. Synthèse

Nous avons décomposé notre DSL en six parties : Application DSL, Screen DSL, Ressources DSL, Views DSL, Sensors DSL, Transitions DSL et Media DSL.

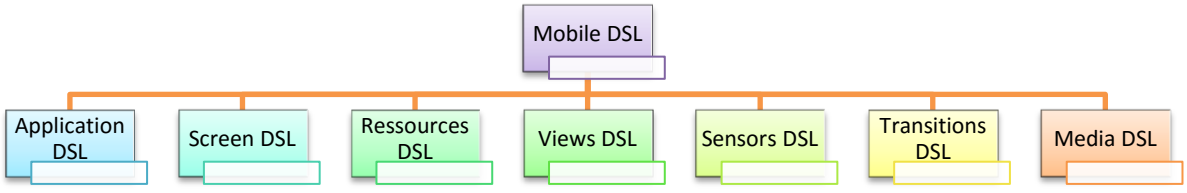


Figure 5. 6 - Composition de notre DSL

Nous pouvons résumer notre méta-modèle par le diagramme de classe suivant (*voir figure ci-dessous pour plus de détails*):

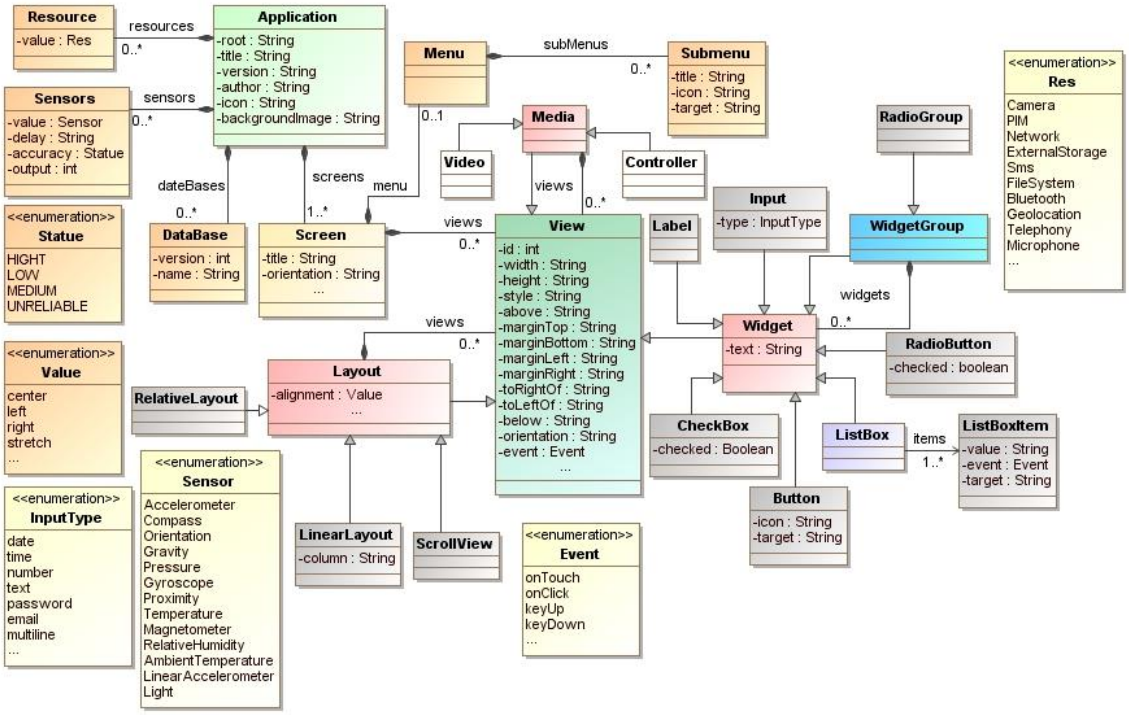


Figure 5. 7 - Méta-modèle proposé (*Lachgar et al, 2016*)

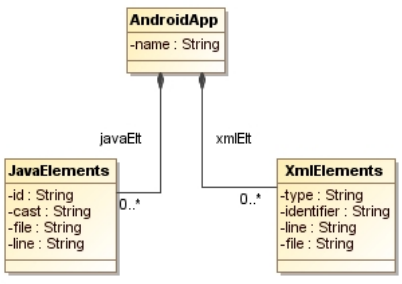
Le méta-modèle proposé couvre tous les aspects de la programmation mobile, tels que les ressources, les médias, les menus, les interfaces graphiques, la structure des classes, la gestion des transitions entre les écrans, etc.

5.4.2. Règles de transformation

5.4.2.1. Transformation de PIM-Mobile vers Android-PSM, Windows Phone-PSM et iPhone-PSM

a. Méta-modèle Android

Une application Android se compose d'une ou plusieurs activités; Une activité est divisée en deux parties: un fichier XML et un fichier Java; Par conséquent, le méta-modèle d'une application Android comporte deux catégories d'éléments: les éléments Java (JavaElements) et les éléments XML (XmlElements), chacun d'eux est caractérisé par certaines propriétés (*voir la figure ci-dessous pour plus de détails*).



JavaElements :
- Activity classes.

XmlElements :
- String File,
- Manifest File,
- Menu Files,
- Layout Files.

Figure 5. 8 - Méta-modèle d'une application Android (*Lachgar et al, 2016*)

b. Méta-modèle Windows Phone

Une application Windows Phone est composée d'un ou plusieurs PhoneApplicationPage. Une PhoneApplicationPage est composée d'une combinaison de deux fichiers XAML (eXtensible Application Markup Language) et C #. Par conséquent, le méta-modèle d'une application Widows Phone comporte trois catégories d'éléments: les éléments C # (CSharpElements), XML elemnts (XmlElements) et XAML (XamlElements), chacun d'eux est caractérisé par certaines propriétés (*voir la figure ci-dessous pour plus de détails*).

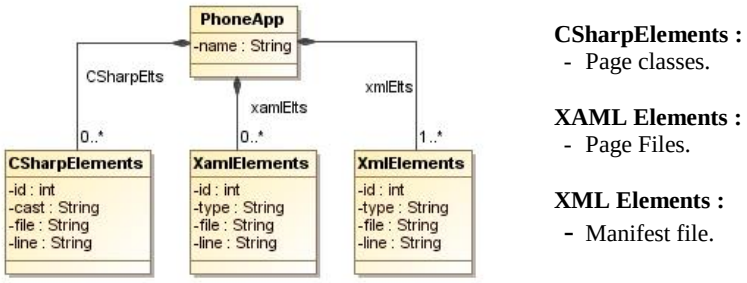


Figure 5. 9 - Méta-modèle d'une application Windows Phone (*Lachgar et al, 2016*)

c. Méta-modèle iOS

Une application iOS est composée d'un ou plusieurs écrans; Chaque écran lui-même est composé d'une combinaison de fichiers (.h), de fichiers (.m) et de fichiers (.xib), chacun d'eux est caractérisé par certaines propriétés (*voir la figure ci-dessous pour plus de détails*).

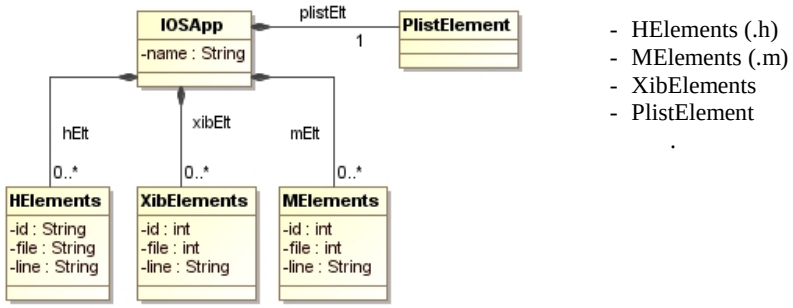


Figure 5. 10 - Méta-modèle d'une application iOS (*Lachgar et al, 2016*)

HElements (.h) et **MElements** (.m) - Les fichiers avec extension .h se réfèrent aux fichiers d'en-tête, tandis que ceux avec l'extension (.m) sont les fichiers d'implémentation. Comme dans la plupart des langages de programmation, le code source d'Objective-C est divisé en deux parties: interface et implémentation.

XibElements - Pour les fichiers avec extension (.xib), ce sont les fichiers Interface Builder qui stockent l'interface utilisateur (UI) de l'application.

PlistElement - (.plist) contient les clés les plus critiques requises par le système. Les clés et les valeurs du fichier décrivent différents aspects du paquet. Le système utilise ces clés et ces valeurs pour obtenir des informations sur l'application et sa configuration.

d. Diagramme de transformation et génération de code pour la plateforme Android

Le diagramme ci-dessous illustre les différentes étapes de génération d'une application Android, basées sur un modèle conforme à notre DSL (*voir la figure ci-dessous plus de détails*):

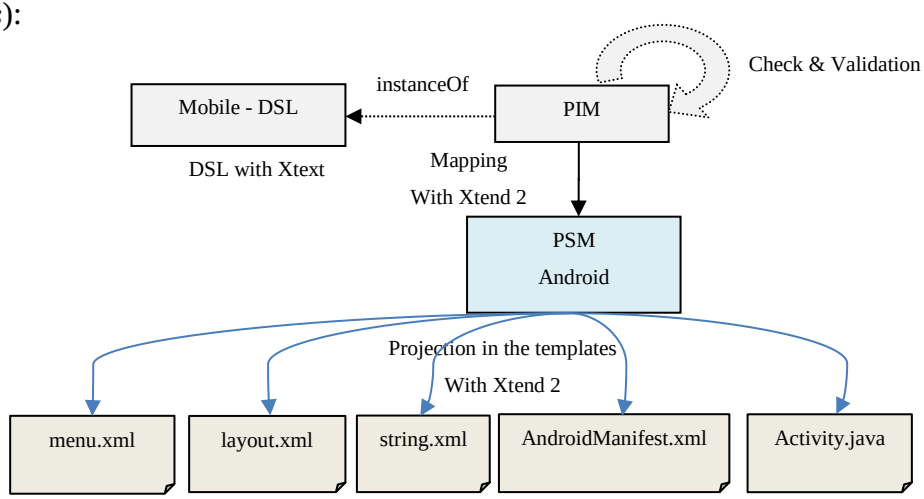


Figure 5. 11 - Diagramme de transformation et génération de code pour la plateforme Android (*Lachgar et al, 2016*)

e. Diagramme de transformation et de génération de code pour la plateforme Windows Phone et la plateforme iOS

Les deux schémas ci-dessous illustrent les différentes étapes de génération d'une application Windows Phone (*le diagramme à gauche*) et une application iOS (*le diagramme à droite*), basée sur un modèle conforme à notre DSL (*voir la figure ci-dessous plus de détails*) :

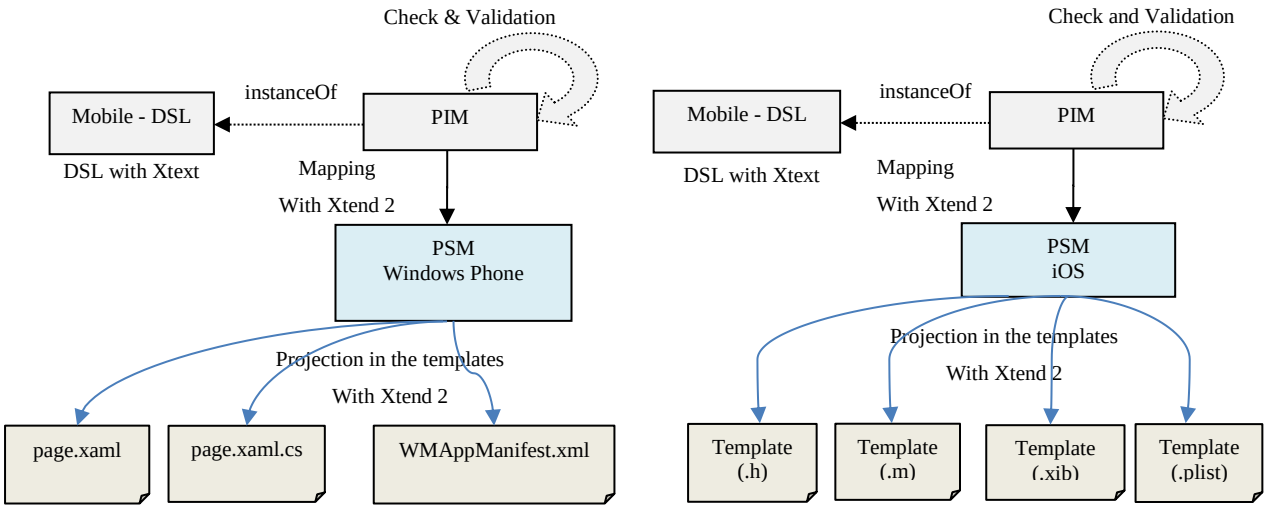


Figure 5. 12 - Diagramme de transformation et de génération de code pour la plateforme Windows Phone et la plateforme iOS (*Lachgar et al, 2016*)

f. Mappage des éléments

La figure ci-dessous présente les différentes transformations, basées sur le modèle indépendant de la plateforme, vers les différents modèles ciblés (*voir le tableau ci-dessous pour plus de détails*):

Tableau 5. 1 – Mappage des éléments (*Lachgar et al, 2016*)

PIM	Android PSM	Windows Phone PSM	iOS PSM
Screen	Activity	PhoneApplicationPage	UIWindows
Widget	Widget	ContentControl	UIControl
View	View	FramworkElement	UIView
Label	TextView	TextBlock	UILabel
Button ▪ type = image ▪ type = simple	 ImageButton Button	 ImageButton Button	 UIImageButton UIButton
CheckBox	CheckBox	CheckBox	UIButton with image UIImage
RadioButton	RadioButton	RadioButton with GroupName attribute	UISwitch (two choices) UIPicker (Three or more choices) UISegmentedControl
Input ▪ type = text ▪ type = date ▪ type = multiline ▪ type = email	EditText ▪ inputType = text ▪ inputType = date ▪ inputType = textMultiLine ▪ inputType = textEmailAddress	TextBox ▪ InputScope="Default" ▪ InputScope="Date" ▪ InputScope="EmailSmtpAddress" ▪ TextWrapping="Wrap" AcceptsReturn="True" VerticalScrollBarVisibility="Visible"	UITextField ▪ keyboardType =UIKeyboardTypeDefault ▪ inputView = UIDatePicker ▪ UITextView ▪ keyboardType = UIKeyboardTypeEmailAddress
ListBox	ListView	ListBox	UITableView
Video	VideoView	MediaElement	UIVideoView
Controller	MediaController	MediaElement	UIViewController
LinearLayout	LinearLayout	IF column <> null then Grid IF column == null then StackPanel	CSLinearLayoutView
ScrollView	ScrollView	ScrollViewer	UIScrollView
RadioGroup	RadioGroup	StackPanel	has no equivalent
RelativeLayout	RelativeLayout	Grid	AutoLayout

g. Mappage des permissions de ressources et de capteurs

La figure ci-dessous présente les différentes transformations, à partir de modèle indépendant de la plateforme qui représente les ressources et les capteurs, vers les différents modèles ciblés (*voir tableau ci-dessous pour plus de détails*):

Tableau 5. 2 - Mappage des permissions de ressources et de capteurs (*Lachgar et al, 2016*)

PIM	**Android PSM (android.permission)**	**Windows Phone PSM**	**iOS PSM (key of UIRequiredDeviceCapabilities)**
Camera	CAMERA	ID_CAP_CAMERA ID_CAP_ISV_CAMERA	camera-flash video-camera
Network	INTERNET	ID_CAP_NETWORKING	bluetooth-le gps location-services wifi peer-peer
Phone	CALL_PHONE SEND_SMS	ID_CAP_PHONEDIALER	Sms Telephony
PIM	READ CONTACTS WRITE CONTACTS	ID_CAP_APPOINTMENTS ID_CAP_CONTACTS	has no equivalent
ExternalStorage	WRITE EXTERNAL STORAGE READ_EXTERNAL_STORAGE	ID_CAP_MEDIALIB	has no equivalent
Audio	CAPTURE_AUDIO_OUTPUT MODIFY_AUDIO_SETTINGS RECORD_AUDIO	ID_CAP_MEDIALIB_AUDIO	Audio
Video	CAPTURE_SECURE_VIDEO_OUTPUT CAPTURE_VIDEO_OUTPUT	ID_CAP_ISV_CAMERA	front-facing-camera video-camera
Microphone	MICROPHONE (permission-group)	ID_CAP_MICROPHONE	Microphone
Magnetometer	BODY_SENSORS (Sensor.TYPE_MAGNETIC_FIELD)	ID_CAP_SENSORS	Magnetometer
Gyroscope	BODY_SENSORS (Sensor.TYPE_GYROSCOPE)	ID_CAP_SENSORS	Gyroscope
Accelerometer	BODY_SENSORS (Sensor.TYPE_ACCELEROMETER)	ID_CAP_SENSORS	Accelerometer

5.5. Exemple illustratif

Dans cette section, nous présentons l'étape finale de la conception pilotée par modèle (MDD), qui est la génération du code de l'application pour chaque plateforme spécifique.

Les technologies mobiles ont ouvert un nouveau horizon dans le domaine de la collecte de données, offrant des possibilités extraordinaires simultanément: instantanéité, portabilité extrême, ergonomie et facilité d'utilisation, communication en temps réel, connexion à des bases de données, localisation GPS, capture photo et vidéo.

De ce point de vue, nous proposons une application mobile simple pour recueillir des informations sur un site Web de films en streaming. Cette application se compose de trois écrans; Chaque écran contient un ensemble de composants et un menu de navigation entre les écrans.

5.5.1.Analyse et création de modèle

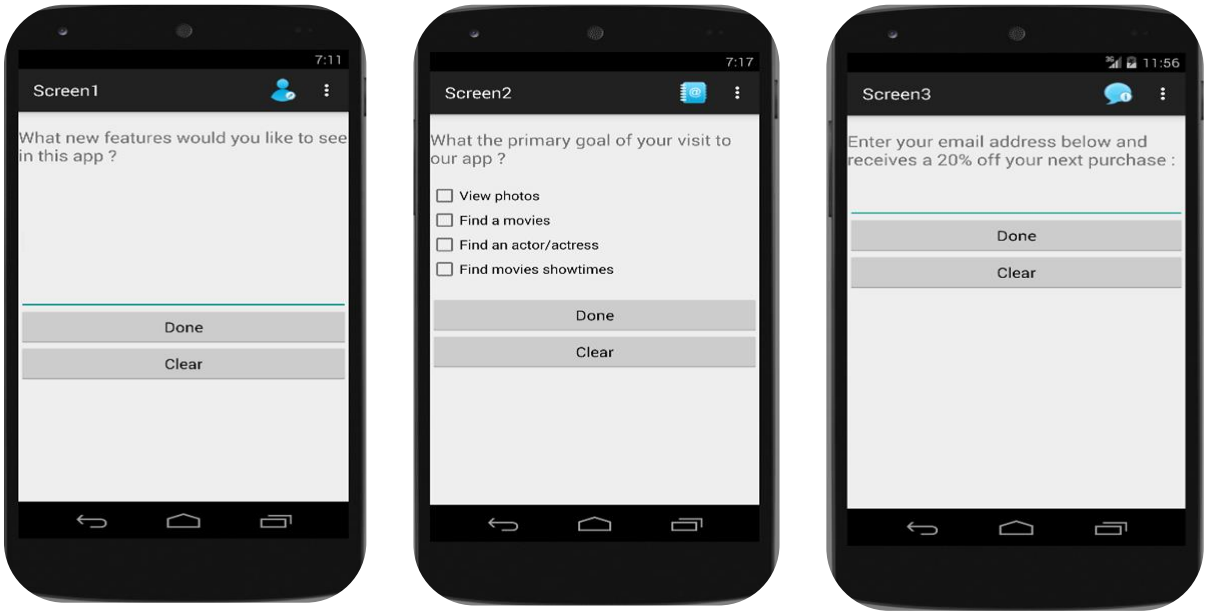
L'objectif de cette étape est de concevoir les différents éléments, qui représentent l'application mobile, en utilisant un modèle correspondant à notre DSL, comme illustré ci-dessous (*voir la figure ci-dessous pour plus de détails*).

```
Application "Survey App"{
        root "C://survey" version "1" author "Author" icon "icon"
        resources(
                resource Network
        )
        dataBases (
                name "Survey" version "1"
        )
        screens (Screen{
                 title "Screen1" orientation "portrait"
                 menu{
                        items(
                                title "Screen2" icon "screen2"  target "Screen2",
                                title "Screen3" icon "screen3"  target "Screen3"
                         )
                 }
                 Views (
                        LinearLayout{id 1 width "match" height "match" orientation "vertical" views (
                                Label {id 1  width "match" height "wrap"  text "What new features would you like
                                        to see in this app ?"},
                                Input {id 1  width "match" height "wrap" type multiline},
                                Button {id 1  width "match" height "wrap" event onClick text "Done" },
                                Button {id 2 width "match" height "wrap" event onClick text "Clear"}
                                )
                        }
                )
                },
                Screen {
                  title "Screen2" orientation "portrait"
                  menu {
                        items (
                                title "Screen1" icon "screen1" target "Screen1",
                                title "Screen3" icon "screen3" target "Screen3"
                            )
                  }
                  views (
                         LinearLayout{id 1 width "match" height "match" orientation "vertical" views(
                                Label {id 1 width "match" height "wrap" text "What the primary goal of your visit
                                        to our app ?"},
                                CheckBox {id 1 width "match" height "wrap" text "View photos"},
                                CheckBox{id 2 width "match" height "wrap" text "Find a movies"},
                                CheckBox{id 3 width "match" height "wrap" text "Find an actor/actress"},
                                CheckBox{id 4 width "match" height "wrap" text "Find movies showtimes"},
                                Button {id 1 width "match" height "wrap" event onClick  text "Done" target
                                        "Screen3"},
                                Button {id 2 width "match" height "wrap" event onClick  text "Clear"}
                                )
                              }
                        )

                },
                Screen {
                  title "Screen3" orientation "portrait"
                  menu {
                        items(
                                title "Screen1" icon "screen1" target "Screen1",
                                title "Screen2" icon "screen2" target "Screen2"
                            )
                  }
                  views (
                        LinearLayout{id 1 width "match" height "match" orientation "vertical" views (
                                Label {id 1 width "match" height "wrap" text "Enter your email address below and
                                        receives a 20% off your next purchase :"},
                                Input {id 1  width "match" height "wrap" type email},
                                Button {id 1 width "match" height "wrap" event onClick text "Done"},
                                Button {id 2 width "match" height "wrap" event onClick text "Clear"}
                                )
                               }

                        )
            }
        )
}
```

Figure 5. 13 - Modèle qui représente notre application de sondage (*Lachgar et al, 2016*)

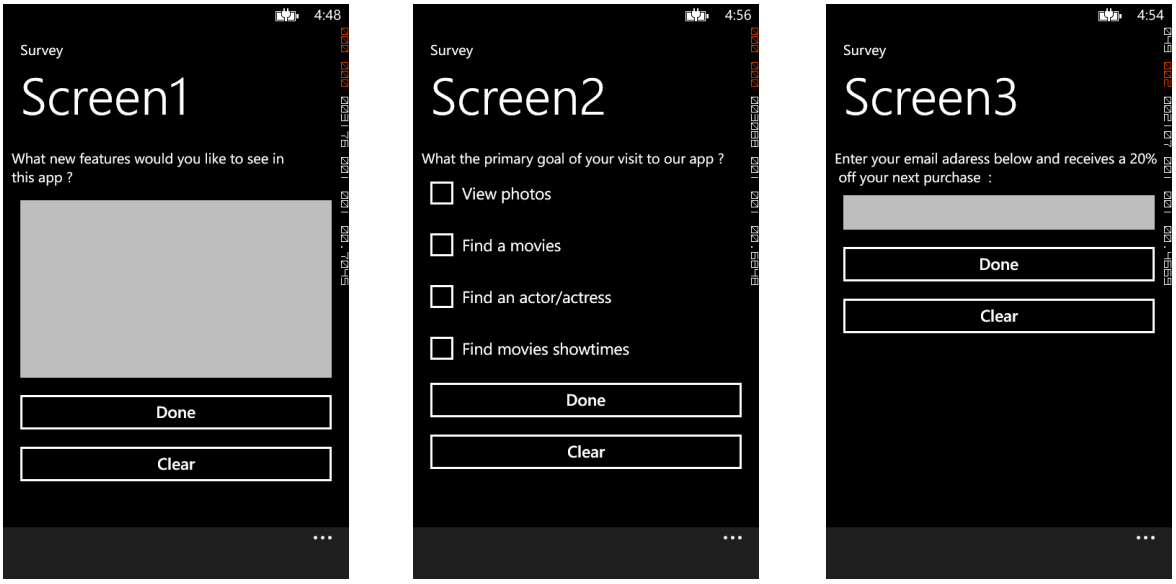
5.5.2. Interface graphique utilisateur générée pour les plateformes Android, Windows Phone et iOS

Nous pouvons automatiquement générer les fichiers sources dans un projet Android, dans un projet Windows Phone ou dans un projet iOS à partir d'un modèle conforme avec notre DSL. Les figures 5.14, 5.15 et 5.16 illustrent les résultats obtenus pour chaque OS:



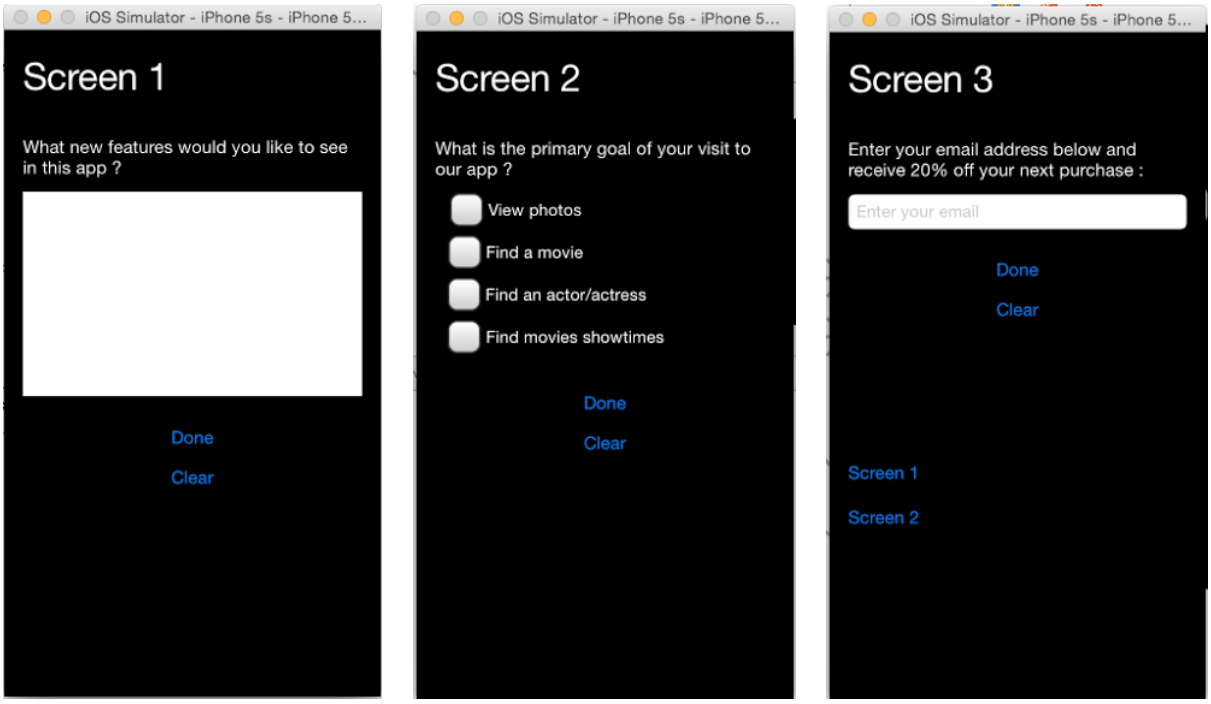
The choice of features to be displayed in the app *Mentioning the main motive of the visit* *Benefiting from a markdown by typing one's email*

Figure 5. 14 - GUI et Menu générés visant la plateforme Android (*Lachgar et al, 2016*)



The choice of features to be displayed in the app *Mentioning the main motive of the visit* *Benefiting from a markdown by typing one's email*

Figure 5. 15 - GUI et Menu générés visant la plateforme Windows Phone (*Lachgar et al, 2016*)



The choice of features to be displayed in the app *Mentioning the main motive of the visit* *Benefiting from a markdown by typing one's email*

Figure 5. 16 - GUI et Menu générés visant la plateforme iOS (*Lachgar et al, 2016*)

5.6. Limitations de ce travail

Le générateur de code suggéré est basé sur un nouveau langage dédié, pour le développement des applications mobiles multiplateformes. En conséquence, le développeur doit apprendre un nouveau langage pour utiliser la solution, afin d'assurer un bon démarrage du projet. D'autre part, ce générateur de code ne permet pas de convertir les applications existantes (*reverse engineering*) Android vers iOS, ou Windows phone, ou vice versa, sans avoir à réécrire le modèle conforme avec le DSL en premier lieu. En outre, les applications générées ne respectent pas une architecture en couches (*séparation des couches*), mais surtout se concentrer sur le développement de nouvelles applications mobiles depuis le début sans bénéficier de code des applications mobiles existantes.

5.7. Conclusion

La principale contribution de ce chapitre est une approche pilotée par les modèles à base de MDA, afin de générer le code natif pour les applications mobiles. Pour ce faire, nous avons défini un nouveau méta-modèle textuel, pour la création d'un modèle indépendant de la plateforme pour les applications mobiles, selon une architecture guidée par les modèles. Ensuite, nous avons montré comment les modèles sont conçus à l'aide de DSL, et comment ils sont transformés en interfaces graphiques, menus, structure des classes, transitions entre les écrans, événements sur les composants graphiques, fichiers de configuration indépendamment des plateformes mobiles cibles (e.g. Android, Windows phone 8, iOS etc.), en utilisant des

modèles paramétrés, permettant au développeur d'éviter d'écrire du code redondant.

Un exemple illustrant est discuté dans ce chapitre, qui montre les différentes étapes suivies au cours de notre approche, afin de générer le code natif pour les applications mobiles. Le grand avantage de notre méthode est l'utilisation d'un modèle textuel. En effet, il offre aux développeurs un mécanisme simple pour produire des applications multiplateformes, contrairement aux méthodes basées sur la notation UML.

Dans le chapitre suivant, nous étendrons notre approche, pour prendre en compte la génération des applications multiplateformes selon une architecture en couche, également, nous présentons les différentes méta-modèles proposées afin de générer, la couche accès aux données, la couche métier, la couche accès aux services réseaux, etc. dans une application mobile.

Chapitre 6

Approche pragmatique pour la modélisation et la génération des applications mobiles multiplateformes

6.1. Introduction

L'univers du développement d'applications mobiles multiplateformes évolue de manière constante et se dirige vers une uniformisation des codes. En effet, de plus en plus des environnements mobiles permettent de créer des applications sur la base d'un seul langage tout en permettant leur distribution sur des systèmes variés. Plusieurs moyens existent et offrent des avantages variés selon les différentes besoins.

Dans ce sens nous avons classé ces approches en trois catégories l'approche native, l'approche hybride et l'approche web (***Lachgar et al, 2017***). Dans un monde idéal, sans contraintes de temps et d'argent, il serait évidemment et plus intéressant de passer vers une solution native. Le résultat présente des avantages en termes d'ergonomie, de performance et d'intégrité. Par contre, les applications natives sont très coûteuses à mettre en œuvre, limitée à une plateforme mobile particulière, nécessitent une collection de connaissances approfondies et des langages de programmation pour les réaliser (***Lachgar et al, 2017***).

Par conséquent, pour remédier aux faiblesses de l'approche native nous avons suggéré dans le chapitre précédant, une approche basée sur le DSL et nous avons constaté que le DSL seul ne permettra pas de modéliser une application métier qui respecte une architecture en couche. Cependant, dans ce chapitre, nous suggérons une approche permettant aux développeurs de générer des applications mobiles natives et cela en combinant le langage UML et le langage dédier (DSL) pour modéliser ces applications et par la suite générer le code pour ces systèmes variés selon une architecture en couches.

6.2. Architecture d'une application mobile multiplateforme

Le principe clé de la construction d'une application multiplateforme est de créer une architecture qui se prête une maximisation du partage de code sur les différentes plateformes et de permettre une réutilisation de code. Les principes de la programmation orientée objet aident à construire une application bien architecturée. Parmi ces principes on cite :

- *Encapsulation*- S'assurer que les classes et même les couches architecturales, n'exposent qu'une API minimale qui exécute leurs fonctions prétendues et masque les détails d'implémentation (***Amstrong, 2006***).
  - Au niveau de la classe, cela signifie que les objets se comportent comme des «boîtes noires» et que le code consommateur n'a pas besoin de savoir comment ils accomplissent leurs tâches ;
  - Au niveau architectural, cela implique la mise en œuvre de modèle comme Façade qui encourage une API simplifiée qui orchestre des interactions plus complexes au

nom du code dans des couches plus abstraites. Cela signifie que le code UI (par exemple) ne doit être responsable que de l'affichage des écrans et de l'acceptation des entrées utilisateurs; Et ne jamais interagir directement avec la base de données. De même, le code d'accès aux données devrait uniquement lire et écrire dans la base de données, mais ne jamais interagir directement avec les boutons ou les champs textes.

- *Séparation des responsabilités* - S'assurer que chaque composante (au niveau de l'architecture et de la classe) a un objectif clair et bien défini. Chaque composant doit exécuter uniquement ses tâches définies et exposer cette fonctionnalité via une API accessible aux autres classes (couches) qui doivent l'utiliser.
- *Polymorphisme* - La programmation vers une interface (*ou classe abstraite*) prenant en charge plusieurs implémentations, signifie que le code de base peut être écrit et partagé entre plateformes tout en interagissant avec des fonctionnalités spécifiques à la plateforme (***Amstrong, 2006***).

Le résultat naturel est une application modelée basée sur des entités abstraites avec des couches logiques distinctes. La séparation des couches rend les applications plus faciles à comprendre, à tester et à maintenir. Il est recommandé que le code de chaque couche soit physiquement séparé (dans des répertoires ou même des projets séparés pour des applications très importantes) ainsi que logiquement séparé (à l'aide d'espaces de noms ou package).

6.2.1. Couches typiques d'une application mobile

Dans le présent chapitre et dans l'étude de cas, nous nous référons aux six couches d'application suivantes (*voir la figure 6.1 ci-dessous pour plus de détails*):

Description des différentes couches :

- *Data Layer* : Persistance non volatile des données, susceptible d'être une base de données SQLite, mais pouvant être implémentée avec des fichiers XML ou tout autre mécanisme approprié.
- *Couche d'accès aux données* : Dans cette couche on trouve les classes permettant de gérer tout ce qui concerne l'accès aux données. Cette couche peut être divisée en plusieurs parties :
  - Une partie pour tout ce qui concerne la définition de la base de données ;
  - Une partie pour tout ce qui concerne la définition des opérations de base sur la base de données telles que (l'ajout, la modification, la suppression et la consultation) ;

- Une partie pour définir la connexion à la base de données cible.

- *Business Layer* : Cette couche contient les définitions des entités métiers (Les modèles), correspondant aux données métier à manipuler dans l'application. Cependant, cette couche s'occupe de garantir que le métier de l'application est respecté, que les règles liées aux données sont toujours suivies.
- *Couche d'accès au service* : Utilisé pour accéder aux services dans le cloud: des services Web complexes (REST, SOAP, etc.) à la simple récupération de données et d'images à partir de serveurs distants. Encapsule le comportement réseau et fournit une API simple à être consommée par les couches Application et UI.
- *Couche d'application* : Code typiquement spécifique à la plateforme (généralement non partagé entre plateformes) ou code spécifique à l'application (généralement non réutilisable). Un bon test pour déterminer si le code doit être placé dans la couche d'application par rapport à la couche d'interface utilisateur est (a) pour déterminer si la classe a des contrôles d'affichage réels ou (b) s'il peut être partagé entre plusieurs écrans ou périphériques (par exemple, iPhone et iPad).
- *Couche de l'interface utilisateur (UI)* : la couche face à l'utilisateur contient des écrans, des widgets et des contrôleurs qui les gèrent.

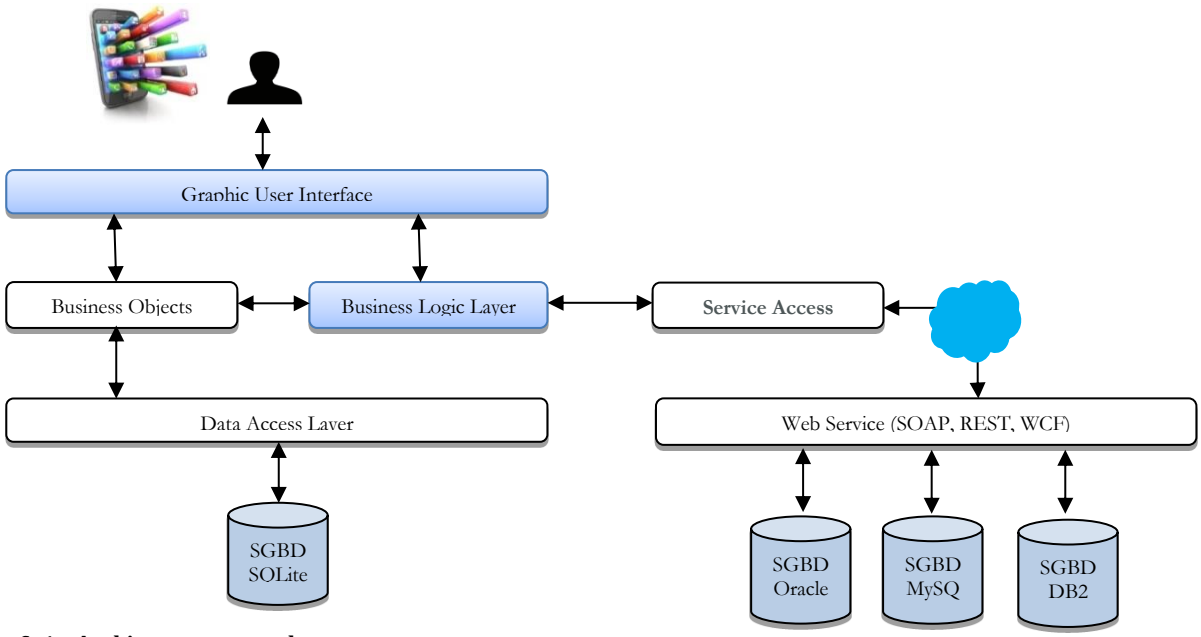


Figure 6. 1 - Architecture en couches

Cette architecture présente plusieurs avantages par rapport à la façon traditionnelle utilisée pour mettre en place les applications informatiques, nous citons :

- La maintenance des données est indépendante du support physique de stockage ;

- La maintenance des traitements est simplifiée : des membres de l'équipe travaillent sur la couche d'accès aux données, un autre peut tout à fait travailler sur la couche métier ou sur l'interface graphique sans perturber le travail des autres ;
- La gestion des traitements depuis la couche de présentation est facilitée ;
- Le travail en équipe est optimisé ;
- La migration d'un environnement graphique à un autre est relativement simple.

Cependant, une application ne peut pas nécessairement contenir tous les couches - Par exemple la couche d'accès au service n'existerait pas dans une application qui n'accède pas aux ressources réseau. Une application très simple peut fusionner la couche de données et la couche d'accès aux données car les opérations sont extrêmement basiques.

6.2.2. Modèles de conception courants dans le développement mobile

Les modèles sont un moyen établi pour capter les solutions récurrentes aux problèmes communs. Il existe quelques modèles clés qui sont utiles à comprendre dans la construction d'applications mobiles maintenables et compréhensibles.

- *Modèle, vue, contrôleur (MVC)* : Un modèle commun et souvent mal compris, MVC est le plus souvent utilisé lors de la création d'interfaces utilisateur et permet une séparation entre la définition réelle d'un écran -l'interface utilisateur- (*View*), le moteur qui gère l'interaction (*Contrôleur*) et les données qui le remplissent (*Modèle*). Le modèle est en fait une pièce complètement optionnelle et donc, le noyau de la compréhension de ce modèle se trouve dans la vue et le contrôleur (***Plakalovic et al, 2010***).
- *Business Façade* : Fournit un point d'entrée simplifié pour les travaux complexes. Par exemple, dans une application de suivi des projets, vous pouvez avoir une classe ProjectManager avec des méthodes telles que findAll (), findById (id), create (project), etc. La classe ProjetctManager fournit une Façade au fonctionnement interne de l'enregistrement / récupération des objets projet (***Jiang et al, 2011***).
- *Singleton* : Le modèle Singleton fournit un moyen par lequel une seule instance d'un objet particulier peut exister (***Stencel et al, 2008***). Par exemple, lors de l'utilisation de SQLite dans les applications mobiles, vous ne voulez qu'une seule instance de la base de données. L'utilisation du modèle Singleton est un moyen simple d'assurer cela.
- *Abstract factory*: Un modèle permettant la réutilisation du code à travers les applications. Le code partagé peut être écrit sur une interface ou une classe abstraite et des implémentations concrètes spécifiques à la plateforme sont écrites et transmises lorsque le code est utilisé (***Sarcar, 2016***).

- *DAO (Data Access Object pattern)* : permet de faire le lien entre la couche métier et la couche persistante, ceci afin de centraliser les mécanismes de mapping entre le système de stockage et les objets métiers.
- *Async* : Le pattern Async est utilisé lorsqu'une tâche de longue durée doit être exécutée sans tenir compte de l'interface utilisateur ou le traitement en cours. Dans sa forme la plus simple, le modèle Async décrit simplement que les tâches de longue durée doivent être lancées dans un autre thread (ou une abstraction de thread similaire, telle qu'une tâche) pendant que le thread actuel continue à traiter et écoute une réponse du processus en arrière-plan. Puis met à jour l'interface utilisateur lorsque les données et / ou l'état est renvoyé.

6.3. Approche proposée

Pour ce faire nous proposerons de suivre une modélisation pragmatique basée dans un premier lieu sur le langage UML et en deuxième lieu sur un langage dédié DSL. À partir des diagrammes UML en particulier le diagramme de classe nous pouvons générer les classes métiers, les classes permettant l'accès aux données, les classes permettant de définir les opérations de base pour une basse de données SQLite telles que la création des tables et la suppression des tables. Concernant le langage dédié, il va nous servir pour modéliser les interfaces graphiques utilisateurs par conséquence la génération de la couche présentation et la couche logique, également avec un langage dédie nous pouvons générer la couche accès aux services.

6.3.1. Génération des couches DAL, BOL et DL

Pour la génération des couches métier et accès aux données, nous sommes basé sur le méta modèle UML, en particulier celui d'un diagramme de classe. En effet, à partir d'un diagramme de classe nous pouvons générer les classes métiers et les classes accès aux données en différents langages (Java, C#, Objective-C, Swift, C++, etc). En outre, nous pouvons également générer le code SQL permettant la création de la base de données sous SQLite par exemple.

Avec le diagramme de classe nous avons pu générer les interfaces graphiques classiques de mise à jour et d'interrogation des données, en faisant appel aux CRUD préalablement générés. L'architecture pour la génération des couches DAL, BOL et DL est présentées dans la figure ci-dessous.

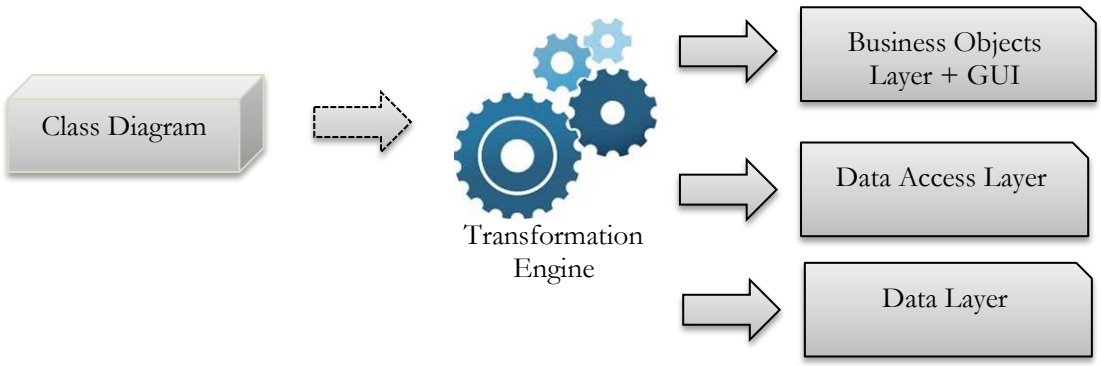


Figure 6. 2 - Architecture pour la génération des couches DAL, BOL et DL

Pour réaliser les différentes transformations nous avons utilisé le langage ATL. En effet, ce langage de transformation hybride, il est à la fois déclaratif et impératif, ce qui le rend plus expressif et lui accorde la possibilité d'exprimer toute sorte de transformations. En ce qui concerne les performances ATL dans la plupart des cas s'exécute plus vite que QVT (Adopté dans quelques travaux tels que (***Benouda et al, 2016a, 2016b***)) dû à deux raisons principales ; la première : il est plus facile de réduire la mise en correspondance avec la clause WHERE dans les règles ; la deuxième : dû au fait que ATL est compilé et exécuté sur une machine virtuelle. ATL permet de réaliser des transformations entre les modèles source et cible par le biais d'un ensemble de règles de correspondance ou de mappage écrites dans ce langage. En ATL, nous pouvons créer des modules permettant d'effectuer des transformations modèle-à-modèle. Cependant, pour les transformations modèle-à-texte nous avons fait recours à Xtend, ce dernier permet de projeter les données dans des Templates à partir d'un modèle XMI instance de PIM Bean ou de PIM DATABASE (*voir figure 6.7 pour plus de détails*) résultat des transformations modèle-à-modèle réalisées avec l'ATL. Un extrait de code permettant de charger un fichier XMI est présenté dans la figure ci-dessous.

```
class Generator {

	def static void main(String[] args) {
		new Generator().generate("model.xmi")
	}

	def dispatch generate(List<Beans> beans) '''

	'''
}
```

Figure 6. 3. Génération de code avec Xtend à partir d'un modèle non textuel

La figure ci-dessous illustre les différentes étapes de notre approche pour la génération des couches DAL, BOL et DL.

(1) Modélisation d'une application à l'aide d'un diagramme de classe ;

(2) Transformation vers une instance du modèle PIM Bean ;

(3) Projection dans des Template pour la génération des classes métiers et les interfaces graphiques standards à partir d'une instance de PIM Bean ;

(4) Transformation d'une instance de PIM Bean vers une instance de PIM DataBase ;

(5) Projection dans des Template pour la génération des classes accès aux données, la base de données à partir d'une instance de PIM DataBase.

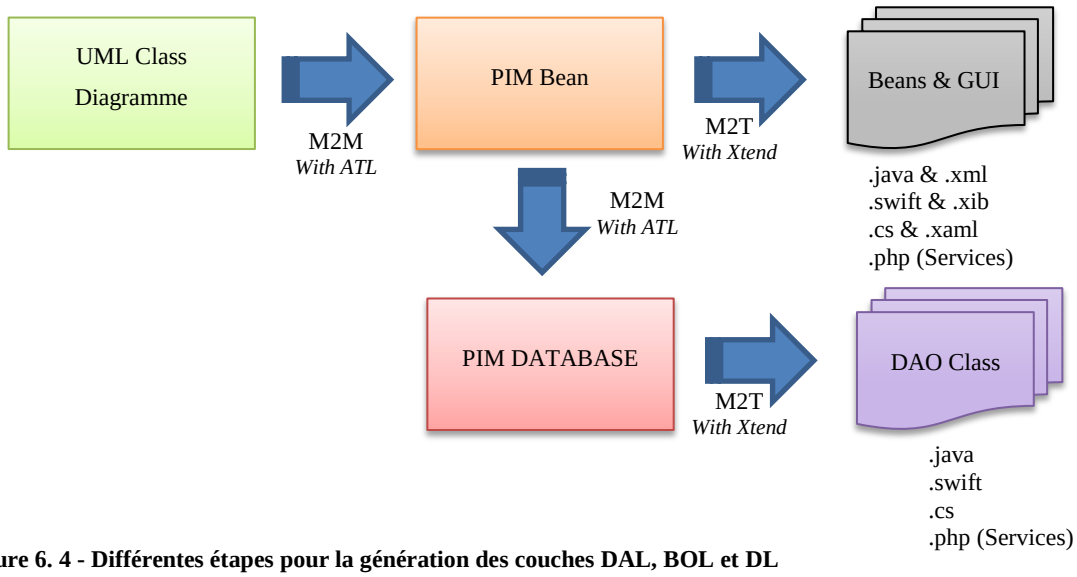


Figure 6. 4 - Différentes étapes pour la génération des couches DAL, BOL et DL

6.3.2. Génération des couches GUI, BLL et SAL

Pour la génération de la couche présentation et la couche application nous nous sommes basés sur un méta-modèle développé à l'aide d'un DSL (***Lachgar et al, 2016***) que nous avons amélioré d'avantage dans la suite de ce chapitre (*voir figure 6.11 pour plus de détails*). Ce dernier, prend en considération tous les composants principales constituant une application mobile, tel que les composants graphiques (bouton, zone de texte, les listes, les conteneurs, les menus, etc.), la navigation entre les écrans, la spécification des capteurs qui vont être utilisés dans l'application (Compass, Accelerometer, Orientation, Light sensor, etc.) et la spécification des fonctionnalités natives demandés dans une application (Camera, SMS, Telephony, Storage, Alert, Vibration, Geolocation, Contacts, etc.). Nous avons également entendu le méta-modèle pour prendre en charge la modélisation des accès aux différents services réseaux. L'architecture pour la génération des couches GUI, BLL et SAL est présentées ci-dessous.

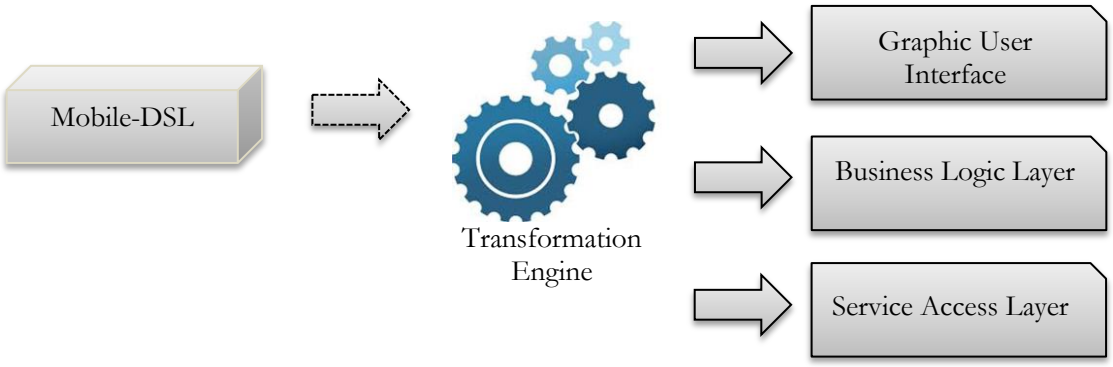


Figure 6. 5 - Architecture pour la génération des couches GUI, BLL et SAL

Pour réaliser les différentes transformations dans cette deuxième étape nous avons fait recours au langage Xtext pour la création de DSL et le langage Xtend 2 pour réaliser les différentes transformations et la génération des différentes couches, la génération de code avec Xtend2 est plus rapide et performent par rapport à l'utilisation de Xpand (*Adopter dans quelques travaux tels que (**Heitkötter et al, 2013**)*), car les templates dans Xtend2 sont compilés en avance, et non interprété comme dans Xpand. Xtext 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. Xtend 2 offre une spécification flexible et modulaire du code généré à travers la gestion d'imports et d'aspects. De plus, les règles de génération de chaque entité du modèle supportent le polymorphic dispatch. Il s'agit d'une extension du patron de conception visiteur permettant à un objet de visiter la fonction adaptée à son type. Dans le cas du polymorphic dispatch, et à l'inverse du visiteur, aucun artefact intrusif n'est nécessaire dans le code du modèle pour obtenir ce comportement. Ce sont les méthodes visitées elles-mêmes qui définissent le type d'objet qu'elles supportent. Ceci est particulièrement utile dans un compilateur où une représentation intermédiaire est souvent décrite par un arbre de syntaxe abstraite dont les nœuds sont des spécialisations d'une unique définition abstraite.

Dans les sections qui suivent, nous présentons les différentes méta-modèles utilisées ainsi que les méta-modèles élaborées pour mettre en place un générateur de code complet, permettant de produire des applications mobiles développées selon une approche native,

disposant de toutes les fonctionnalités offertes par les Smartphones.

6.3.3. Transformation et génération des classes métiers, classes accès aux données, la base de données et les interfaces classiques de mise à jours

6.3.3.1. Méta-modèle source

Le diagramme de classe présente un moyen permettant de modéliser une application en point vue métier. Ce diagramme décrit les relations entre les différents objets qui interagissent entre eux pour édifier un système d'information particulière. Ainsi, nous nous sommes basés sur le méta-modèle UML comme source pour avoir par la suite un modèle indépendant de la plateforme permettant de présenter les différentes classes métiers (*Beans*) d'une application mobile, et cela via des transformations modèle à modèle. Une fois que le nouveau modèle est créé des transformations modèle à texte sont appliquées afin de générer les classes métiers Java pour Android, C# pour Windows Phone et Swift pour iOS. Un extrait de méta modèle UML utilisé est présenté dans la figure ci-dessous.

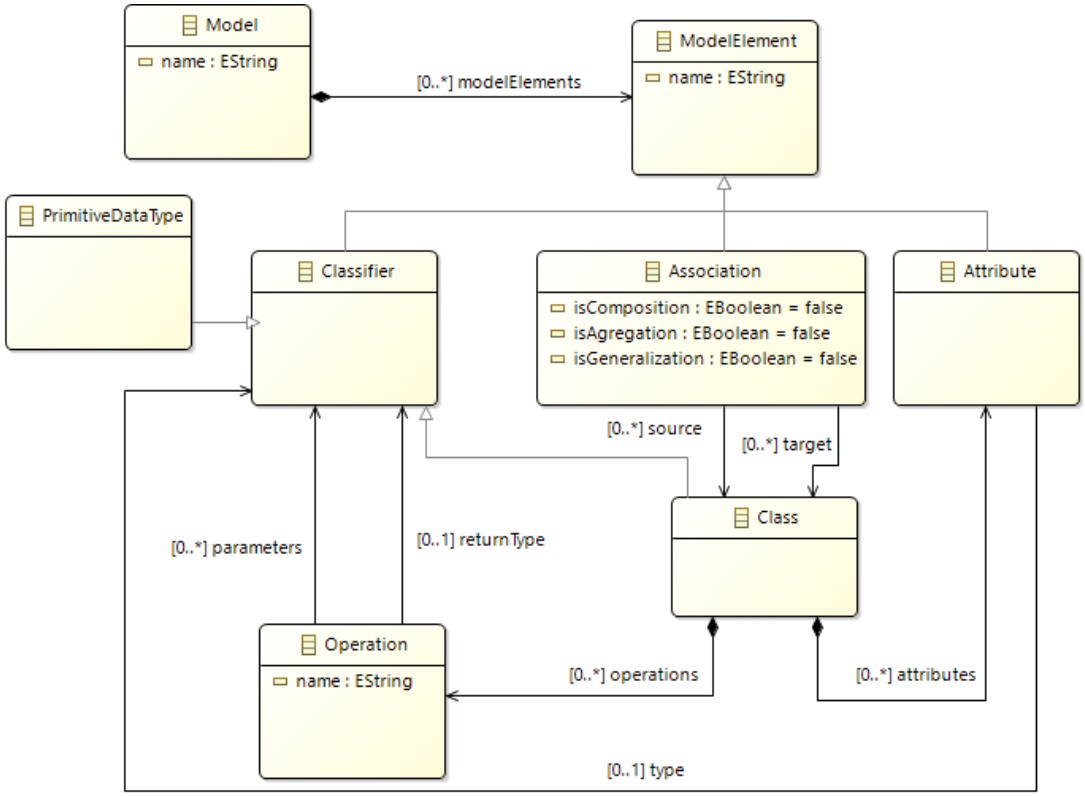


Figure 6. 6 - Extrait du Méta-modèle UML d'un diagramme de classe

6.3.3.2. Méta-Modèles cibles pour la génération de la couche métier

Pour la couche métiers nous sommes basées sur le méta-modèle présentée ci-dessous

comme cibles. Et à l'aide du langage ATL nous avons effectué les différentes transformations model-to-model à partir de méta-modèle UML vers notre méta-modèle PIM Bean. Ensuite, des transformations Modèle à Texte sont implémentées pour générer le code natif pour la couche métier et la couche présentation (*interfaces graphiques de mise à jour*).

Le méta-modèle cible est représenté dans la figure ci-dessous:

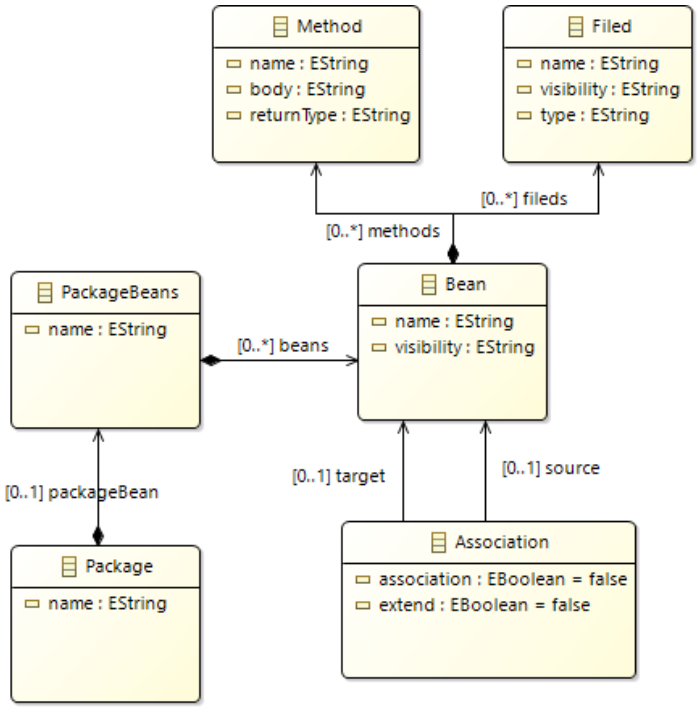


Figure 6. 7 - Meta-Modèle cible pour la génération des classes métiers (PIM Bean)

a. Les règles de transformation modèle à modèle sont présentées ci-dessous :

- Pour chaque instance de modèle UML, une instance PIM Package doit être créée.
  - Leurs noms doivent correspondre. Le nom du package contient les informations de chemin complet. La séparation de chemin est un point (.).
- Pour chaque instance de classe UML, une instance PIM Bean doit être créée.
  - Leurs noms doivent correspondre ;
  - La référence du paquet doit correspondre ;
  - Les modificateurs des Beans doivent être *public*.
- Pour chaque occurrence d'attribut UML, une instance PIM Field doit être créée.
  - Leurs noms doivent correspondre ;
  - Leurs types doivent correspondre ;
  - Les modificateurs doivent être *private* si la classe n'est pas une classe de base, et *protected* si la classe et une classe de base (*principe d'encapsulation*).

- Pour chaque instance UML Operation, une instance PIM Method doit être créée.
  - Leurs noms doivent correspondre ;
  - Leurs types doivent correspondre ;
  - Les Modificateurs doivent correspondre.

- Pour chaque instance UML Association, une instance PIM Association doit être créée.
  - Leurs noms doivent correspondre ;
  - Si l'association et une généralisation, on spécifié la valeur *true* pour *extend*. Si l'association est une agrégation ou une composition on spécifié *true* pour *association.*

b. Les règles de transformation modèle à texte sont présentées ci-dessous :

- Pour Android :
  - Pour chaque instance PIM PackageBean, une arborescence de dossiers sera générée dans le package principale, la séparation entre chaque dossier est identifié par le point « . » dans le nom de package.
  - Pour chaque instance PIM Bean, une classe JAVA doit être créée.
    - Leurs noms doivent correspondre ;
    - Les noms des packages doivent correspondre ;
    - Les modificateurs doivent correspondre ;
    - Les champs doivent correspondre ;
    - Pour chaque champ deux méthodes doivent être générées (Setters et les getters), avec un modificateur *public* ;
    - Cette classe doit contenir deux constructeurs, un pour initialiser tous les champs, et l'autre sans paramètres ;
    - Les méthodes doivent correspondre et générée dans la classe.
  - Pour chaque PIM Association :
    - Si la valeur de *extend = true* signifié que la classe source hérite (extends) de la classe cible. Ainsi, le constructeur des dérivés doit faire appel au constructeur de la classe de base ;
    - Si la valeur de *association = true* signifié que la classe cible est inclue dans la classe source (on déclare un objet de type classe cible dans la classe source et on applique la règles comme dans les cas de champs).

- Pour Windows Phone :
  - Pour chaque instance PIM PackageBean, une arborescence de dossiers sera générée dans le package principale, la séparation entre chaque dossier est identifié par le point « . » dans le nom de package.
  - Pour chaque instance PIM Bean, une classe C# doit être créée.
    - Leurs noms doivent correspondre ;
    - Les noms des packages et des namespaces doivent correspondre ;
    - Les modificateurs doivent correspondre ;
    - Les champs doivent correspondre ;
    - Pour chaque champ les getters et setters doivent être générées ;
    - Cette classe doit contenir deux constructeurs, un pour initialiser tous les champs, et l'autre sans paramètres ;
    - Les méthodes doivent correspondre et générée dans la classe.

  - Pour chaque PIM Association :
    - Si la valeur de *extend = true* signifié que la classe source hérite (:) de la classe cible. Ainsi, le constructeur des dérivés doit faire appel au constructeur de la classe de base ;
    - Si la valeur de *association = true* signifié que la classe cible est inclue dans la classe source (on déclare un objet de type classe cible dans la classe source et on applique la règles comme dans les cas de champs).

- Pour iOS :
  - Pour chaque instance PIM PackageBean, on associe un module Swift.
  - Pour chaque instance PIM Bean, une classe Swift doit être créée.
    - Leurs noms doivent correspondre ;
    - Les noms des modules prend le dernier mot après le point « . » dans le nom de PIM PackageBean ;
    - Les modificateurs doivent correspondre ;
    - Les champs doivent correspondre ;
    - Pour chaque champ les getters et setters doivent être générées ;
    - La classe doit contenir la méthode init() sans paramètre, et la méthode init(paramètres) pour initialiser les différentes champs ;
    - Les méthodes doivent correspondre et générée dans la classe.

- Pour chaque PIM Association :
  - Si la valeur de *extend = true* signifié que la classe source hérite (:) de la classe cible. Ainsi, la méthode init () de la classe dérivée doit faire appel à la méthode init() de la classe de base (super) ;
  - Si la valeur de *association = true* signifié que la classe cible est inclue dans la classe source (on déclare un objet de type classe cible dans la classe source et on applique la règles comme dans le cas de champs).

6.3.3.3. Méta-modèles cibles pour la génération de la couche de données et la couche accès aux données

Pour la génération de la base de données nous avons appliqué des transformations modèle à modèle à partir de notre méta-modèle PIM Bean vers le méta-modèle Relationnelle PIM DataBase ci-dessous.

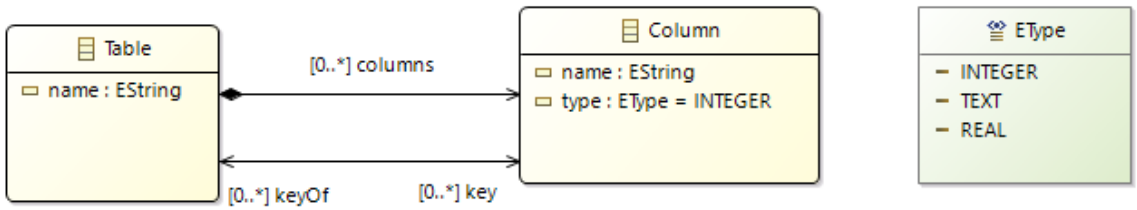


Figure 6. 8 - PIM DataBase

Les différentes règles de transformations appliquées sont décrites ci-dessous :

- Pour chaque instance PIM Bean, une table doit être créée.
  - Leurs noms doivent correspondre ;
  - La clé primaire de chaque table doit être de type INTEGER, auto-incrément nommée id « TABLE_NAME » ;
  - Les attributs de type primitif sont transformés en colonnes, leurs noms doivent correspondre, leurs types sera (INTEGER, TEXT ou REAL). Chaque type primitif doit être converti en un de ces trois types ;
  - Pour les types objets sont transformés en des clés étrangères, leurs types et le type de l'attribut correspondant converti à l'un des types : INTEGER, TEXT ou REAL.

Dans le cas d'une association d'héritage, la clé primaire de la classe fille ne doit pas être auto-incrément et jouera également le rôle d'une clé étrangère qui fait références vers la table correspondante à la classe mère.

a. Génération des classes et interfaces

Pour la génération de la couche accès aux données et la couche de données, nous sommes basés sur le PIM DataBase précédemment obtenu. Le modèle de Template cible proposé est présenté sur la figure ci-dessous. La classe MySQLiteHelper permet de définir le nom de la base de données, la version de la base de données et les requêtes de la création de la base de données. Ainsi que les requêtes de la suppression de la base de données en cas de mise à jour.

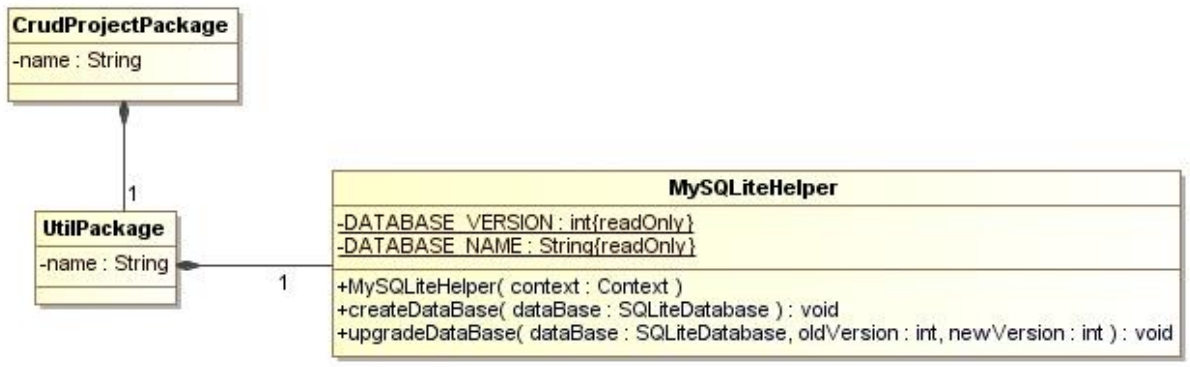


Figure 6. 9 - Méta-modèle création et mise de la base de données

Les règles de projections sont définies comme suit :

- Dans la classe MySQLiteHelper :
  - On définit les constantes suivantes :
    - Le nom de la base de données identique au nom du projet ;
    - La version de la base de données identique à la version de l'application.
    - Une requête de la création de chaque table nommée « CREATE_TABLE_ « table.name » ;
  - Dans la méthode createDataBase (), on exécute les requêtes de la création préalablement définies ;
  - Dans la méthode upgradeDataBase(), on supprime la base de données si la nouvelle version est augmenté par rapport à l'ancienne version, ensuite on fait appelle à la méthode createDataBase() pour récréer la base de données.

Pour la génération des classes dao, nous proposons le méta-modèle cible qui est basé sur le patron de conception « data access object pattern » présenté dans la figure ci-dessous.

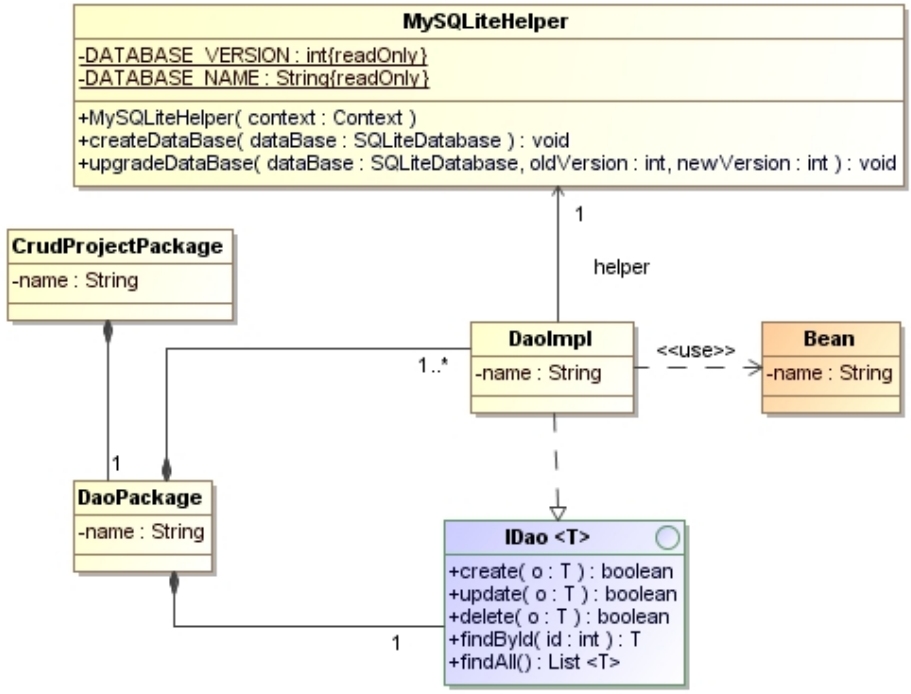


Figure 6. 10 - Méta-modèle création des classes services (DAO)

Description:

- L'interface générique (IDao) définit les opérations standards à effectuer sur un objet modèle.
- La classe concrète (DaoImp) qui implémente l'interface IDao est responsable d'obtenir des données à partir d'une source de données qui peut être une base de données / un fichier xml ou tout autre mécanisme de stockage.

Les règles de projections sont définies comme suit :

- Pour chaque instance PIM Bean, une classe dao sera créée.
  - Le nom de la classe dao est généré comme suit : «Bean.name»Service.
  - Cette classe contient la déclaration des constantes :
    - Le nom de la table associée ;
    - Les champs de la table ;
    - Un tableau qui stock les champs de la table ;
    - Les définitions des méthodes (CRUD) générées en utilisant des Template spécifiques pour chaque plateforme.

- Echange de données entre un serveur SQL et une application mobile.
  - Principe générale :
    - L'application construit des requêtes http (de type GET ou POST)

URL = http://*serveur*/*script*?*paramètres*

paramètres = conditions du select, p. ex. identifiant=1.

- L'application cliente (Mobile) envoie cette requête au serveur SQL puis attend la réponse ;
- Le script serveur exécute la requête puis retourne le résultat encodé en JSON à l'application ;
- L'application mobile décode le résultat et l'affiche ;
- Chaque méthode CRUD sera associée à un script spécifique coté serveur ;
- L'application coté serveur est généré à partir de modèle PIM Bean, cette application respecte une architecture en couche (en PHP 5), l'échange de données se fait avec JSON.

- Génération des interfaces graphiques CRUD associées aux différents Beans :
  - Pour chaque instance PIM Bean, trois interfaces graphiques utilisateurs sont générées :
    - *Une interface d'ajout* : cette interfaces fait appel à la méthode create () de la couche accès aux données. Après l'ajout l'utilisateur sera redirigé vers une deuxième interface pour afficher la liste des données ;
    - *Une interfaces pour lister les données* : cette interfaces fait appel à la méthode findAll() de la couche accès aux données. Au clic sur un élément de la liste, l'utilisateur aura le choix entre la suppression de l'objet en question en appelant les deux méthodes findById() et delete() ou la redirection vers une 3ème interface de modification en transmettant l'identifiant de l'objet en question ;
    - *Une interface de modification* : cette interface permet de modifier l'objet résultat de la sélection à partir de la liste en appelant les méthodes findById() et update() . Après la modification l'utilisateur sera redirigé vers la liste des données ;
    - Les navigations entre les différentes interfaces sont générer dans un fichier menu_main redirigeant vers les différents écrans liste associées à chaque bean.

- Quelques correspondances entre les différents langages cibles:
  - Types et déclarations des variables

	Variables		
	Swift	C#	Java
Boolean	Bool	Bool	Boolean
Constant	Let	Const	Final
Declaration	Var	Var	(no equivalent)
Float	Float, Double	float, double	float, double
Integer	Int	Int	Int
String	String (value)	String (reference)	String (reference)

- Conditions, boucles et fonctions

	Boucles, Conditions et fonctions		
	Swift	**C#**	**Java**

Iterating Over Array	for item in arr {
	// do something
	}
		foreach (var item in arr) {
		// do something
		}
			for (type item : arr){
			// do something
			}

Is Array Empty?	if arr.isEmpty {
	// array is empty
	}
		if (arr.Length == 0) {
		// array is empty
		}
			if (arr.length == 0) {
			// array is empty
			}

For Loops	for var i = 0; i <= 5; ++i {
	// do something with i
	}
		for(var i = 1; i <= 5; i++) {
		// do something with i
		}
			for(int i = 1; i <= 5; i++) {
			// do something with i
			}

Conditional Statements	if i > 6 {
	// do something
	} else if i > 3 && i <= 6 {
	// do something
	} else {
	// do something
	}
		if (i > 6) {
		// do something
		} else if (i > 3 && i <= 6) {
		// do something
		} else {
		// do something
		}
			if (i > 6) {
			// do something
			} else if (i > 3 && i <= 6) {
			// do something
			} else {
			// do something
			}

Switch statement	var word = "A"
	switch word {
	case "A":
	// do something
	case "B":
	// do something
	default:
	// do something
	}
		var word = "A";
		switch(word) {
		case "A":
		// do something
		break;
		case "B":
		// do something
		break;
		default:
		// do something
		break;
		}
			String word = "A";
			switch(word) {
			case "A":
			// do something
			break;
			case "B":
			// do something
			break;
			default:
			// do something
			break;
			}

Functions	func sayHello(name: String) -> String
	{
	// do something
	}
		string sayHello(string name) {
		// do something
		}
			String sayHello(string name) {
			// do something
			}

- Protocoles

	Protocols		
	Swift	C#	Java
Protocol	Protocol	Interface	Interface
Implements	:	:	Implements

- Classes et la généricité

	Classes		
	Swift	C#	Java
Constructor	Init	Constructor	Constroctor
Class	Class	Class	Class
Inheritance	:	:	Extends
Access	private, public	private, public, protected, internal	private, public, protected, defalut
Self	Self	This	This
Object	AnyObject, Any	Object	Object
Parent	:	:	Super
generics type	generic types	generic types	generic types
generics function	generic functions	generic functions	generic functions

6.3.4. Transformation et génération des interfaces graphiques personnalisée, les classes de traitements et les classes accès aux services

6.3.4.1. Méta-modèle source

Le méta-modèle publié dans (***Lachgar et al, 2016***) permet de créer des modèles de base permettant la génération :

- Des interfaces graphiques utilisateurs,
- Des fichiers de configuration contenant :
  - Les informations sur le projet (domaine, icône, version, auteur, etc.) ;
  - La déclaration des activités ;
  - La spécification des permissions ;
  - La spécification des capteurs embarqués ;
  - etc.
- Des navigations entres les différentes écrans ;
- Des menus de navigation ;
- Des classes de traitements de données avec les évènements sur les composants graphiques ;
- Des accès aux différentes API natives ;
- Des accès aux capteurs embarqués.

Dans ce chapitre, nous contribution à une extension de notre méta-modèle en rajoutant la possibilité de modéliser les écrans de base (e.g. LoginScreen, MapScreen, MediaScreen, etc.), de généraliser les styles appliqués sur les écrans, d'offrir un mécanisme permettant de

modéliser l'accès aux services web préalablement définies. Les classes rajoutées sont marquées en vert. Le méta-modèle source est présenté dans la figure ci-dessous :

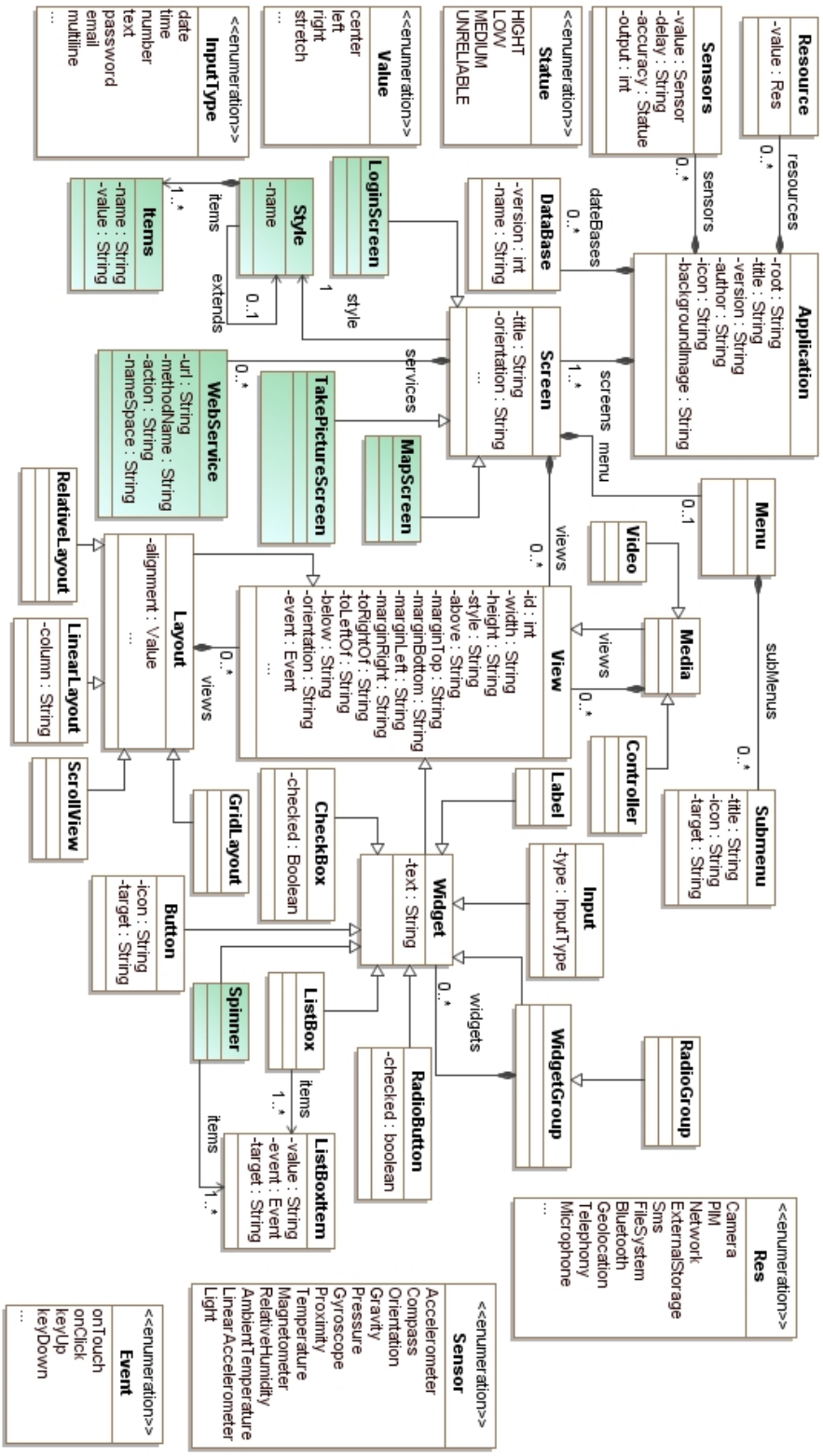


Figure 6. 11 - Extension de Méta-modèle Mobile DSL

6.4. Etude de cas

Pour démontrer l'efficacité de l'approche proposée, une étude de cas a été réalisée dans

laquelle seules les classes métier d'une application ont été modélisées avec un diagramme de classe, ensuite une application Android complète a été générée à la fin.

L'application sélectionnée une application simple pour la gestion des produits. Le digramme de classe ci-dessous décrit les classes métiers de cette application.

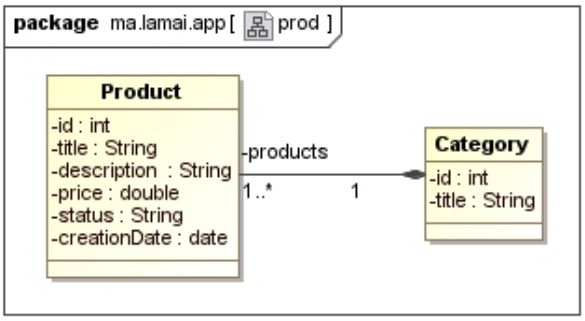


Figure 6. 12 – Diagramme de classe simplifié « Gestion des produits »

L'architecture de l'application générée dans un projet sous *Android Studio* :

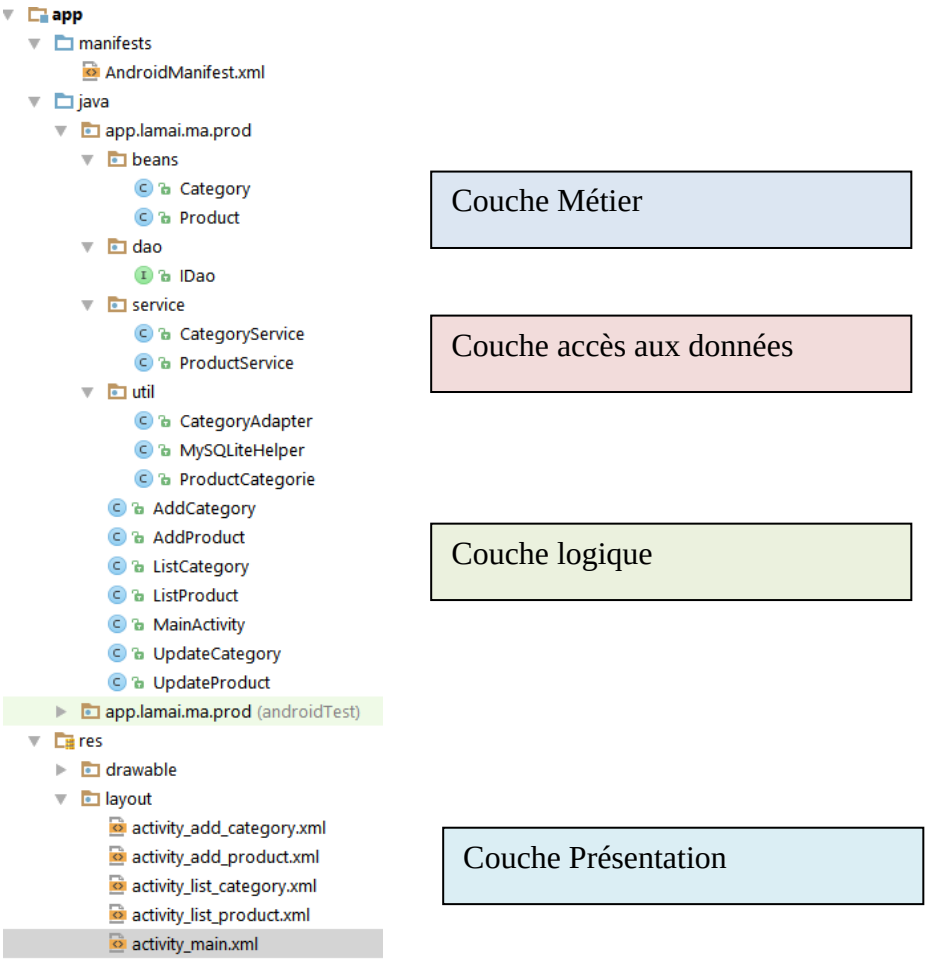


Figure 6. 13 - Architecture de l'application Android générée sous Android Studio

Le diagramme de navigation suivant décrit les différentes transitions entre les différents écrans de cette application :

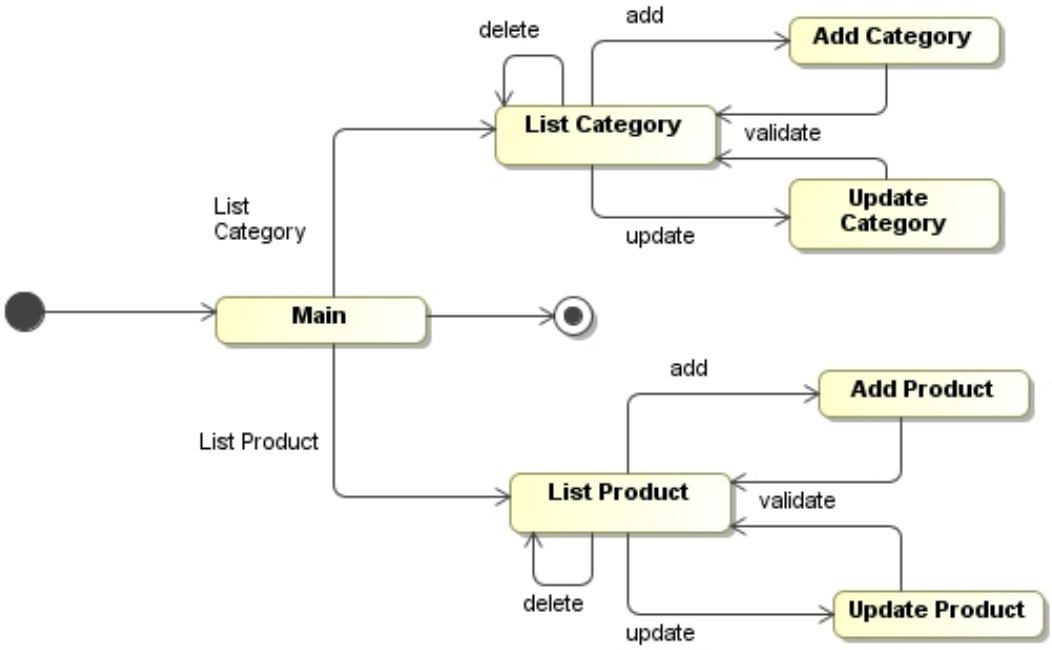
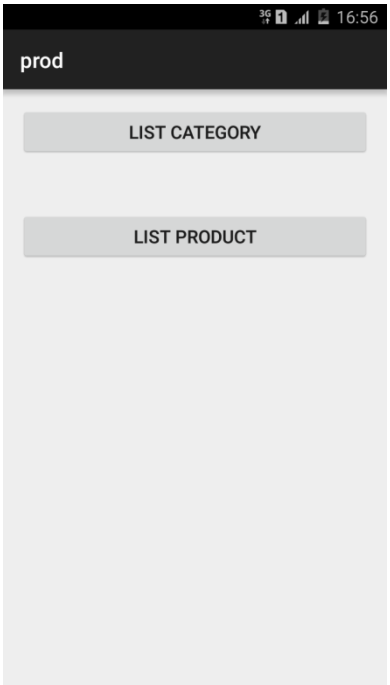


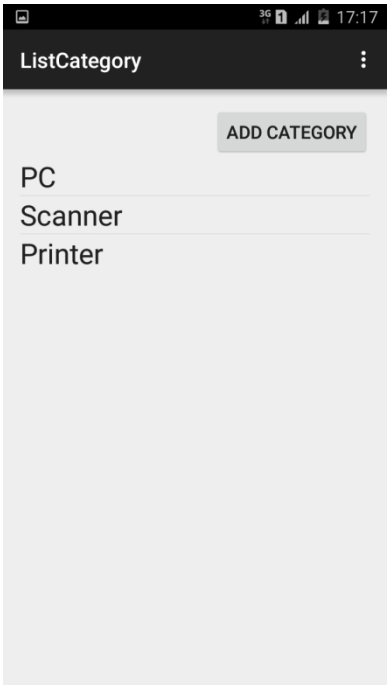
Figure 6. 14 - Diagramme de navigation entre les écrans de l'application « Gestion des produits »

Un menu de navigation est générer permettant de se déplacer entre les activités d'une manière plus fluide.

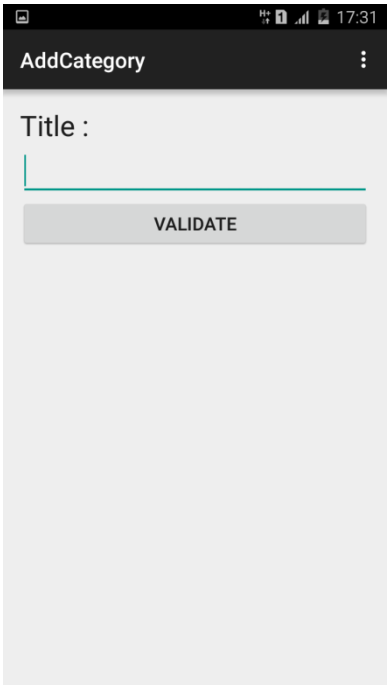
Les interfaces graphiques générées ciblant la plateforme Android:



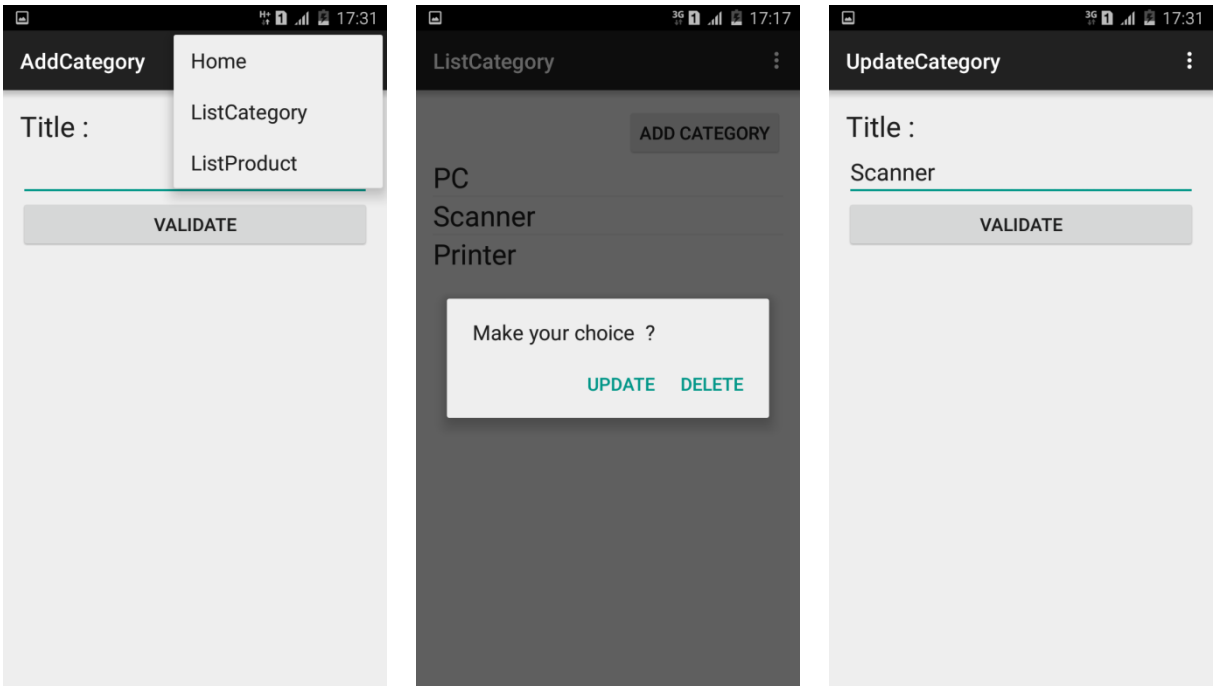
L'écran principale



Liste des catégories



Ajouter une catégorie



Menu de navigation *Modification et suppression* *Modification d'une catégorie*

Figure 6. 15 - Quelques écrans de l'application « Gestion des produits »

Dans cette étude de cas nous avons généré les méthodes et les interfaces CRUD d'une application simple de gestion des produits, les données manipulées sont stockées dans une base de données embarquées SQLite.

Le tableau ci-dessous présente une comparaison entre notre approche et l'approche tradionnelle. Les résultats présentés ci-dessous sont recueillis durant l'examen de fin de formation de la filière programmation web mobile à l'Institut Spécialisé dans les Technologies de l'Information et les Métiers de l'Offshoring de Marrakech.

Tableau 6. 1 - Comparsaion entre notre approche et l'approche tradionnelle

	Selon notre approche	Avec la méthode tradionnelle
Nombre de fichiers dans le projet	23 fichiers	23 fichiers
Durée	10 min	300 min
Nombre de fichiers édités par le développeur	0	21 Fichiers

6.5. Conclusion

Dans ce chapitre nous avons présenté notre approche pour la génération des applications mobiles multiplateformes en respectant une architecture multicouches. Pour se faire, nous avons combiné entre la modélisation avec le langage UML et le langage dédié ***Mobile DSL***. Cette nouvelle approche permet de générer les classes métiers, les classes accès aux données, les classes accès aux services, les fichiers de configurations, les services web pour les

fonctions standard CRUD etc. Egalement, nous avons présenté une étude de cas afin de valider cette approche, et nous avons généré les fonctionnalités CRUD pour une application Android simple. Pour les travaux futurs, nous allons travailler sur l'implémentation de notre générateur de code afin de générer des applications pour les autres plateformes mobiles (e.g. iOS, Windows phone, etc.).

Conclusion et Perspectives

L'hétérogénéité des systèmes d'exploitation mobile est devenue un vrai problème, devant les entreprises et les développeurs indépendants qui se spécialisent dans l'édition de logiciels pour smartphone. En effet, le développement d'une même application pour ces différentes plateformes mobiles, devient une tâche épuisante.

Dans ces travaux de thèse, nous nous sommes intéressés à cette problématique. Pour y répondre, nous avons proposé d'utiliser et d'adapter l'approche MDA pour le développement d'applications mobiles multiplateformes. En effet, l'approche MDA apporte des avancées significatives dans la maîtrise du développement des applications informatiques et permet notamment des gains de productivité, une fiabilité accrue, une amélioration significative de la pérennité et une meilleure agilité face aux changements, ainsi, permet d'augmenter la qualité des logiciels produits et facilite la maintenance. Par conséquence, l'approche MDA apporte une solution pratique facile et efficace à une telle problématique, en développant une application multiplateforme.

Ce travail de recherche a permis de contribuer dans le domaine de la modélisation et la génération de code natif des applications mobiles multiplateformes en répondant aux objectifs suivants :

(i) Propositions d'un cadre décisionnel pour les méthodes de développement mobile ;
(ii) Proposition d'une démarche pour la modélisation et génération automatiques des interfaces graphiques des applications web et mobile basées sur les composants ;
(iii) Proposition d'une démarche pour la modélisation et la génération de code natif pour les applications mobiles multiplateforme ;
(iv) Proposition d'une démarche pour la modélisation et la génération de code natif pour les applications mobiles multiplateformes en respectant une architecture en couche.

Le premier défi (i) est abordé dans le chapitre 1 en proposant un cadre décisionnel pour décider dans un premier lieu sur la méthode à utiliser pour développer une application spécifique, en se basant sur un arbre binaire de décision, dans un deuxième lieu sur l'outil à adopter pour la mise en place de l'application, en reposant sur un ensemble de critères spécifiques aux plateformes mobiles et sur des critères qualitatifs. Le second défi (ii) a été abordé dans le chapitre 4. Cette phase s'appuie sur un *DSL GUI* qui offre la possibilité de décrire le corps d'une page web ou d'un écran mobile, ensuite, des transformations modèle à

modèle vers la plateforme cible sont effectuées, avant de projeter dans des modèles des plateformes visées. Le troisième défi (iii), décrit dans le chapitre 5, a consisté à développer un *DSL Mobile* permettant de modéliser les applications mobiles en tenant compte des différentes caractéristiques de ces derniers (e.g. API natives, GPS, Capteurs embarqués, etc.), ensuite, un ensemble de transformations vers les plateformes (*Android, iOS et Windows phone*) leader de marché ont été proposées, afin de projeter sur des modèles (*Template*) proposés pour chaque plateforme cible. Le quatrième défi (iv) a été décrit dans le chapitre 6. Il a consisté à traiter une démarche permettant de modéliser et de générer des applications mobiles multiplateformes qui respectent une architecture en couche. L'approche proposée combine entre une modélisation avec le langage UML et le langage dédiés précédemment proposé dans le chapitre 5 que nous avons également amélioré.

Notre approche ne prétend pas répondre de façon parfaite et complétement opérationnelle à la problématique traitée. Certaines limitations existent ; Le générateur de code suggéré est basé sur un nouveau langage dédié, pour le développement des applications mobiles multiplateformes. En conséquence, le développeur devrait apprendre un nouveau langage pour utiliser la solution, afin d'assurer un bon démarrage du projet. D'autre part, ce générateur de code ne permet pas de convertir les applications existantes (*rétro-ingénierie*) Android vers iOS, ou Windows phone, vice versa, sans avoir à réécrire le modèle conforme avec notre DSL en premier lieu. En outre, le générateur de code proposé se concentre principalement sur le développement de nouvelles applications mobiles depuis le début sans bénéficier de code des applications mobiles existantes.

Le générateur de code présenté dans le chapitre 6, permet pour l'instant de générer les fonctionnalités CRUD pour une application Android, en outre, ne permet pas de faire une liaison entre le code généré à partir de DSL et le code généré à partir de diagramme de classe, ce qui rend la génération de code semi-automatique.

Pour les améliorations à venir, nous allons compléter le générateur de code afin d'avoir une liaison entre un modèle basé sur le *DSL Mobile* et le code généré à partir d'un diagramme de classe, et de supporter les différentes plateformes leader du marché de la mobilité. Egalement, nous allons étudier la possibilité d'étendre notre DSL pour prendre en compte la modélisation des traitements métiers, ensuite, la génération de code ciblant les différentes plateformes mobiles.

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 Conceptual Modeling (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).

Armstrong, D. J. (2006). The quarks of object-oriented development. Communications of the ACM, 49(2), 123-128.

Android (2017). Platform Architecture. Retrieved from https://developer.android.com/guide/platform/index.html#api-framework.

AndroMDA (2014). Generate components quickly with AndroMDA. Retrieved from http://andromda.sourceforge.net.

ATLAS group LINA & INRIA Nantes (2006). ATL User Manual - version 0.7. Retrieved from http://www.eclipse.org/m2m/atl/doc/ATL_User_Manual[v0.7].pdf

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.

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).

Benouda, H., Azizi, M., Esbai, R., & Moussaoui, M. (2016). Code generation approach for mobile application using acceleo. International Review on Computers and Software (IRECOS), 11(2), 160-166.

Benouda, H., Azizi, M., Esbai, R., & Moussaoui, M. (2016). MDA approach to automate code generation for mobile applications. In Mobile and Wireless Technologies 2016 (pp. 241-250). Springer Singapore.

Bergsten, H. (2004). JavaServer faces. O'Reilly Media, Inc.

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.

Burns, E., Schalk, C., & Griffin, N. (2010). JavaServer Faces 2.0. McGraw-Hill.

Cabot, J. (2015). Clarifying concepts: MBE vs MDE vs MDD vs MDA. Retrieved from http://modelinglanguages. com/clarifying-concepts-mbe-vs-mde-vs-mdd-vs-mda/.

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

Corral, L., Janes, A., & Remencius, T. (2012). Potential Advantages and Disadvantages of Multiplatfonn Development Frameworks-A Vision on Mobile Environments. Procedia Computer Science, vol. 1 0, pp. 1202- 1207.

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

Dalmasso, I., Datta, S., Bonnet, C., & Nikaein, N. (2013). Survey, comparison and evaluation of cross platform mobile application development tools. Proceedings of the 9th International Wireless Communications and Mobile Computing Conference (IWCMC), IEEE Xplore press, Sardinia, pp. 323-328. DOI: 10.1109/IWCMC.2013.6583580.

Diep, C. K., Tran, Q. N., & Tran, M. T. (2013, December). Online model-driven IDE to design GUIs for cross-platform mobile applications. In Proceedings of the Fourth Symposium on Information and Communication Technology (pp. 294-300). ACM.

Eclipse Foundation (2011). Eclipse4/RCP/Architectural Overview. Retrieved from https://wiki.eclipse.org/Eclipse4/RCP/Architectural_Overview.

Eclipse Foundation (2013a). Xtext. Retrieved from http://www.eclipse.org/Xtext/.

Eclipse Foundation (2013b). Xpand. Retrieved from https://projects.eclipse.org/projects/modeling.m2t.xpand.

Eclipse Foundation (2014). Xtend. Retrieved from http://www.eclipse.org/xtend/.

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

El Hamlaoui, M. (2015). Mise en correspondance et gestion de la cohérence de modèles hétérogènes évolutifs (Doctoral dissertation, Université Toulouse le Mirail-Toulouse II).

El-Kassas, W. S., Abdullah, B. A., Yousef, A. H., & Wahba, A. (2014, December). ICPMD: integrated cross-platform mobile development solution. In the 9th International Conference on Computer Engineering & Systems (ICCES), (pp. 307-317). IEEE.

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.

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.

Farhana, I., Riyadh, M. (2013). Comparison Between Two Model Transformation Frameworks: Kermeta and ATL. University of Ottawa.

Favre, J. M., Estublier, J., & Blay, M. (2006). L'ingénierie dirigée par les modèles: au-delà du MDA. chapter 3, pages 53–69. Hermes-Lavoisier.

Fleurey, F., Drey, Z., Vojtisek, D., Faucher, C., & Mahé, V. (2006). Kermeta language, reference manual. Retrieved from http://www. kermeta. org/docs/KerMeta-Manual. pdf. IRISA.

Florent, D. & Lamia, G. (2013). 5 bonnes raisons du MDA pour le développement Mobile. Retrieved from https://blog.netapsys.fr/5-bonnes-raisons-du-mda-pour-le-developpement-mobile/.

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.

Hailpern, B., & Tarr, P. (2006). Model-driven development: The good, the bad, and the ugly. IBM systems journal, 45(3), 451-461.

Heitkötter, H., Majchrzak, T. A., & Kuchen, H. (2013, March). Cross-platform model-driven development of mobile applications with md 2. In Proceedings of the 28th Annual ACM Symposium on Applied Computing, 526-533. ACM.

Heitkötter, H., Hanschke, S., & Majchrzak, T. A. (2012). Evaluating cross-platform development approaches for mobile applications. Web information systems and technologies. Springer Berlin Heidelberg, 120-138.

Herndon, R. M., & Berzins, V. A. (1988). The realizable benefits of a language prototyping language. IEEE Transactions on Software Engineering, 14(6), 803-809.

Höbler, J., Soden, M. and Eichler, H. (2006). Coevolution of Models, Metamodels and Transformations. Retrieved from http://www.ikv.de/index.php,2006.

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.

Jézéquel, J. M., Combemale, B., & Vojtisek, D. (2012). Ingénierie Dirigée par les Modèles: des concepts à la pratique.. (p. 144). Ellipses.

Jiang, S., & Mu, H. (2011, July). Design patterns in object oriented analysis and design. In Software Engineering and Service Science (ICSESS), 2011 IEEE 2nd International Conference on (pp. 326-329). IEEE.

Jonkers, H., Steen, M., Heerink, L., & Van Leeuwen, D. (2007). From Business Process Design to Code: Model-Driven Development of Enterprise Applications.

Jouault, F., Bézivin, J. (June, 2006). KM3 : a DSL for Metamodel Specification FMOODS 2006, Bologna, Italy, 14-16.

Jouault, F., Allilaire, F., Bézivin, J., & Kurtev, I. (2008). ATL: A model transformation tool. Science of computer programming, 72(1), 31-39.

Juliano, D., Ferreira, E. O., Campos, H. F., da Cunha, A. M., & Dias, L. A. V. (2011, April). Applying MDA approach to create graphical user interfaces. In Information Technology: New Generations (ITNG), 2011 Eighth International Conference on (pp. 766-771). IEEE.

Kahlaoui, A., & Abran, A. (2014). Demystifying domain specific languages. Computational Linguistics: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications, 228.

Kent, S. (2002). Model driven engineering. Integrated Formal Methods (IFM), Turku (Finland) Springer, 286-298.

Kerensen Consulting (2015). Evolution des usages Mobiles, prévision 2015.

Klatt, B. (2007). Xpand: A closer look at the model2text transformation language. Language, 10(16), 2008.

Kleppe, A. G., Warmer, J. B., & Bast, W. (2003). MDA explained: the model driven architecture: practice and promise. Addison-Wesley Professional.

Koji, M., & Saeko, M. (2014, October). MDD for Smartphone Application with Smartphone Feature Specific Model and GUI Builder. In: The 9th International Conference on Software Engineering Advances. p. 64-68. ISBN: 978-1-61208-367-4.

Lachgar, M., & Abdali, A. (2014a, October). Generating Android graphical user interfaces using an MDA approach. In 2014 Third IEEE International Colloquium in Information Science and Technology (CIST) (pp. 80-85). IEEE.

Lachgar, M., & Abdali, A. (2014b, Juin). Modélisation et Génération des Interfaces Graphiques Android et JSF 2.2 selon une approche MDA. La 6 ème édition des Journées Doctorales en Technologies de l'information et de la Communication. Les 19 et 20 Juin 2014 ENSIAS, Rabat – Maroc.

Lachgar, M., & Abdali, A. (2015). Modeling and Generating the User Interface of Mobile Devices and Web Development with DSL. Journal of Theoretical & Applied Information Technology, 72(1).

Lachgar, M., & Abdali, A. (2016). Modeling and generating native code for cross-platform mobile applications using DSL. Intelligent Automation & Soft Computing, 1-14.

Lachgar, M., & Abdali, A. (2017). Decision Framework for Mobile Development Methods. International Journal of Advanced Computer Science and Applications (IJACSA), 8(2). http://dx.doi.org/10.14569/IJACSA.2017.080215.

Laine (2013). Les DSL appliqués à MDA, Retrieved from http://laine.developpez.com/tutoriels/alm/mda/dsl-appliques-mda/. le magazine des développeurs.

Lettner, M., Tschernuth, M., & Mayrhofer, R. (2011, February). Mobile platform architecture review: android, iPhone, qt. In International Conference on Computer Aided Systems Theory (pp. 544-551). Springer Berlin Heidelberg.

Link, S., Schuster, T., Hoyer, P., & Abeck, S. (2008, February). Focusing graphical user interfaces in model-driven software development. In 1st International Conference on Advances in Computer-Human Interaction, (pp. 3-8). IEEE.

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.

Madari, I., Lengyel, L., & Levendovszky, T. (2007, November). Modeling the User Interface of Mobile Devices with DSLs. In 8th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics, Budapest, Hungary (pp. 583-589).

Mannadiar, R., & Vangheluwe, H. (2010, October). Domain-specific engineering of domain-specific languages. In Proceedings of the 10th Workshop on Domain-Specific Modeling (p. 11). ACM.

Mellor, S. J., Clark, T., & Futagami, T. (2003). Model-driven development: guest editors' introduction. IEEE software, 20(5), 14-18.

Mens, T., & Van Gorp, P. (2006). A taxonomy of model transformation. Electronic Notes in Theoretical Computer Science, 152, 125-142.

Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. ACM computing surveys (CSUR), 37(4), 316-344.

Minhyuk, K., Yong-Jin, S., Bup-Ki, M., Seunghak, K., & Hyeon, S. K. (2012, May). Extending UML meta-model for android application. In Computer and Information Science (ICIS), IEEE/ACIS 11th International Conference on (pp. 669-674). IEEE.

Minsky, M. (1965). Matter, mind and models. International Federation for Information Processing (IFIP), 1, 45-50.

Mohagheghi, P., & Dehlen, V. (2008, June). Where is the proof?-a review of experiences from applying MDE in industry. In European Conference on Model Driven Architecture-Foundations and Applications (pp. 432-443). Springer Berlin Heidelberg.

Mtsweni, J. (2012, December). Exploiting UML and acceleo for developing semantic web services. In Internet Technology and Secured Transactions, 2012 International Conference for (pp. 753-758). IEEE.

Musset, J., Juliot, É., Lacrampe, S., Piers, W., Brun, C., Goubet, L., Lussaud, Y. & Allilaire, F. (2006). Acceleo user guide. Retrieved from http://www.acceleo.org/doc/obeo/en/acceleo-2.6-user-guide.pdf, 2006.

Object Management Group (1989). Retrieved from http://www.omg.org.

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.

OMG (2006). Object Constraint Language (OCL) Specification, version 2.0. Retrieved from http://www.omg.org/spec/OCL/2.0/.

OMG (2011a). Abstract Syntax Tree Metmodel (ASTM). Retrieved from http://www.omg.org/spec/ASTM/1.0.

OMG (2011b). Knowledge Discovery Metamodel (KDM). Retrieved from http://www.omg.org/spec/KDM/1.3.

OMG (2014). Metadata Interchange (XMI), version 2.4.2. Retrieved from http://www.istr.unican.es/pyemofuc/PyEmofUCFiles/XMI_formal-14-04-04.pdf.

OMG (2015). Meta Object Facility (MOF), Query/View/Transformation (QVT), version 2.0. Retrieved from http://www.omg.org/spec/QVT/1.2.

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.

Palyart, M., Lugato, D., Ober, I., & Bruel, J. (2011, October). Improving scalability and maintenance of software for high-performance scientific computing by combining MDE and frameworks. Model Driven Engineering Languages and Systems. (pp. 213-227). Springer Berlin Heidelberg.

Parada, A. G., & de Brisolara, L. B. (2012, November). A model driven approach for android applications development. In Brazilian Symposium on Computing System Engineering (SBESC), (pp. 192-197). IEEE.

Parada, A. G., Siegert, E., & de Brisolara, L. B. (2011). GenCode: A tool for generation of Java code from UML class models. In Proc. 26th South Symposium on Microelectronics (SIM 2011) (pp. 173-176).

Perchat, J., Desertot, M., & Lecomte, S. (2013). Component based framework to create mobile cross-platform applications. Procedia Computer Science, 19, (pp. 1004-1011).

Pérez-Martínez, J. E., & Sierra-Alonso, A. (2006, July). From analysis model to software architecture: A PIM2PIM mapping. In European Conference on Model Driven Architecture-Foundations and Applications (pp. 25-39). Springer Berlin Heidelberg.

Plakalovic, D., & Simic, D. (2010). Applying MVC and PAC patterns in mobile applications. arXiv preprint arXiv:1001.3489.

Presso, M. J., & Belaunde, M. (2005, November). Applying MDA to voice applications: an experience in building an MDA tool chain. In European Conference on Model Driven Architecture-Foundations and Applications (pp. 1-8). Springer Berlin Heidelberg.

Quinton, C., Demarey, C., Dolet, N., & Duchien, L. (2011). AppliDE: modélisation et génération d'applications pour smartphones. In Journées sur l'Ingénierie Dirigée par les Modèles (IDM'11) (pp. 41-45).

Raghu, R., & Shobha, K. R. (2012). JavaScript application framework for mobile devices. In Global trends in computing and communication systems (pp. 291-299). Springer Berlin Heidelberg.

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.

Rongge, M. D. L. (2002). A Three-tier Design Model for ASP. net-based Web Applications [J]. Microcomputth Applications, 3, 120-124.

Sabraoui, A., El Koutbi, M., & Khriss, I. (2012, November). GUI code generation for Android applications using a MDA approach. In International Conference on Complex Systems (ICCS), (pp. 1-6). IEEE.

Sabraoui, A., El Koutbi, M., Khriss, I. (2013). A MDA-Based Model-Driven Approach to Generate GUI for Mobile Applications. International Review on Computers and Software Journal (IRECOS), 2013, vol.

8, no 3 (pp. 845-852).

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.

Sarcar, V. (2016). Abstract Factory Patterns. In Java Design Patterns (pp. 109-114). Apress.

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.

Selvarajah, K., Craven, M. P., Massey, A., Crowe, J., Vedhara, K., & Raine-Fenning, N. (2013, July). Native Apps versus Web Apps: which is best for healthcare applications?. In International Conference on Human-Computer Interaction (pp. 189-196). Springer Berlin Heidelberg.

Serrano, N., Hernantes, J., & Gallardo, G. (2013). Mobile Web Apps. IEEE Software. pp: 22 – 27, DOI:10.1109/MS.2013.111.

Sewall, S. J. (2003). Executive Justification for Adopting Model Driven Architecture (MDA). Object Management Group, Tech. Rep.

Smutný, P. (2012, May). Mobile development tools and cross-platform solutions. In Carpathian Control Conference (ICCC), 2012 13th International (pp. 653-656). IEEE.

Snyder, A. (1986, June). Encapsulation and inheritance in object-oriented programming languages. In ACM Sigplan Notices (Vol. 21, No. 11, pp. 38-45). ACM.

Staff (2008, November). Guide méthodologique pour les transformations de modèles. Rapport de recherche n° 8, IRIT/MACAO.

Stencel, K., & Węgrzynowicz, P. (2008, November). Implementation variants of the singleton design pattern. In OTM Confederated International Conferences" On the Move to Meaningful Internet Systems" (pp. 396-406). Springer Berlin Heidelberg.

Sun, M., & Tan, G. (2014, July). NativeGuard: Protecting android applications from third-party native libraries. In Proceedings of the ACM conference on Security and privacy in wireless & mobile networks (pp. 165-176). ACM.

Tian, L., Du, H., Tang, L., et al. (2013). The discussion of cross-platform mobile application based on Phonegap. In 4th IEEE International Conference on Software Engineering and Service Science (ICSESS) (pp. 652-655).

Tracy, K. W. (2012). Mobile application development experiences on Apple's iOS and Android OS. Ieee Potentials, 31(4), 30-34.

Trask, B., Paniscotti, D., Roman, A., & Bhanot, V. (2006).Using Model-Driven Engineering to Complement Software Product Line Engineering in Developing Software Defined Radio Components and Applications. In ACM SIGPLAN International Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA), (pp. 846– 853). ACM.

Tschernuth, M., Lettner, M., & Mayrhofer, R. (2011, February). Evaluation of descriptive user interface methodologies for mobile devices. In International Conference on Computer Aided Systems Theory (pp. 519-526). Springer Berlin Heidelberg.

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.

Varró, D., & Balogh, A. (2007). The model transformation language of the VIATRA2 framework. Science of Computer Programming, 68(3), (pp. 214-234).

Veldhuis, M. M. O. (2013). Multi-Target User Interface design and generation using Model-Driven Engineering. Unpublished dissertation in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Master of Science in Human Media Interaction Faculty of Electrical Engineering, Mathematics and Computer Science University of Twente, the Netherlands.

Wee, A. (2017). New Android Nougat Beta easter egg will live up your childhood memories. Nexus.

Weigert, T., & Weil, F. (2006). Practical Experiences in Using Model-Driven Engineering to Develop Trustworthy Computing Systems. In IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC), (Vol. 1, pp. 208–217). IEEE.

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.

Annexe 1 : Evaluation des outils de développement multiplateformes

Le tableau ci-dessous présente les fonctionnalités et les capteurs supportés par chaque outil (*présentées dans l'étude de cas de chapitre 1*). Les informations présentées ci-dessous ont été recueilles auprès des sites officiels de chaque outil, dans les manuels techniques des solutions, via leurs outils de support ou en utilisant les solutions. De plus, une application de test a été réalisée pour les principales solutions, afin de mieux comprendre leurs environnements de travail et leurs limites. Les décisions ci-dessous resteront valables jusqu'une nouvelle mise à jour d'un outil.

Sensors (S1, S2, S3, S4), Features (F1, …, F19) sont présenté respectivement dans les tableaux "Tableau 1.9" et "Tableau 1.8". Les outils étudiés présenté dans l'entête du tableau sont :

- F1 : PhoneGap + jQuery Mobile
- F2 : PhoneGap + Sencha Touch
- F3 : PhoneGap + Onsen IU
- F4 : PhoneGap + Angular UI
- F5 : PhoneGap + Ionic
- F6 : Titanium Appcelerator
- F7 : Xamarin
- F8 : Flex + Air

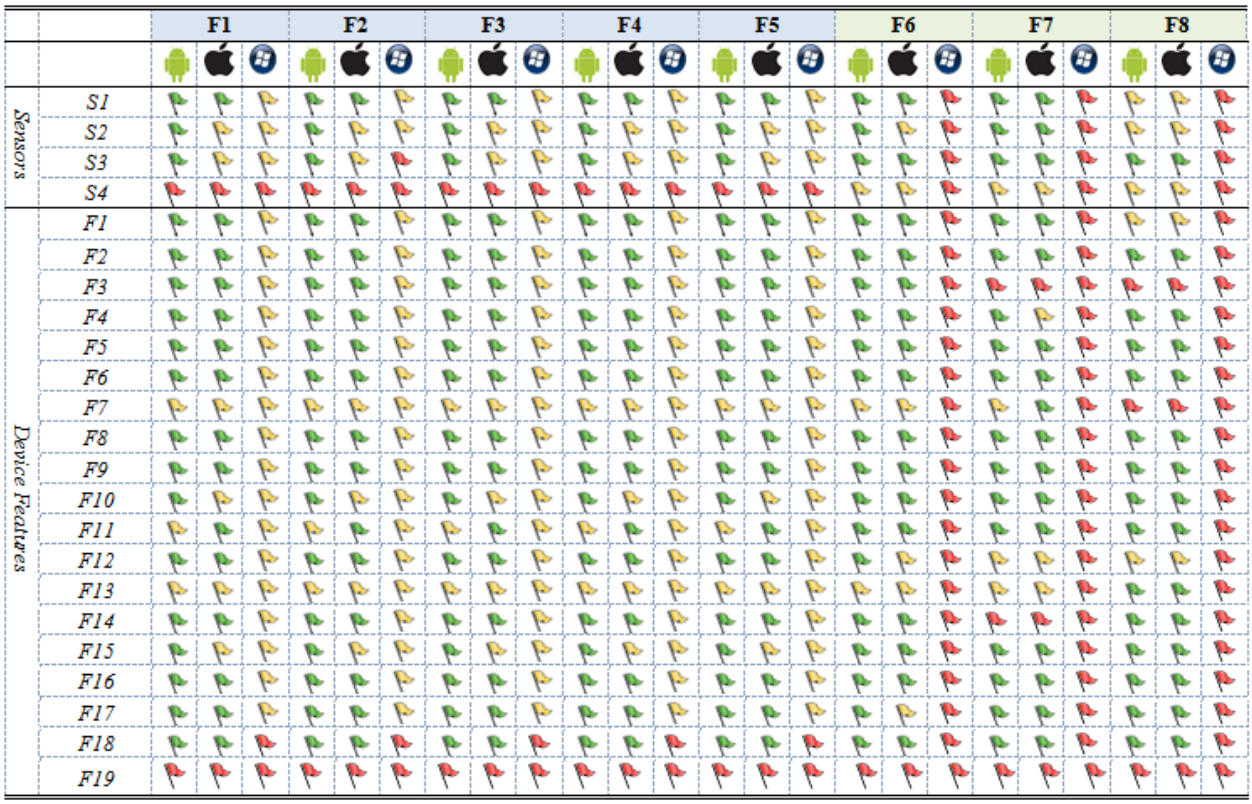
L'échelle utilisée est le suivant :


Well supported (Excellent)


Supported (Sufficient)


Not supported

Tableau A. 1. Evaluation des fonctionnalités natives des outils de développement multiplateformes



Le tableau ci-dessous illustre l'évaluation de chaque outil par rapport aux critères (C1,…, C6) présentés dans le tableau 1.10:

Tableau A. 2. Evaluation des outils par rapport aux critères

	C1	C2	C3	C4	C5	C6
F1	Fast	Good	Good	High	Fast	No
F2	Fast	Good	Good	High	Fast	Yes
F3	Medium	Fair	Good	Very High	Fast	No
F4	Medium	Fair	Good	High	Fast	No
F5	Fast	Good	Very Good	Very High	Fast	No
F6	Fast	Good	Very Good	Very High	Medium	Yes
F7	Medium	Fair	Good	Medium	Medium	Yes
F8	Medium	Fair	Good	High	Medium	Yes

Le tableau ci-dessous présente les différentes plateformes supportées par chaque outil.

✓ : supporté. ✗ : non supporté.

Tableau A. 3. Les plateformes supportées par les outils de développement multiplateformes étudiées

	F1	F2	F3	F4	F5	F6	F7	F8
Android	✓	✓	✓	✓	✓	✓	✓	✓
iOS	✓	✓	✓	✓	✓	✓	✓	✓
Windows Phone	✓	✓	✓	✗	✓	✗	✗	✗
BlackBerry OS	✓	✓	✓	✗	✗	✓	✗	✓
Symbian	✓	✗	✗	✗	✗	✗	✗	✗
Bada	✓	✗	✗	✗	✗	✗	✗	✗
Web OS	✓	✗	✗	✗	✗	✗	✗	✗