République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Master en Informatique

Option: *Système d'Information et de Connaissances (S.I.C)*

Thème

Application de l'approche d'ingénierie dirigée par les modèles pour le développement des applications mobiles (Interfaces graphiques).

Réalisé par :

- **Abdellaoui Nezha Nesrine**

Présenté le 05 Juin 2016 devant le jury composé de MM.

- *Benamar Abdelkrim* *(Président)*
- *Smahi Mohammed Ismail* *(Encadreur)*
- *Chouiti Sidi Mohammed* *(Examinateur)*
- *Boudefla Amine* *(Examinateur)*

Année universitaire: 2015-2016

Application de l'approche d'ingénierie dirigée par les modèles pour le développement des applications mobiles (Interfaces graphiques).

Abdellaoui Nezha Nesrine

05 juin 2016

Remerciement

Je commence d'abord par remercier le bon dieu de m'avoir donné le courage et la volonté pour réaliser ce travail.

Je tiens à remercier Monsieur Smahi d'avoir accepté de diriger ce travail et de m'avoir accompagnée toujours avec un mot d'encouragement positif et optimiste. Aussi pour sa disponibilité, sa pédagogie et ses directives fructueuses qu'il n'a cessé de me prodiguer tout au long de ce projet.

Monsieur Benaamar , je vous adresse mes remerciements pour avoir accepté de juger ce travail et d'en être le président de jury .

Tous mes sincères remerciements à Monsieur Chouiti et Monsieur Boudefla d'avoir aimablement acceptés de juger ce travail.

Enfin je tiens à remercier tous mes enseignants tout au long de mes études.

Résumé

Le développement des applications mobiles représente un segment spécifique dans le marché logiciel .

La diversité et la variété des systèmes d'exploitation mobiles (Android, iOS, Black Berry, Windows Phone, etc.) font l'ingénierie logicielle en face d'un grand défi pour le développement d'une même application pour les différentes plates-formes, et pour atteindre un seuil de rentabilité intéressant.

Dans ce travail nous proposons une solution qui applique le principe de l'approche d'architecture dirigée par les modèles afin de développer une application mobile multiplateformes .

Pour se faire, nous avons développés, en premier lieu, une grammaire qui représente un méta modèle d'une application mobile via un langage DSL en l'occurrence XText. En second lieu, nous avons développés notre méta modèle via Acceleo afin de générer le code spécifique d'une application mobile.

Mots clés : Application mobile, MDA, DSL, Xtext, acceleo.

Abstract

The development of mobile applications represents a specific segment in the software market.

The diversity and variety of mobile operating systems (Android, iOS, Blackberry, Windows Phone, etc.) make software engineering facing a great challenge to develop the same application for different platforms, and achieve an interesting profitability.

In this work we propose a solution which applies the principle of model driven architecture to develop a multiplatform mobile application.

To do so, we have developed in the first place, a grammar that represents a meta model of a mobile application via a DSL language the XText occurrence. Second, we developed our meta model via Acceleo to generate specific code of a mobile application.

Keyword: mobile applications, MDA, DSL, Xtext , acceleo

الملخص

تطوير التطبيقات النقالة تمثل شريحة مهمة في سوق البرمجيات.

تنوع أنظمة تشغيل الهواتف المحمولة (الروبوت، دائرة الرقابة الداخلية، وبلاك بيري، ويندوز موبايل، الخ)جعلوا هندسة البرمجيات تواجه تحديا كبير لتطوير نفس التطبيق لمختلف المنابر ، وتحقيق ارباح مهمة .

في هذا المشروع نقترح الحل الذي يطبق مبدأ قائم على الهندسة النماذج لتطوير تطبيقات الهاتف النقال متعدد المنابر.

للقيام بذلك، وضعنا في المقام الأول قواعد اللغة التي تمثل نموذجا فوقيا عبر لغة مجال معين بواسطة Xtext, ثانيا تم تطوير النموذج عن طريق Acceleo, من اجل الحصول على كود خاص بنظام التطبيقات النقالة.

الكلمات المفتاحية : التطبيقات النقالة , لغة مجال معين , Xtext, Acceleo .

1 Introduction

Le développement mobile devient de plus en plus important en raison de l'utilisation intensive des applications dans les appareils mobiles tels que les Smartphones et les tablettes, avec différents systèmes d'exploitation (Android, IOS, Black Berry, Windows Phone, etc.). À cause de grand nombre de diversité des technologies mobiles, le développement de la même application pour les différentes plates-formes devient une tâche difficile et épuisante. Parce que chaque plate-forme utilise différents langages et outils de programmation, ce qui rend difficile le développement d'applications multiplateformes. Ainsi, il exige des ingénieurs logiciels pour construire des applications sur les plates-formes, tout en assurant le plus grand déploiement.

Le développement des applications mobiles multiplateforme doit être amélioré en utilisant des techniques de développement guidées par le modèle basé sur l'approche MDA pour générer des interfaces utilisateur compatibles aux plateformes. Nous utilisons le modèle indépendant de toutes plateformes PIM afin de réaliser l'application mobile pour des différents systèmes d'exploitation, cela réduit le temps et des ressources de développement. Ensuite Chaque plate-forme cible doit être couvert par le correspondant transformation modèle to modèle qui transforme un modèle indépendant de la plate-forme à une plate-forme spécifique (PSM), et la génération automatique de code en utilisant Acceleo.

Ce travail est organisé comme suit: la deuxième section définit l'approche MDA, tandis que la troisième présente la solution proposée. Dans la quatrième section, nous examinons l'applicabilité de notre approche à travers une étude de cas.

2 Model Driven Architecture (MDA)

Le MDA C'est une proposition à la fois d'une architecture et d'une démarche de développement. L'idée de base du MDA est la séparation des spécifications fonctionnelles d'un système des détails de son implémentation sur une plate-forme donnée. Pour cela, le MDA classe les modèles en modèles indépendants des plates-formes appelés PIM (Platform-Independent Models), et en modèles spécifiques appelés PSM (Platform-Specific Models) [14].

L'approche MDA permet de déployer un même modèle de type PIM sur plusieurs plates-formes (modèles PSM) grâce à des projections standardisées. Elle permet aux applications d'inter opérer en reliant leurs modèles et favorise l'adaptabilité aux évolutions des plates-formes et des techniques. La mise en œuvre du MDA est entièrement basée sur les modèles et leurs transformations[14].

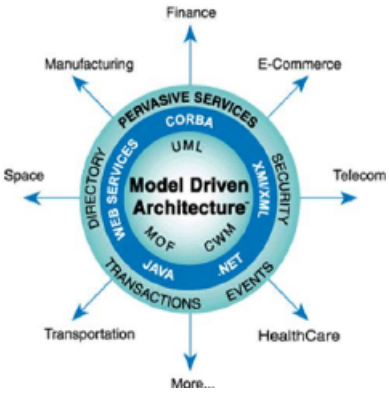

Figure 1: Model Driven Architecture de l'OMG[13].

D'après (Bézivin et Gérard,[8]) la MDA fournit un processus de conception et met en œuvre des outils pour :

- Spécifier un système indépendamment de la plate-forme qui le supporte, et donc réaliser un PIM ;
- Enrichir ce modèle par étapes successives ;
- Spécifier les plateformes ;
- Choisir une plateforme particulière pour le système ;
- Transformer la spécification du système (PIM) en une autre spécification pour une plateforme particulière (PSM) ;
- Transformer un CIM en un PIM et un PIM en un autre PIM ;
- Raffiner le PSM jusqu'à obtenir une implémentation exécutable.

Les trois objectifs préliminaires de MDA sont la portabilité, l'interopérabilité et la réutilisabilité à travers une architecture de séparation des préoccupations.

2.1 Les différents modèles de MDA

2.1.1 Le CIM (Computation Independent Model)

Il est indépendant de tout système informatique. C'est le modèle métier ou le modèle du domaine d'application. Le CIM permet la vision du système dans l'environnement où il opèrera, mais sans rentrer

dans le détail de la structure du système, ni de son implémentation. Il aide à représenter ce que le système devra exactement faire[16].

2.1.2 Le PIM (Platform Independent Model)

Il est indépendant de toute plate-forme technique (EJB, CORBA, .NET,Ě) et ne contient pas d'informations sur les technologies qui seront utilisées pour déployer l'application. C'est un modèle informatique qui représente une vue partielle d'un CIM. Le PIM représente la logique métier spécifique au système ou le modèle de conception. Il représente le fonctionnement des entités et des services. Il doit être pérenne et durer au cours du temps. Il décrit le système, mais ne montre pas les détails de son utilisation sur la plate-forme[16].

2.1.3 Le PSM (Platform Specific Model)

Une fois le PIM est suffisamment détaillé, il est projeté vers un modèle spécifique (PSM). Pour obtenir un modèle spécifique, il faut choisir la ou les plates-formes d'exécution (plusieurs plates-formes peuvent être utilisées pour mettre en œuvre un même modèle). Les caractéristiques d'exécution et les informations de configuration qui ont été définies de façon générique sont converties pour tenir compte des spécificités de la plate-forme[4].

2.1.4 Le PDM (Platform Description Model)

Cette notion n'est pas encore bien définie par l'OMG, pour l'instant il s'agit plus d'une piste de recherche. Un PDM contient des informations pour la transformation de modèles vers une plate-forme en particulier, et il est spécifique de celle-ci. C'est un modèle de transformation qui va permettre le passage du PIM vers le PSM[16].

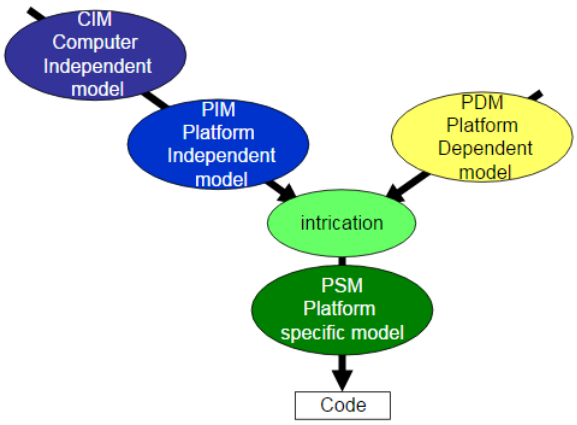

Figure 2: Les modèles du MDA pour réaliser une application[12].

2.2 Les transformations MDA

Les transformations possibles entre ces différents types de modèles sont représentées sur la Figure 3 [3]:

- Transformations PIM -> PIM et PSM -> PSM. Les transformations de type PIM vers PIM ou PSM vers PSM visent à enrichir, filtrer ou spécialiser le modèle. Il s'agit de transformations de modèle à modèle. Elles sont automatisables (ou partiellement automatisable) dans certains cas comme la traduction vers un autre langage mais les transformations de type raffinement ne le sont généralement pas ;
- Transformation PIM -> PSM. La transformation de PIM vers PSM permet de spécialiser le PIM en fonction de la plate-forme cible choisie. Elle n'est effectuée qu'une fois, le PIM suffisamment raffiné. Cette transformation de modèle à modèle est réalisée en s'appuyant sur les informations fournies par le PDM ;
- Transformation PSM -> code. La transformation de PSM vers l'implémentation (le code) est une transformation de type modèle à texte. Le code est parfois assimilé par certains à un PSM exécutable. Dans la pratique, il n'est généralement pas possible d'obtenir la totalité du code à partir du modèle et il est alors nécessaire de le compléter manuellement ;
- Transformations inverses PIM -> PSM et PSM -> code. Ces transformations sont des opérations de rétro-ingénierie (reverse engineering). Ce type de transformation pose de nombreuses difficultés mais il est essentiel pour la réutilisation de l'existant dans le cadre de l'approche MDA.

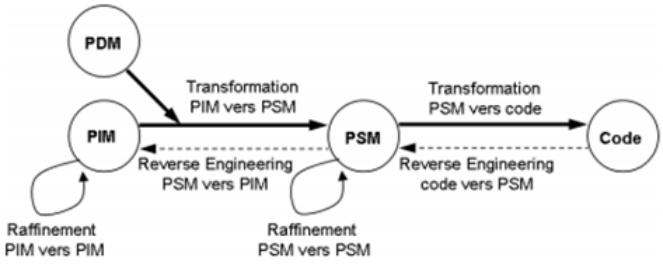

Figure 3: Les modèles et transformations dans l'approche MDA[16].

2.3 L'architecture MDA

Le modèle MDA est décomposé en 4 niveaux [9]:

- Le niveau M0 qui est le niveau des données réelles, est composé des informations que nous souhaitons modéliser. Ce niveau est souvent considéré comme étant le monde réel ;
- Le niveau M1 est composé de modèles d'information. Lorsque nous voulons décrire les informations de M0, le MDA considère que nous faisons un modèle appartenant au niveau M1. Tout modèle est exprimé dans un langage unique dont la définition est fournie explicitement au niveau M2 ;
- Le niveau M2 est donc composé de langages de définition des modèles d'information, appelés aussi méta-modèles. Un méta-modèle définit donc un langage de modélisation spécifique à un domaine et un profil définit un dialecte (une variante) de ce langage ;
- Le niveau M3 qui est composé d'une unique entité qui est le langage unique de définition des méta-modèles, aussi appelé le méta-méta-modèle ou le MOF (Meta-Object Facility). Le MOF définit la structure de tous les méta-modèles qui se trouvent au niveau M2. Le MOF est réflexif, c'est-à-dire que la description du MOF s'applique au MOF lui-même, ce qui permet de dire que le niveau M3 est le dernier niveau de l'architecture. Le niveau M3 correspond donc aux fonctionnalités universelles de modélisation logicielle, alors que le niveau M2 correspond aux aspects spécifiques des différents domaines, chaque aspect particulier étant pris en compte par un méta-modèle spécifique.

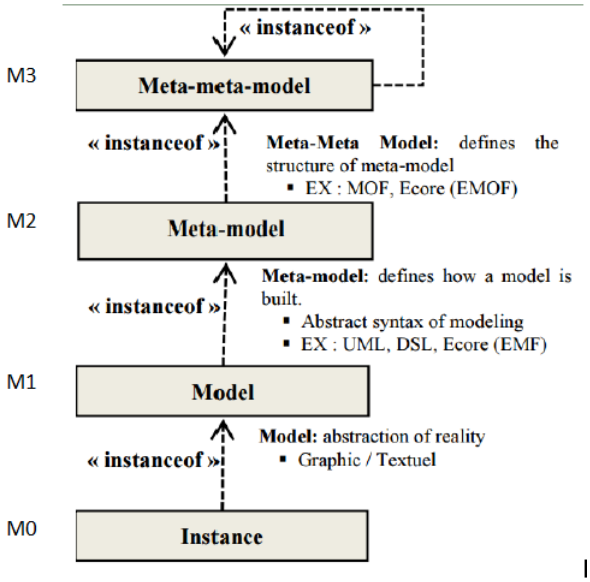

Figure 4: L'architecture MDA[11].

3 Le développement des applications mobiles

3.1 Le développement natif

Le développement natif consiste à développer avec les outils/langages propres à chaque système d'exploitation.

- Objective C avec l'IDE XCode pour iOS (Il vous faudra nécessairement un mac pour pouvoir compiler l'application) ;
- Java avec Eclipse pour Android ;
- C# avec Visual Studio pour Windows Phone.

Cette technique permet d'obtenir le résultat le plus proche du système ciblé. Nous utilisons les outils dédiés à chacun des environnements, et nous disposons des meilleures conditions pour la réalisation et les tests de notre application[6].

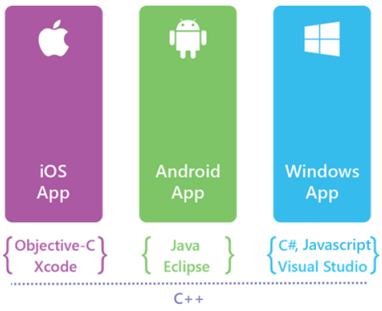

Figure 5: Stratégie de développement natif [5].

3.2 Le développement hybride

Si nous écartons la solution native pour des raisons de coût et de réactivité lors des mises à jour et des correctifs, il faut se diriger vers une approche qui permet de partager le maximum de code possible. Une des manières d'arriver à ce résultat est ce que nous appellons le "quasi-natif". Le principe est simple : nous utilisons un seul langage de programmation, et nous développons une seule fois mais nous utilisons un ensemble d'outils pour compiler et packager une version de l'application adaptée à chaque cible. Pour cela, soit le code "générique"' est traduit dans le langage compris par la plateforme puis compilé nativement, soit il est compris et exécuté par une machine virtuelle lors de l'exécution[5]. Deux frameworks sortent du lot pour atteindre cet objectif : Xamarin et Titanium.

3.3 Le développement web

Enfin, la dernière solution est de développer une application en utilisant les langages web classique : HTML, CSS et Javascript. En fait nous allons pouvoir compiler une application qui ne sera en fait qu'une WebView (une élément qui peut contenir une page Web) dans laquelle nous allons placer notre application crée en HTML. Pour créer une telle application, nous pouvons l'écrire en natif, en hybride (nous pouvons utiliser titanium pour ne créer qu'une webview), ou des solutions clefs en mains comme PhoneGap.

Figure 6: Stratégie de développement hybride(Xamarin) [5].

PhoneGap et Titanium offre quelques API permettant d'intéragir avec le téléphone directement depuis le Javascript[6]. Quelques Frameworks pour le développement web des applications mobiles tels que: PHONEGAP, jQuery Mobile..

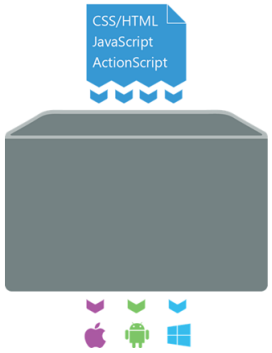

Figure 7: Stratégie de développement web [5].

4 Approche MDA, pour le développement d'application mobile

Dans cette partie, nous expliquons la solution étape par étape :

- Nous supposons que l'application en langage naturel est le CIM qui aide à la construction des modèles successeurs le PIM et le PSM ;
- Nous devons transformer le CIM vers un modèle PIM à l'aide de langage DSL[1], en utilisant l'outil Xtext [1] qui est une composante de TMF (Textual Modeling Framework) intégré dans le framework de modélisation d'Eclipse (Eclipse Modeling Framework : EMF) ;
- Nous implémentons Xtext pour définir un langage de méta-modélisation de notre application mobile ;
- Nous utilisons cette grammaire pour écrire notre modèle (code PIM) ;
- Ensuite, nous appliquons la théorie de transformation de modèle PIM vers un modèle PSM pour chaque plateforme mobile ;

[1]Domain specific language:est un langage de programmation dont les spécifications sont conçues pour répondre aux contraintes d'un domaine d'application précis.

- Afin de définir le PSM, nous utilisons un générateur de code acceleo qui est une implémentation de l'Object Management Group (OMG) sur la norme de MOF, MOF Models to Text (MOFM2T).

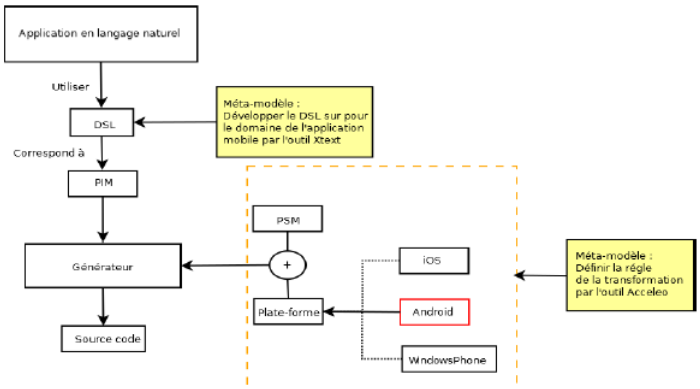

Figure 8: La vue de l'ensemble de solution [15].

4.1 Méta-modélisation

La méta-modélisation est l'activité qui consiste à définir le méta modèle d'un langage de modélisation. Elle vise donc à bien modéliser un langage, qui joue alors le rôle de système à modéliser[2].

Face au succès de ce mécanisme d'ouverture, l'OMG a finalement adopté une approche permettant de définir et utiliser des langages spécifiques aux métiers ou technologies en jeu. Le Domain Specific Language (DSL), est proposé de nouveaux standards et une nouvelle architecture pour la définition et la manipulation de langages de modélisation. Ce langage apporte une solution ciblée et efficace à un ensemble restreint et particulier de problèmes de modélisation ou de programmation. Il existe un très grand nombre de DSLs avec des niveaux d'abstraction très différents[3].

4.2 Conception de la méta-modélisation

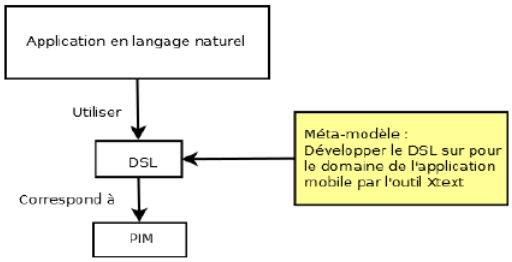

Figure 9: L'étape de réalisation de PIM[15].

La méta-modélisation repose sur la construction de modèles représentant eux-mêmes des modèles, c'est-à-dire des modèles de modèles. Il existe différents langages pour l'écriture de métamodèles comme le MOF, ECore, Kermeta ou encore DSL Tools de Microsoft.

Enfin, nous allons créer une grammaire de transformation qui est capable de transformer un fichier écrit selon cette syntaxe textuelle et de construire une instance appropriée de notre modèle de domaine. La Figure 9 représente l'idée du processus de la transformation le CIM vers le PIM.

4.3 Génération de code

Une fois les modèles d'analyse et de conception sont réalisés, le travail de génération de code peut commencer. Cette phase la plus délicate du MDA doit aussi utiliser des modèles.

MDA considère que le code d'une application peut être facilement obtenu à partir de modèles de code. La différence principale entre un modèle de code et un modèle d'analyse ou de conception réside dans le fait que le modèle de code est lié à une plate-forme d'exécution. Dans le vocabulaire MDA, ces modèles de code sont appelés des PSM (Platform Specific Model). Les modèles de code servent essentiellement à faciliter la génération de code à partir d'un modèle d'analyse et de conception. Ils contiennent toutes les informations nécessaires à l'exploitation d'une plate-forme d'exécution, comme les informations permettant de manipuler les systèmes de fichiers[17].

4.4 Conception de la génération de code

Pour la conception de réalisation du générateur de code, nous devons définir d'abord la règle de la transformation pour le générateur de code. En utilisant acceleo pour profiter des technologies MDA pour

accroître la productivité de leurs développements informatiques, il permet de générer des fichiers à partir de modèles UML, MOF, EMF, Xtext.Il s'agit d'un plugin caractérisé par [10]:

- Son intégration complète à l'environnement Eclipse et au framework EMF ;
- Gestion de la synchronisation entre le code et le modèle ;
- La génération incrémentale ;
- La simplicité d'adaptation à tout type de cible technique.

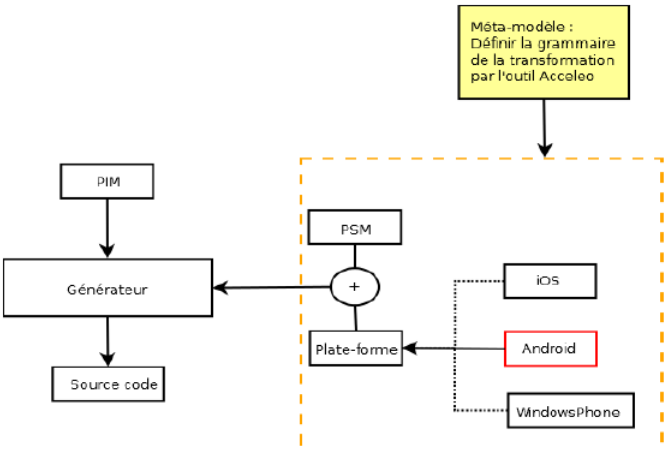

Figure 10: L'étape de réalisation de générateur[15].

5 Solution développée

5.1 La plateforme cible ń Android ż

Android [2] est un système d'exploitation ouvert dontle code source est

librement accessible (contrairement aux systèmes de Apple ou Microsoft), ce qui permet à n'importe quel fabricant de l'intégrer dans son système gratuitement.

Lors de la création d'un nouveau projet, voici l'arborescence qui est automatiquement générée :

- src : Le répertoire de l'ensemble des sources du projet. C'est dans lequel que vous allez ajouter et modifier le code source de l'application ;
- gen : répertoire contenant l'ensemble des fichiers générés par le plugin correspondant à l'environnement de développement. Aucune modification ne doit être faite dans ces fichiers ;
- androidManifest.xml : fichier XML décrivant l'application et ses composantes, tels que les activités, les services... etc ;
- res : répertoire contenant toutes les ressources (les images, les vues de l'interface graphique... etc) nécessaires à l'application. Ce répertoire est structuré par défaut de la manière suivante :

[2] http://topandroid.fr/faq/quest-ce-que-android-cest-quoi-definition-simple#.VzHGLISLTDf

– res/drawable : contient les ressources de type image ;

– res/layout : contient les descriptions des interfaces graphiques au format XML (les vues) ;

– res/xml : contient les fichiers XML supplémentaires (non présents par défaut) ;

– res/menu : contient la description des menus, composants très courants d'une vue ;

– res/values : contient diverses ressources, tels que les textes qui sont empaquetées sans aucun traitement.

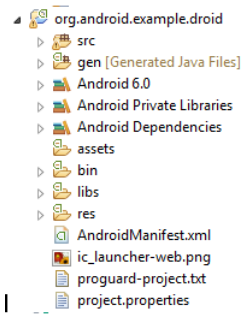

Figure 11: Arborescence d'un projet Android.

Le but de l'étude de la plateforme Android est de réaliser un générateur de code capable de générer le code source d'une application mobile compatible à la plateforme de l'appareil mobile à partir de modèle PIM.

5.2 Implémentation de langage DSL

Notre DSL est développé sous la conception de regrouper des éléments en trois composants : View, Ressources(modele) et Screen(controller). Cela nous permet de réaliser la transformation de code pour les plateformes spécifiées. Vu que nous avons bien conçus la structure, nous pouvons transformer le code facilement. Il soutient également la modification de modèle si l'éxigence a changée. La Figure 12 est la structure de notre DSL.

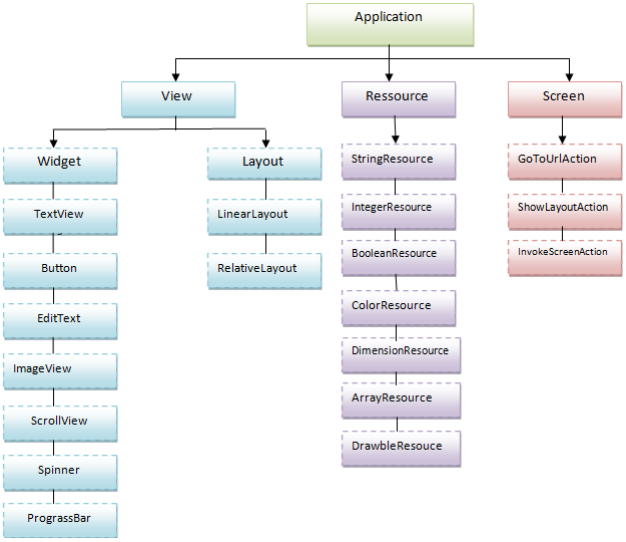

Figure 12: Le concept de DSL.

5.2.1 Application

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

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

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

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

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

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

Figure 13: Grammaire Xtext du concept Application.

5.2.2 Ressource

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

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

Figure 14: Grammaire Xtext du concept Ressource.

```
DrawableResource:
    (BitmapDrawableResource | TransitionDrawableResource)
;

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

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

ArrayResource:
    IntegerArrayResource | StringArrayResource
;

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

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

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

Figure 15: Grammaire Xtext des attributs du concept Ressource.

5.2.3 View

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

```
View:
    Widget | Layout
;
```

Figure 16: Grammaire Xtext du concept View.

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

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

Figure 17: Grammaire Xtext du concept Widget.

5.2.4 screen

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

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

Figure 18: Grammaire Xtext du concept Screen.

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

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

- ShowScreen : C'est l'action qui soutient la fonction de l'affichage de l'interface. Sa dèfinition commence avec le mot-clé show suivi par le nom de l'interface.

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

GoToURLAction:
    'goTo' url=STRING
;

ShowLayoutAction:
    'show' layout=[Layout]
;

InvokeScreenAction:
    'invoke' activity=[Screen]
;

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

Figure 19: Grammaire Xtext du concept Action.

5.3 Implémentation de générateur de code

Une fois que le développeur a modélisé le modèle PIM qui est compté comme l'entrée de génération, Il est très important de définir les règles de transformation parce que cette dernière est la clé de génération de code de la plateforme cible à partir de modèle PIM.

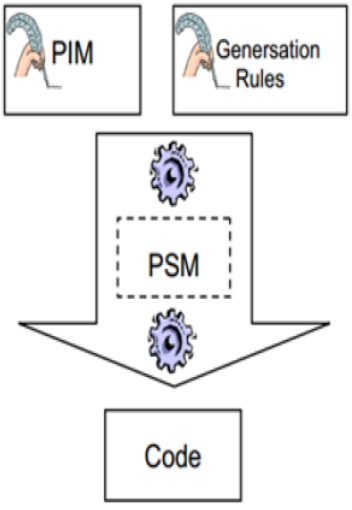

Figure 20: Les transformation de l'approche MDA[7].

Afin de développer le générateur de code, nous pouvons exploiter l'Acceleo qui est un générateur de code source de la fondation Eclipse permettant de mettre en oeuvre l'approche MDA (Model driven architecture) pour réaliser des applications. Notre génération de code Android est basée sur le langage DSL, qui est utilisé par Acceleo pour générer la structure d'application. La relation entre les interfaces telle que l'association et l'héritage, sont respectés lors de la génération de code. Lorsque le code PIM représente une composante de l'API Android, la génération comprend éventuellement les importations selon la déclaration et les paramètres d'attributs.

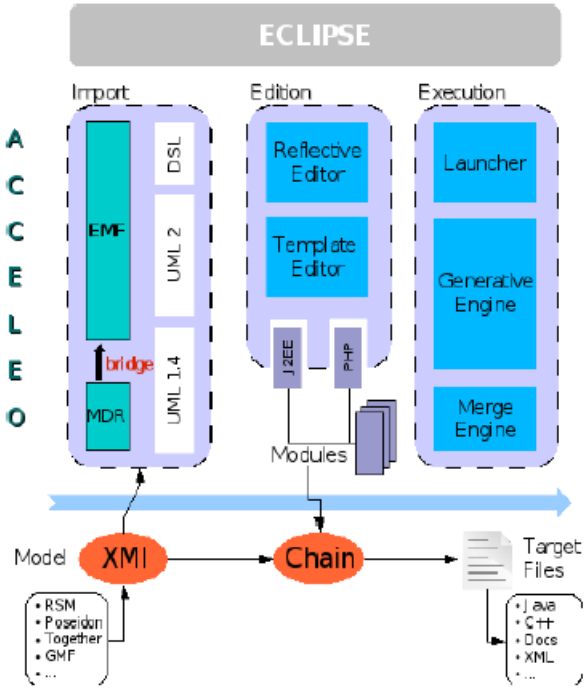

Figure 21: L'architecture de acceleo[10]

5.3.1 Le module generateManifest

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

5.3.2 Le module generateView

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

5.3.3 Le module generateResource

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

5.3.4 Le module generateScreen

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

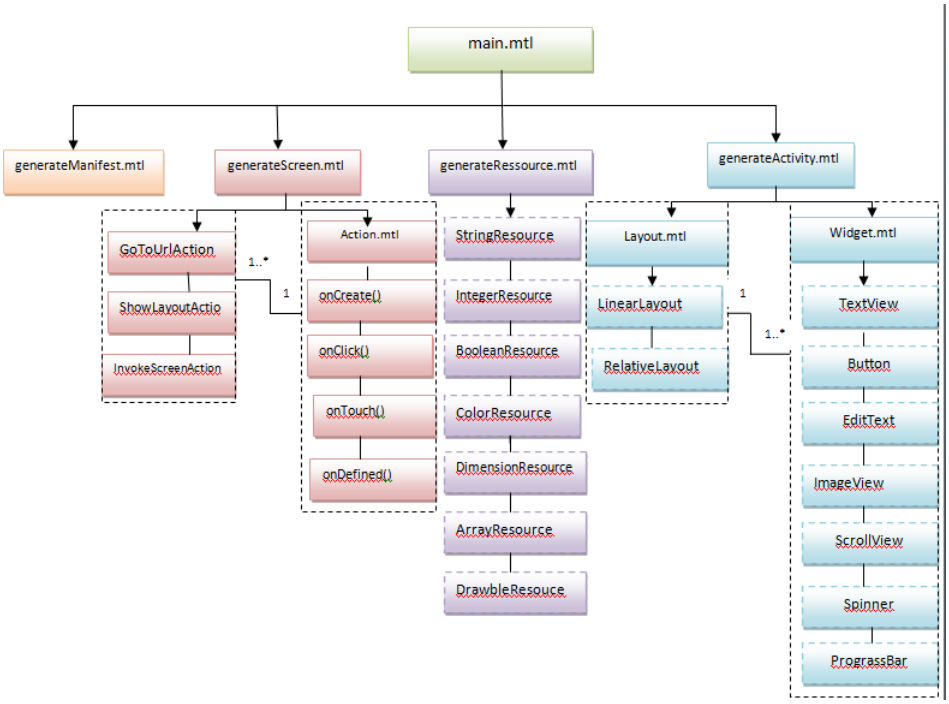

Figure 22: Le concept des transformations pour la plateforme Android.

5.4 Application de la méthode proposée

Dans notre étude, à partir de notre DSL nous réalisons deux modéles PIM , qui jouent le rôle de modèle d'entrée afin de générer le code de plateforme cible. Le premier PIM modélise une application mobile simple, l'autre représente une application de journal d'appel.

5.4.1 Application 1 : Une application mobile simple

L'application est composée de deux classes main-activity et second-screen. L'interface résultante est représentée dans la Figure 23, et le modèle PIM responsable de transformation est représenté dans la Figure 24. La description est la suivante:

- Classe main-activity
  - "Textview" pour afficher un commentaire dans la première intent ;

- "Boutton Next" qui permet l'utilisateur de passer au deuxième intent.

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

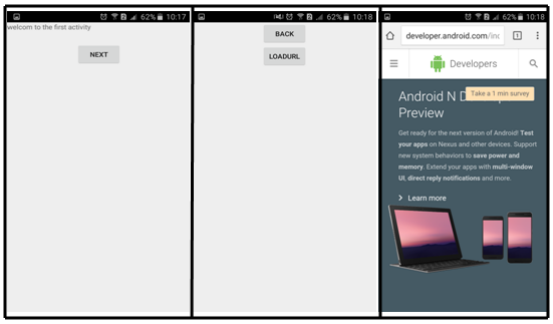

Figure 23: Interface de la première application.

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

screen Main {
    show main
}

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

        }
        width: 50px;
    }

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

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

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

Figure 24: Modèle PIM de la première application.

5.4.2 Application2 : Application journal d'appel

L'application est composée d'une seule classe main-activity. L'interface résultante est représentée dans la Figure 25, et le modèle PIM responsable de transformation représenté dans la Figure 26. la description est la suivante:

- Classe main-activity
  - "ScrollView" Contenant la mise en page pour une hiérarchie d'affichage ;
  - "Textview" permet l'affichage de la liste des appels.

Figure 25: Interface du journal d'appel.

```
application journalappel=>org.journal.appel.droid
version:1 => "1.0"
sdk:{
    min:11;
    max:14;
}

android.permission.READ_CALL_LOG
android.permission.WRITE_CALL_LOG

screen Main{
    show activity_main
}
layout activity_main {
    layout:{
    width:fill_parent;
    height:fill_parent;

}
orientation:vertical;
#scrollview{
layout:{
    width:fill_parent;
    height:fill_parent;
}
#textView textview_call{
layout:{
    width:fill_parent;
    height:fill_parent;
    centerHorizontal:TRUE;
    centerInParent:TRUE;
}

}
}
}
```

Figure 26: Modèle PIM du journal d'appel.

5.5 Èvaluation

À la fin de notre étude sur l'approche MDA pour le développement des applications mobiles, nous pouvons analyser les résultats obtenus:

- Le générateur de code a donné le code qui peut bien fonctionner pour tous les éléments testés ;
- Le code en forme.xml est bien structuré comme le code qui est réalisé manuellement ;
- Le code en forme.java fait marcher les actions d'application exactement comme le code réalisé manuellement ;
- La méthodologie de développement d'autre plateformes(IOS, BlackBerry, WindowPhone ..) est pareille. Ce qu est essentiel est que nous devons avoir des connaissances de chaque plateforme.

6 Conclusion

Le but de ce travail est d'étudier l'approche d'ingénierie dirigé par modèle qui permet de concevoir des applications en séparant les préoccupations et en plaçant les notions de modèles, méta modèles et transformations nécessaires. C'est la raison pour laquelle, nous proposons une approche d'architecture dirigée par les modèles MDA pour le développement d'application Android. Cette approche comprend la modélisation basée sur un langage Xtext et génération automatique de code en utilisant acceleo dans le but d'accélérer et de faciliter le développement d'application Android.

7 Perspective

Pour les futures études, il convient d'améliorer le générateur de code pour automatiser la génération de code source pour toutes les plateformes mobiles (IOS, Black Berry, Windows phone Ĕ), et de compléter notre langage (grammaire) au niveau d'interface (RadioButton, CheckBox, TableLayout..). Aussi au niveau des actions, il faut rajouter d'autres fonctions qui permettent de réaliser d'autres tâches et effectuer de nouvelles options (bases de données, services..).

Contents

References

[1] BERNARD Alain. http://alain-bernard.developpez.com/tutoriels/eclipse/creation-grammaire-xtext/. 2012.

[2] Combemale Benoit. Ingénierie dirigée par les modèles (idm). 2009.

[3] Hardebolle Cécile. Composition de modèles pour la modélisation multi-paradigme du comportement des systèmes. 2008.

[4] Projet ACCORD (Assemblage de composants par contrats en environnement ouvert et ré parti)*. La démarche mda. 2002.

[5] Margraff Etienne. 3 approches pour créer son application mobile sur toutes les plateformes. 2014.

[6] Grafikart. Développement d'application mobile. 2013.

[7] BENOUDA Hanane, ESSBAI Redouane, AZIZI Mostafa, and MOUSSAOUI Mimoun. Modeling and code generation of android applications using acceleo. 2016.

[8] Chettaoui Hanène. Interopérabilité entre modèles hétérogènes en conception coopérative par des approches d'ingénierie dirigée par les modèles. 2008.

[9] Bézivin Jean and Blanc Xavier. Mda : Vers un important changement de paradigme en genie logiciel. 2002.

[10] MUSSET Jonathan, JULIOT Etienne, and LACRAMPE Stéphane. http://www.acceleo.org/doc/obeo/fr/acceleo-2.6-reference.pdf. 2006-2008.

[11] M. Lachgar and A. Abdali. Şmodeling and generating the user interface of mobile devices and web development with dsl. 2015.

[12] GARNIER Laurent. Stratégie de test au sein du processus d'évolution d'architecture de sodifrance. 2011.

[13] Probst Richard. http://www.omg.org/mda/. 2015.

[14] Diaw Samba, Lbath Rédouane, and Coulette Bernard. Etat de l'art sur le développement logiciel basé sur les transformations de modèles. 2010.

[15] Jariya SIRISANG. études de l'approche d'ingénierie dirigée par les modèles pour le développement des applications mobiles. 2015.

[16] ANDRE Sylvain. Mda (model driven architecture) principes et états de l'art. 2004.

[17] Blanc Xavier. Mda en action "ingénierie logicielle guidée par les modèles. 2012.