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

Mémoire de fin d'études

Pour l'obtention du diplôme de Master en Informatique

Option: *Génie Logiciel (GL)*

Thème

Développement d'un atelier graphique pour la génération d'applications mobiles multi-plateforme.

Réalisé par :

- **BELHADJ KACEM Soufyane.**
- **SENOUSSAOUI Ikram.**

Présenté le 15 Juin 2017 devant le jury composé de MM.

- *BENAMAR Abdelkrim* *(Président)*
- *SMAHI MED ISMAIL* *(Encadrant)*
- *MESSABIHI Mohammed* *(Examinateur)*

Année universitaire : 2016-2017

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

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

Un remerciement particulier à :

Mr. SMAHI Mohammed Ismail, notre enseignant responsable, pour nous avoir encadrés tout au long de ce travail. Merci pour ses conseils, sa disponibilité et son implication. Ainsi qu'à tous nos enseignants de master Génie Logiciel.

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

Dédicace

Aux membres de nos familles qui nous ont donné le courage et la volonté afin que nous réussissions dans nos études.

Aux précieux conseils de nos amis, et spécialement Salim Saïdi.

À tous ceux qui nous ont donné le courage et qui ont cru en nous.

Résumé

Ce sujet de mémoire vise à étudier et appliquer l'approche d'ingénierie dirigée par les modèles pour le développement des applications mobiles.

Entre des applications pour Android, iOS et Windows Phone, la demande en développeurs n'a jamais été aussi grande. Chacun a un langage de développement qui lui est propre. Cela oblige les entreprises à avoir un panel de personnel possédant les capacités de développement dans différentes plate-formes pour but de créer une application sur chacune d'entre elles.

D'où la naissance des applications multi-plateformes. Et vu le nombre important des plate-formes existantes, ce type de développement est devenu très complexe.

Cela nous ramène à penser à une solution pour améliorer le développement des applications mobiles multi-plateformes en se basant sur l'approche **MDA**. Le but est d'offrir un atelier permettant de manipuler des interfaces utilisateurs compatibles avec les différentes plate-formes et qui utilisent des bases de données (utilisations des bases de données distantes) afin de générer des applications react-native qui s'exécuteront sur les systèmes d'exploitation ANDROID, iOS et WindowsPhone.

Toute la partie de développement de ce projet fut mis en place de manière itérative, basée sur la méthodologie Agile.

Mots-clés : MDA, développement multi-plateforme, react-native, applications mobiles.

Abstract

The subject of this memory is in the field of Software Engineering, and is designed to study the model-driven engineering approach (MDA).

Between applications for Android, iOS and WindowsPhone, the demand for developers has never been greater. These three big players compete for the top of the standings. Each of them has its own programming language, this requires companies to form a team of developers with development capabilities in differents platforms in order to create applications on each of them, and this is the reason for the appearance of cross-platform applications. Development has become very complex because of this multiplicity of platform execution.

This leads us to think of a solution to improve the development of cross-platform mobile applications using the model-driven engineering approach (MDA). The object is to generate user interfaces that are compatible with differents platforms and that use databases (remote databases) in order to generate react-native applications that will be running on ANDROID, iOS and WindowsPhone operating systems.

Keywords: MDA, cross-platform, react-native, mobile applications.

ملخص

موضوع هذه الذاكرة في مجال هندسة البرمجيات, و هو يهدف إلى دراسة الهندسة الموجهة بالنماذج من أجل تطوير التطبيقات النقالة.

بين تطبيقات الأندر ويد , أيفون و ويندوزفون , لم يحدث أن عرفنا طلبا كبيرا على مطوري المجال. يتنازع العمالقة الثلاث على رأس الصدارة, لكل منهم لغته الخاصة في البرمجة . هذا ما يفرض على الشركات تكوين فريق من المطورين ذوي قدرات برمجية في كل منصات التنفيذ لإنشاء تطبيقات على كل منهم.

و هذا ما أدى إلى ظهور ما نسميه في مجال تطوير التطبيقات النقالة, بالتطبيقات متعددة المنصات. و بالنظر إلى العدد الهائل من المنصات الموجودة حاليا أصبح هذا النوع من التطور معقدا جدا.

هذا ما دفعنا إلى التفكير في إيجاد حل لتعزيز تطوير التطبيقات النقالة متعددة المنصات باستعمال الهندسة الموجهة بالنماذج.الهدف من ذلك هو صنع واجهات المستخدم تتوافق مع منصات مختلفة إضافة إلى دمج قواعد البيانات فيها (استخدامات قواعد البيانات عن بعد) المحمول من خلال تقديم ورشة عمل لإنشاء تطبيقات من شأنها أن تعمل على أنظمة التشغيل: الأندر ويد ,أيفون وَ ويندوزفون.

كلمات مفتاحية: الهندسة الموجهة بالنماذج, التطبيقات النقالة, التطبيقات متعددة المنصات.

L'avant-propos

À cause du grand nombre et de la diversité des technologies mobiles, le développement de la même application pour les différentes plate-formes devient une tâche difficile et épuisante.

Dans le cadre d'un thème de master de fin d'études proposé par Mr "SMAHI Mohammed Ismail", Maître Assistant à l'université de Tlemcen, un atelier de modélisation graphique pour concevoir des applications compatibles avec différents systèmes d'exploitation mobile, a été développé et délivré.

Ce rapport fournit toutes les informations concernant l'approche MDA et le framework react-native. De plus, toutes les étapes importantes de la création d'atelier pour générer des applications mobiles ainsi qu'une application teste y sont documentés.

Sommaire

Introduction Générale

I. Introduction

Actuellement, le nombre d'utilisateurs de smartphones est en constante augmentation.

De ce fait, le développement mobile gagne de l'importance d'année en année. Grâce aux différents SDK[1] et aux différentes boutiques en ligne, de nombreuses applications font leur apparition engendrant ainsi de nouveaux besoins qui poussent les utilisateurs à migrer plus rapidement sur un smartphone.

En effet, le nombre croissant d'acteurs métier que ce nouveau marché a attiré, a amplifié une problématique qui, sans concerner directement l'utilisateur, constitue un obstacle de taille pour le développeur, il s'agit là de la multiplication des supports et plate-formes.

II. Problématique

Comme mentionné précédemment, l'univers du développement mobile est vaste et plusieurs barrières d'entrée sont présentes.

Premièrement, des applications mobiles sont souvent fournies pour plusieurs systèmes d'exploitation (Android, iOS, Windows Phone). Il est essentiel de développer séparément pour chaque plate-forme en raison des différents détails dans la programmation des interfaces, des bibliothèques et des langages de programmation. De plus, dans le milieu du développement mobile, toutes les applications ne se ressemblent pas. Nous ne parlerons pas d'applications de type jeux, productivité ou communication mais il est plutôt question des différentes approches de développement d'applications smartphone.

Comme conséquence de cette première barrière, développer une même application mobile pour deux plate-formes différentes iOS et android par exemple, nécessite une période importante de travail ainsi qu'un budget énorme.

[1] **SDK**: est un ensemble d'outils d'aide à la programmation proposé aux développeurs d'applications mobiles.

La deuxième barrière d'entrée est très importante dans notre cas, c'est le facteur humain ou bien les ressources humaines. En fait, développer une application pour chaque plate-forme, nécessite la connaissance des différents langages de programmation. Si nous avons choisi de créer l'application sur la plate-forme iOS, nous aurons donc la nécessité d'avoir dans l'équipe un développeur connaissant les principes de la programmation sous iOS. Les ressources humaines sont donc une variable importante à considérer au moment de la création d'équipe de développement.

Ces barrières dans le développement d'applications peuvent être résumées de la manière suivante :

- ➔ Le multi-plateforme est devenue très complexe à cause de multiplicité des plate-formes.
- ➔ Développer une même application sur différentes plate-formes nécessite un temps important ainsi qu'un grand budget.
- ➔ Les Ressources Humaines à disposition dans l'entreprise auront une importance dans le développement mobile multi-plateformes.

Le défi pour notre travail est la possibilité de créer une application multi-plateformes tout en enlevant cette contrainte des ressources humaines à disposition dans une entreprise.

La question suivante résume cette problématique :

Comment créer une application multi-plateformes (iOS, Android et Windows Phone), sans avoir besoin de plusieurs développeurs connaissant les langages de développement propres à chaque plate-forme ?

III. Démarche

Depuis des années, différentes solutions ont été proposées pour traiter cette problématique du développement multi-plateforme. Parmi ces solutions nous trouvons l'approche de l'ingénierie Dirigée par les Modèles (MDA) qui est une pratique d'ingénierie des systèmes utilisant les capacités des technologies informatiques pour décrire à la fois le problème (besoin) et sa solution à travers les modèles, les concepts, et les langages. Cela peut simplifier le développement et réduire le temps et le coût d'implémentation. Le modèle représente clairement l'idée de l'application à l'aide des outils de modélisation comme UML (Unified

Modeling Language) et s'il y a des erreurs, il est facile et rapide de corriger en revoyant et modifiant les modèles.

L'idée de l'approche d'ingénierie dirigée par les modèles, que nous avons adopté pour notre travail, est que le développement de l'application est guidé par les modèles. Elle est très bien adaptée avec l'approche Cross-Platform[2] afin de réaliser l'application mobile pour des différents systèmes d'exploitation. Le code est généré à partir des modèles qui sont définis par le développeur.

En effet, en premier lieu, nous avons conçu et développé à l'aide de l'outil Xtext, une grammaire générale permettant de définir un langage dédié aux applications mobiles DSL[3] (Domain Specific Language). En second lieu, et en se basant sur le méta-modèle généré par Xtext à partir de notre grammaire, nous proposons un atelier de modélisation graphique à l'aide de l'outil Sirius pour générer des modèles indépendants des plate-formes PIM (Platform Independent Model). Ces derniers, seront utilisés pour définir la sémantique de notre langage en les traduisant en un code react-native compatible avec les trois plate-formes iOS, android et WindowsPhone.

IV. Plan de travail

Ce travail comporte quatre chapitres, dans le premier nous présentons le domaine de développement des applications mobiles. Dans le second chapitre, nous faisons un comparatif sur les frameworks du développement mobile, tandis que dans le troisième chapitre nous parlons de l'approche d'ingénierie dirigée par les modèles MDA, et dans le dernier chapitre, nous détaillons la solution proposée.

Comme annexe :

- Annexe A : Définir une grammaire DSL avec Xtext.
- Annexe B : Création d'un éditeur graphique avec Sirius.

[2]**Cross-Platform**: un logiciel multi-plateforme est conçue pour fonctionner sur plusieurs plate-formes.
[3] **DSL :** un DSL (Domain Specific Language) est un langage dédié à une problématique métier spécifique.

Chapitre I : Développement des applications mobiles

I. Introduction

Les technologies de l'information et de la communication ont été la révolution la plus importante qui a marqué ces dernières années, grâce aux applications mobiles qui sont devenues une partie indispensable des smartphones.

Pour avoir plus de détails sur le domaine de développement des applications mobiles, dans cette partie du rapport nous allons répondre aux questions suivantes : qu'est-ce qu'une application mobile ? Quelles sont les solutions pour développer une application mobile ? Et quelle solution choisir ?

II. Applications mobiles

Une application mobile est un logiciel ou un programme téléchargeable, gratuit ou payant et exécutable à partir d'un système d'exploitation sur un appareil électronique à savoir les smartphones ou les tablettes. **[Bathelot, 2017]**

Les applications mobiles sont adaptées aux différents environnements techniques des smartphones et à leurs contraintes et possibilités ergonomiques (écran tactile notamment). Elles permettent généralement un accès plus confortable et plus efficace à des sites ou services accessibles par ailleurs.

III. Les solutions pour développer une application mobile

Actuellement, Il existe trois solutions possibles pour le développement des applications pour mobile. Le choix de ses trois solutions se fait en prenant compte du système d'exploitation sur lequel nous voulons utiliser l'application et des caractéristiques que doit avoir cette dernière.

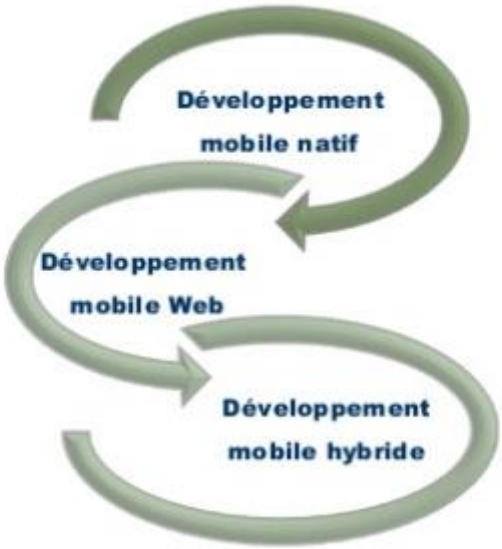


Figure I.1:Les solutions pour le développement mobile [Marwa, 2017]

Il est possible de regrouper ces solutions dans les points suivants:

III.1. Le développement natif

Le développement natif consiste à développer des applications avec les outils et les langages propres à chaque système d'exploitation. Le fait d'utiliser cette méthode et développer une application native permet de bénéficier des éléments et toutes les fonctionnalités de l'API (GPS, appareil photo, etc.) pour un développement plus rapide et plus propre.

Ces types d'applications sont téléchargés depuis un magasin d'applications mobiles (store), puis installées sur le périphérique, elles peuvent aussi fonctionner en mode hors ligne (sans connexion internet).

Le problème rencontré dans cette méthode est que nous devons développer un code pour chaque plate-forme par exemple, pour les deux plate-formes android et iOS, il faut avoir deux codes sources complètement différents, par conséquent, le temps et le prix de développement augmentent et les mises à jour seront pénibles à faire.

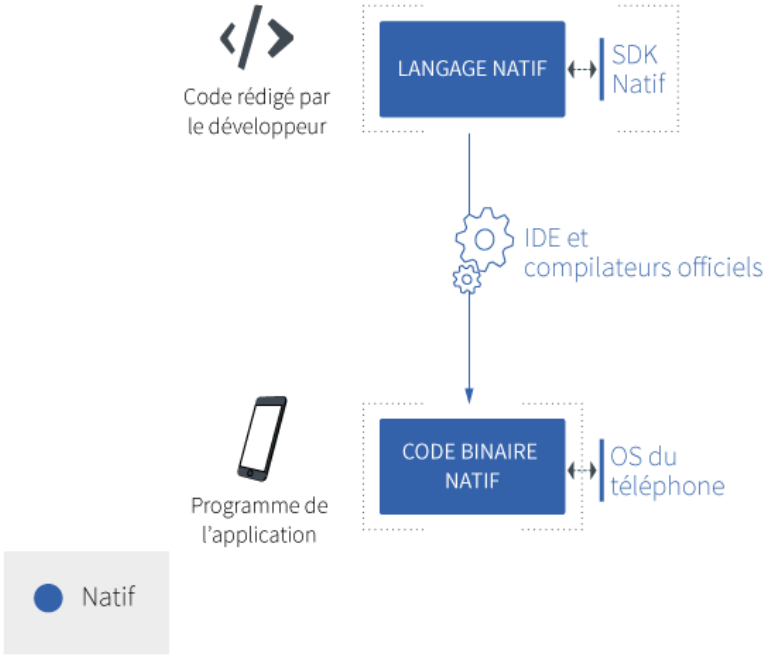


Figure I.2: Le développement natif d'une application mobile [Emilie, 2015]

III.2. Le développement web(HTML)

La deuxième solution consiste à développer des applications en utilisant les langages web classiques comme : HTML, CSS et JavaScript. Ces applications fonctionnent comme des sites web, et dans ce cas-là elles ne sont pas concernées par les problèmes de compatibilité, par contre nous ne pouvons pas profiter de toute la panoplie de fonctionnalités offertes par l'API du mobile.

III.3. Le développement hybride

Certaines communautés ont très bien compris le problème de compatibilité et proposent d'autres méthodes qui permettent, avec un seul code, de lancer des applications sur les différentes plate-formes d'exécution, **"Write once, run everyWhere".** Une application hybride est développée à partir des langages web (HTML, JavaScript, CSS….), et qui s'appuie sur des technologies natives mobiles afin de pouvoir utiliser des fonctionnalitées du smartphone telles que : caméra, GPS, etc…, ces types d'applications sont aussi téléchargées depuis les magasins d'applications en ligne et installées sur le mobile. La solution la plus

connue étant Titanium Platform pour décrire une application en utilisant du JavaScript et leur API, une autre nouvelle solution qui vient d'apparaître est l'utilisation du framework react-native.

L'avantage majeur de ces solutions est de coder avec un langage connu qui est le JavaScript pour avoir un code unique compatible avec plusieurs plate-formes, par exemple, l'utilisation du framework react-native permet d'avoir un code source d'applications mobiles exécutables dans trois systèmes d'exploitation : android, iOS et WindowsPhone.

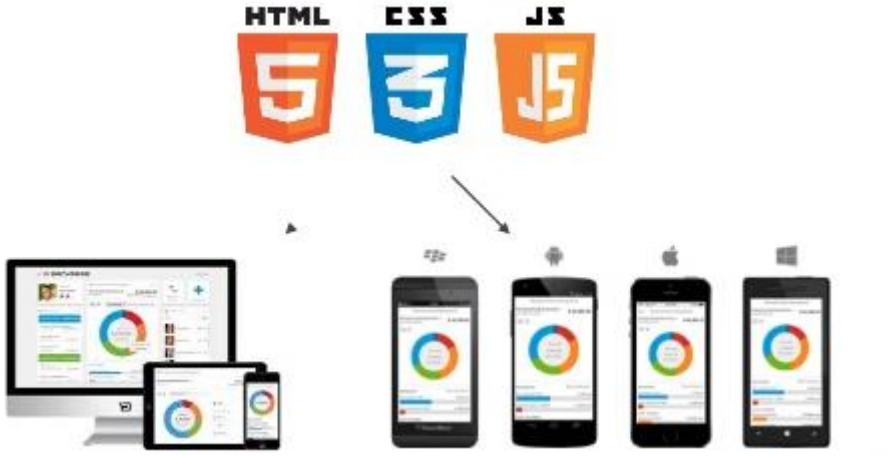


Figure I.3:Le principe de la solution hybride [Marwa, 2017]

IV. Choix de la solution

Le choix d'une solution est une phase très importante qui précède le développement de l'application. Pour développer une application qui doit utiliser le maximum de fonctionnalitées, il est plus intéressant de se former dans le langage souhaité et de la développer en utilisant les outils propres au système choisi.

Par contre, le développement HTML permet de développer une application assez rapidement et qui fonctionne de la même manière dans les différentes plate-formes d'exécution, dans ce cas, il faut faire attention aux caractéristiques des mobiles, tandis que la dernière solution "**le développement hybride**" est utilisable pour avoir un code multi-plateformes, c'est une solution très à la mode vu le nombre important des plate-formes qui existent actuellement sur le marché.

Le tableau suivant représente les fonctionnalitées de chaque solution:

Fonctionnalité \ Type d'application	Application Web	Application native	Application hybride
Accès aux fonctionnalités natives (GPS, etc.)	Non (Accès limité)	Oui	Oui
Téléchargement depuis les Mobile stores	Non	Oui	Oui
Codage 1 seule fois et exécution sur plusieurs plateformes	Oui	Non	Oui
Accès hors ligne	Non	Oui	Oui
Portabilité du code	Oui	Non	Oui
Coût de développement et de maintenance	Réduit	Élevé	Moyen

Tableau I.1: Fonctionnalités de chaque solution du développement mobile [Olfa, 2006]

V. Conclusion

Nous avons présenté dans ce chapitre un côté du développement d'applications mobiles ainsi que les solutions existantes dans ce domaine pour pouvoir faire le bon choix et arriver à l'application souhaitée.

Pour répondre à notre problématique, nous utilisons dans ce travail la solution "**Hybride**", qui nous offre juste ce qu'il nous faut pour développer une application à la fois pas très lourde et assez riche en fonctionnalités et multi-plateformes.

Dans le chapitre suivant nous passerons en revue les meilleurs frameworks hybrides existants à ce jour, afin de choisir celui qui nous convient le mieux.

Chapitre II : Les frameworks hybrides

I. Introduction

Parmi les trois types de solutions pour développer une application mobile présentés dans le chapitre précédent, il existe le développement hybride.

La solution hybride garantit le multi-plateforme ce qui permet un gain de temps non négligeable et une baisse des coûts importante. Généralement, ces solutions sont développées en JavaScript[4]. Parmi les frameworks des applications mobiles hybrides, nous présentons dans ce chapitre comparatif : Cordova[5], Ionic[6] et le nouveau framework react-native[7].

II. Frameworks des applications mobiles hybrides

Les applications hybrides offrent le meilleur des deux mondes native et web, elles combinent à la fois des éléments d'application web[8] et d'application native[9] afin d'utiliser les fonctionnalités natives des smartphones. Sans trop tarder nous commençons à donner des détails sur les plate-formes hybrides que nous avons déjà citées dans l'introduction.

II.1. Apache Cordova

Si vous êtes intéressés par le développement web sur Android, vous avez entendu parler de PhoneGap. Cordova (PhoneGap) est un projet réalisé avec Adobe qui a fourni le code source à la fondation Apache, un framework open source, utilisé pour le développement des applications mobiles hybrides, il utilise les standards du web ou en d'autre termes, les technologies HTML, CSS et Javascript.

[4] JavaScript : https://www.javascript.com/
[5] Cordova : https://cordova.apache.org/
[6] Ionic : http://ionicframework.com/
[7] React-native : https://facebook.github.io/react-native/
[8] Application web : https://www.definitions-marketing.com/definition/web-application-mobile/
[9] Application native : https://www.definitions-marketing.com/definition/application-native/

Figure II.1:Logo Apache CORDOVA(PhoneGap) [Vincent, 2016].

Ce framework a connu une grande évolution, et actuellement, il supporte un nombre important de systèmes d'exploitation mobiles tels que : android, iOS, WindowsPhone, BlackBerry ou encore les plate-formes Windows 8 et ubuntu.

Nous pouvons penser à Cordova comme un conteneur pour connecter notre application Web aux fonctionnalités mobiles natives. L'idée est simple: compilez un shell à l'aide d'un module **"WebView"** chargé de l'application. C'est un peu comme regarder l'application dans un navigateur très léger.

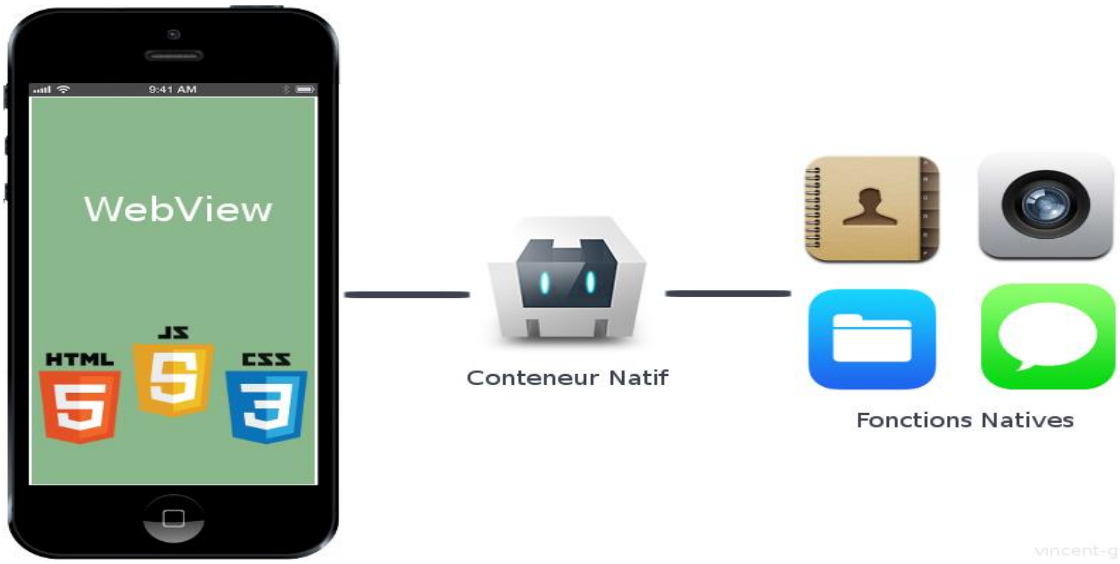


Figure II.2:Unique webView offerte par le conteneur Natif de Cordova [Vincent, 2016].

PhoneGap existe toujours et peut-être utilisé via une offre "cloud" pour générer ses applications pour Android et iOS, Windows Phone, BlackBerry...

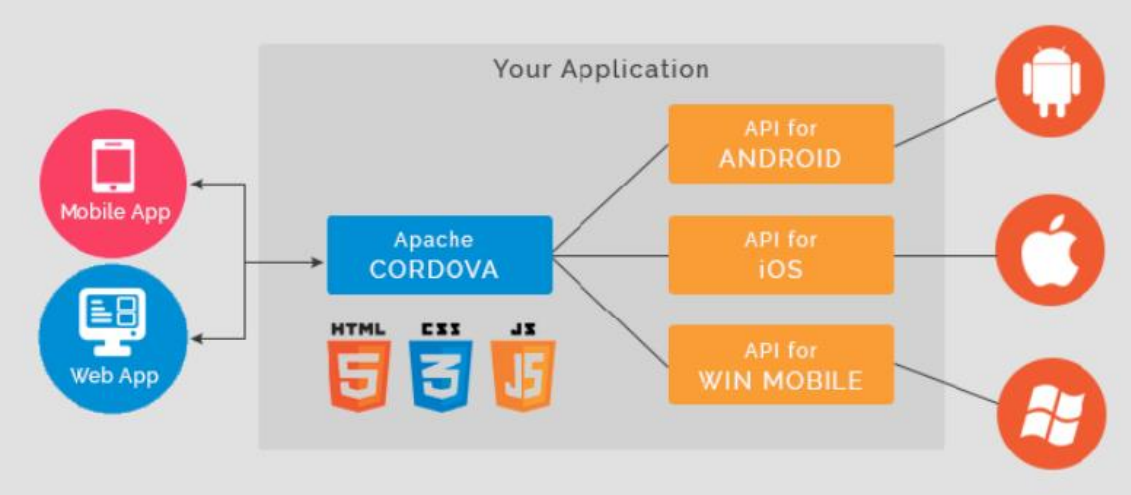


Figure II.3: Cordova et le multi-plateforme [Olfa, 2006].

Cordova propose également des plugins pour enrichir les possibilités déjà disponibles via la technologie Web.

Tous les points cités jusqu'à maintenant sont considérés comme des avantages pour Cordova, mais comme tous les frameworks, il a un inconvénient majeur, selon [Ludovic, 2012], l'inconvénient de Cordova est que toute la logique métier doit être développée en Javascript. Cordova est donc adaptée à des projets mobiles avec une faible logique métier.

Malgré cet inconvénient et aussi la faille de sécurité annoncée par IBM Security X-Force Research en 2014, Cordova reste parmi les meilleures solutions de développement cross-Platform dans le marché.

II.2. Ionic

Ionic est un framework d'applications mobiles hybride populaire, créé en 2013. Il se base sur AngularJS[10] pour la partie application web du framework et sur Cordova pour la partie applications natives.

[10]**AngularJS** : est un framework JavaScript bien connu pour la construction d'applications Web, site officiel : https://angularjs.org/ .

Figure II.4: Les bases du framework Ionic [Cihad, 2014].

Ce framework open source est utilisé pour développer des applications mobiles hybrides rapidement et facilement et qui seront déployées sur des sites web ou des applications mobiles avec différentes plate-formes telles que : android, iOS et WindowsPhone.

Figure II.5:Ionic et le multi-plateforme [Cihad, 2014].

Ionic fournit un ensemble de composants d'interface à utiliser dans des applications hybrides. Ces composants sont similaires à celles fournies par les SDK natif comme: des onglets et des boîtes de dialogue.

Une installation de système de plugin Cordova est nécessaire pour avoir des fonctionnalités matérielles supplémentaires. Ainsi il fournit une API utilisée par exemple pour demander des coordonnées GPS ou accéder à la caméra.

Les plugins se composent toujours de deux parties. La première est une partie JavaScript qui s'exécute dans le WebView qui expose une API agréable à l'application hybride. La deuxième partie est spécifique à la plate-forme et écrite dans la langue maternelle de la plate-

forme, par exemple Java pour Android et Objective-C pour iOS. Les API natives sont contrôlées par cette deuxième partie. L'image suivante illustre l'architecture de plugin de haut niveau.

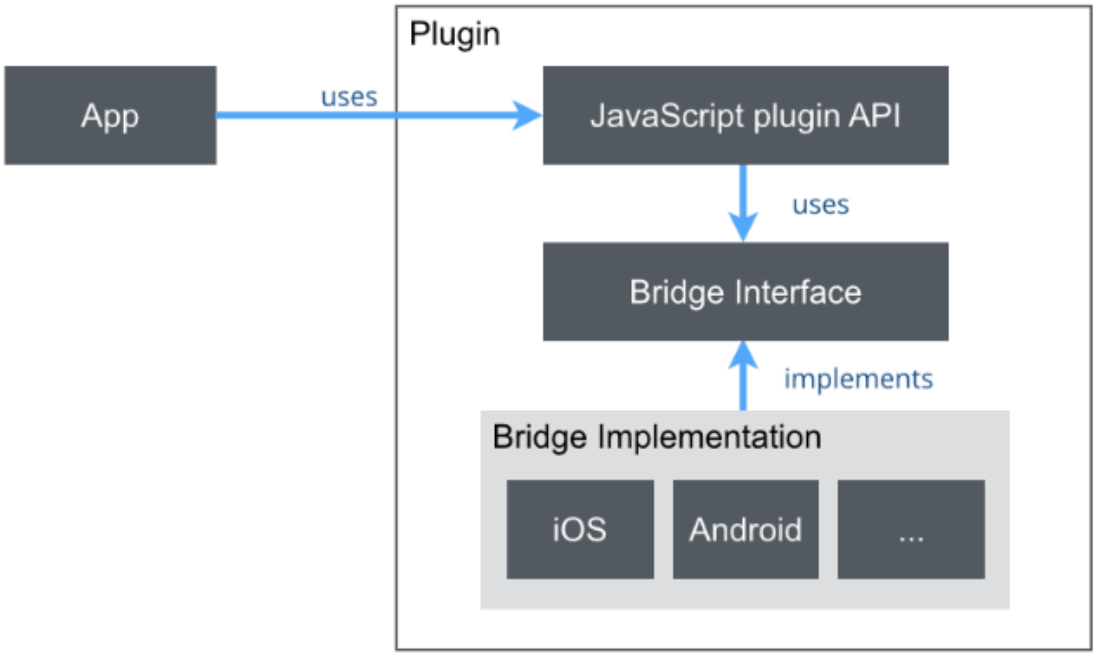


Figure II.6: Architecture de plugin Cordova de haut niveau [Ripkens, 2014].

Ionic n'est pas seulement un ensemble de composants d'interface utilisateur, mais aussi un ensemble d'outils de développement et un écosystème pour la construction rapide d'applications mobiles hybrides.

Il existe actuellement deux versions Ionic distinctes, incompatibles entre elles : la première **'version 1.3.3'** qui intègre AngularJS 1.x équipée d'un moteur JavaScript qui interprète le code JavaScript. La deuxième **'version 2.2.0'** intègre Angular 2 et un moteur de conversion TypeScript[11] vers JavaScript, cela facilite la compréhension du code. Sans doute, que Ionic 2 est plus performant que celui de la version 1.

[11] **TypeScript** : est un langage de programmation développé par Microsoft pour améliorer et sécuriser le code JavaScript.

II.3. React-native

Cela fait déjà plus d'un an que l'annonce d'une possible révolution dans le développement mobile est faite par facebook : il s'agit de React-Native ou comme certains l'appellent « **La nouvelle bombe de chez facebook** ». [MaxLab, 2015]

Comme react, react-native est un framework mobile hybride développé par Facebook en 2015, à la différence de son grand frère qui vise les interfaces web, react-native cible les applications natives. [Antoine, 2017]

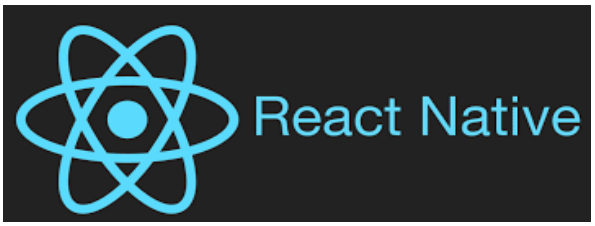


Figure II.7: Le logo de framework react-native [Rodrigo, 2016].

React-native comme son nom l'indique est basé sur react, et il repose sur la même API que son grand frère, sans offrir pour autant la même librairie JavaScript. Il est très récent sur le marché des outils multi-plateformes, il adopte la philosophie: "**Learn once, write everywhere**" [MaxLab, 2015], car il utilise le langage JavaScript ainsi que react comme couche d'abstraction sans réécrire un nouveau framework mobile, la seule différence ici est au lieu de créer des éléments de DOM[12] [TECFA, 2003], nous allons piloter des composants natifs. La figure suivante est une comparaison entre l'architecture des deux moteurs :

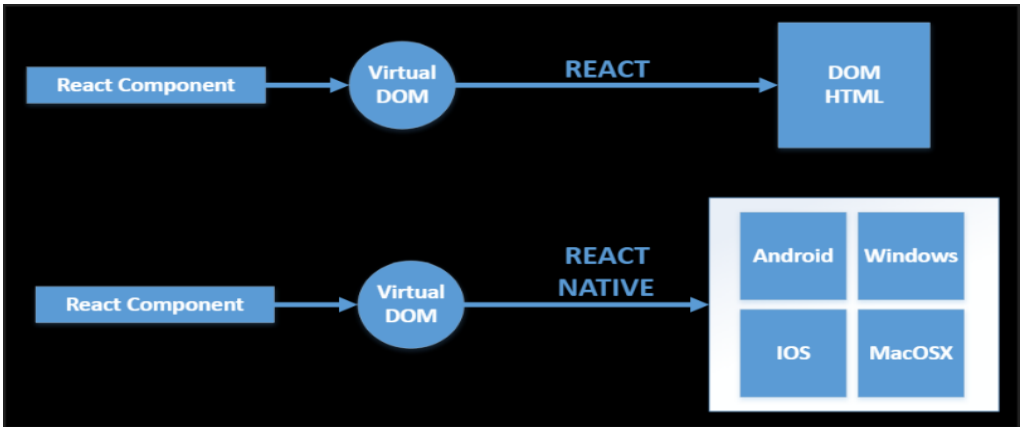


Figure II.8: Comparaison entre l'architecture des moteurs react et react-native [Rodrigo, 2016].

[12] **Dom :** le Modèle Objet de Document est une interface de programmation d'application (API) pour définir la structure logique des documents HTML valides et XML bien-formés et la manière de les manipuler.

Ce qui est intéressant ici est l'utilisation des ingrédients de base : JavaScript, des feuilles de style CSS et même des balises HTML ce qui donne la même syntaxe que dans un navigateur, seuls les composants changent.

Comme react, les composants sont le cœur du react-native, ils s'intègrent bien dans toute application react-native et peuvent être utilisés avec du code JS. Chaque composant graphique créé possède un template, des évènements, des paramètres et un état, de manière complètement indépendante aux autres composants. Chaque composant peut avoir des sous composants ainsi de suite, pour obtenir à la fin un seul composant global qui englobe tous les sous composants qui lui appartiennent.

Parmi les avantages de ces composants, c'est qu'ils sont : réutilisables, testables, maintenables [Sandrine, 2017].

Et dans la figure suivante nous présentons quelques exemples de composants et éléments react-native :

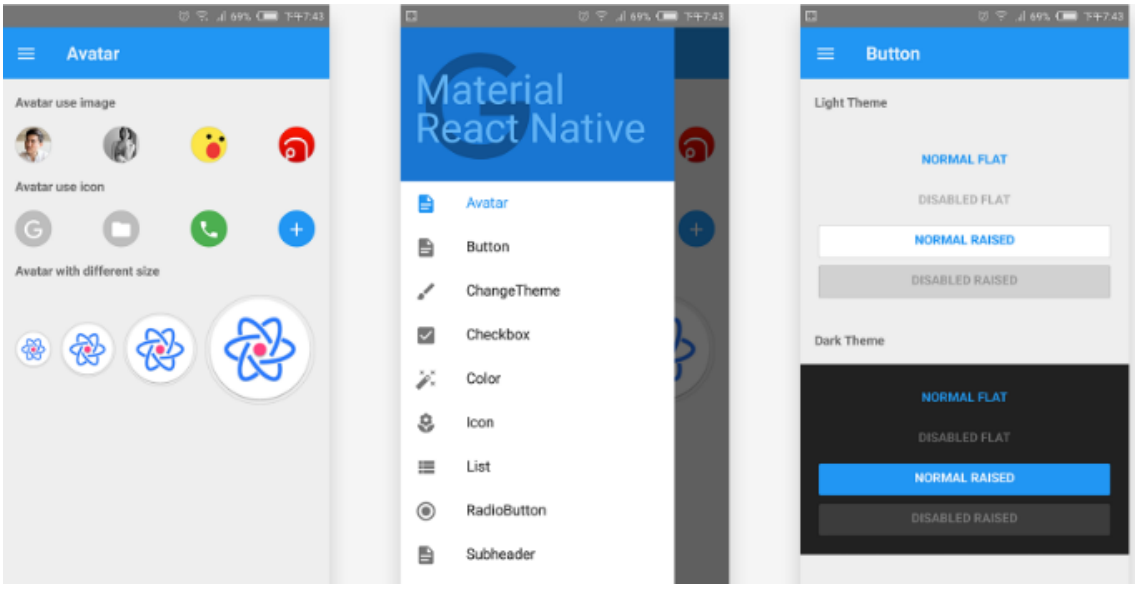


Figure II.9: Exemple d'applications avec les composants react-native [MRN, 2017].

Dans un premier temps, react-native est utilisé pour créer des applications mobiles compatibles avec les deux plate-formes Android et iOS, et pas très longtemps d'aujourd'hui exactement avril 2017 il intègre la troisième plate-forme qui est WindowsPhone.

De plus en plus react-native devient célèbre, sa popularité n'a cessé de progresser ces derniers mois, car il présente beaucoup de points forts ce qui le rend spécial, les dernières

statistiques montrent que **78%** des développeurs s'intéressent à l'apprentissage de cette nouvelle technologie comme le montre diagramme à secteur suivant :

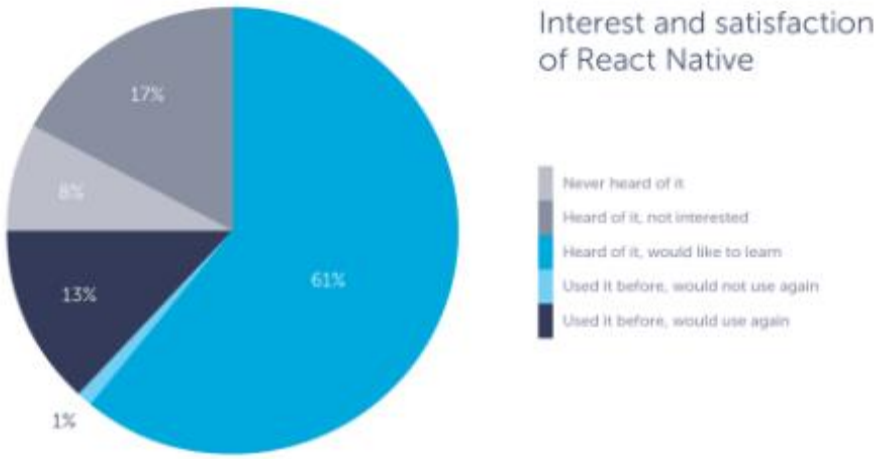


Figure II.10: Diagramme à secteur d'Intérêt et de la satisfaction de react-native [Robert, 2016].

III. Choix du framework

Il existe plusieurs solutions hybrides multi-plateformes, comme Cordova, Ionic et react-native, la question qui se pose toujours dans le développement mobile est : quel framework choisir pour notre future application ?

Pour répondre à cette question nous faisons une comparaison entre ces différents frameworks.

Cordova et d'autres types comme Ionic, permettent la création des applications mobiles avec un accès aux API natives (l'accéléromètre, caméra...). Côté interface graphique, ils sont conçus autour d'une WebView qui est affichée en pleine écran, ce qui veut dire que leurs interfaces sont basées sur des technologies web classiques HTML/CSS/JS. Pour plus de fonctionnalités, il est possible d'étendre l'API du navigateur en installant des plugins afin d'enrichir le moteur JavaScript. Mais le problème est que les performances de ses applications seront donc équivalentes à une application web, et non une application mobile.

Avec react-native, il est obligatoire de composer chaque interface à l'aide de composants React, ces derniers font appel au layout natif de chaque plateforme.

Les performances de l'UI sont donc quasi similaires aux performances natives. [Zoontek, 2016]

Pour y voir plus clair, nous avons résumé ici les possibilités et les limites de ces différentes technologies de développement :

	React Native	PhoneGap	Ionic
Langage	JS	HTML/CSS/JS	HTML/CSS/JS
Multiple déploiement	Oui	Oui	Oui
Moteur de template	React	Au choix	Angular
Déployable sur le web	Oui	Oui	Oui
Bridge natif	Oui	Non	Non
Utilise une webview	Non	Oui	Oui
Mise en page facile	Oui	Oui	Oui

Tableau 2.1: Les caractéristiques des trois frameworks (React-native, cordova, Ionic) [Sandrine, 2017].

De plus, il offre un cadre multi-plateforme pour l'avenir qui dépasse l'api iOS et Android, car il couvrira selon les prévisions même les téléviseurs intelligents, consoles Xbox, périphériques Windows et la liste reste longue, il s'appuie sur une communauté active et il a une interface ergonomique.

Pour terminer, vous l'avez compris, notre choix se tourne vers react-native qui est une technologie très prometteuse avec un énorme potentiel, et qui a déjà été appréciée par des géants comme Baidu, Discovery, Instagram et d'autres, elle est à deux pas de devenir une alternative à part entière à Objective C, Swift et Java, en particulier sur des projets qui ne sont pas critiques.

IV. Conclusion

React-native est un bon compromis entre performance, légèretés et multi-plateforme et vu que les utilisateurs de notre application finale vont l'utiliser pour créer une application avec ces caractéristiques alors nous avons jugé que ce framework est le plus adéquat dans ce travail.

De plus react-native se base sur une programmation par composants et ça tombe bien car cela nous facilitera la tâche pour manipuler nos modèles en utilisant l'ingénierie dirigée par

les modèles que nous verrons dans le chapitre suivant, car nous aurons juste à ajouter des composants les uns avec les autres afin de monter notre application tel un jeu de puzzle.

Chapitre III : L'approche de l'ingénierie dirigé par les modèles MDA

I. Introduction

Pour bien manipuler les composants react-native nous utiliserons l'approche d'ingénierie dirigée par les modèles MDA, que nous allons présenter dans ce chapitre.

Après l'approche objet des années 80, l'ingénierie logicielle s'oriente aujourd'hui vers l'ingénierie dirigée par les modèles (IDM).

Selon [Elkouhen, 2015], l'IDM offre un cadre méthodologique et technologique permettant d'unifier et de favoriser en un processus homogène l'étude des différents aspects du système.

II. Généralité sur l'IDM

L'IDM peut être vu comme une famille d'approches qui se développent à la fois dans les laboratoires de recherche et chez les industriels impliqués dans les grands projets de développement logiciel [Estublier 2006], elle se base sur le principe «tout est modèle».

Un modèle est un ensemble de faits caractérisant un aspect d'un système dans un objectif donné, il représente donc un système selon un certain point de vue, à un niveau d'abstraction facilitant la compréhension du système [Elkouhen, 2015]. Ces abstractions permettent de concevoir des applications indépendantes des plate-formes cibles.

La notion de modèle dans l'IDM fait explicitement référence à la définition des langages utilisés pour les construire. La définition d'un langage de modélisation prend la forme d'un modèle, appelé méta-modèle [Elkouhen, 2015].

Un méta-modèle est un modèle qui définit le langage d'expression d'un modèle [OMG, 2006], ou en d'autre terme, un méta-modèle est un modèle d'un ensemble de modèles.

Après l'acceptation du concept clé de méta-modèle comme langage de description de modèle, de nombreux méta-modèles ont émergés afin d'apporter chacun leurs spécificités dans un domaine particulier.

Devant le danger de voir émerger indépendamment et de manière incompatible cette grande variété de méta-modèles, il y avait un besoin urgent de donner un cadre général pour leur description. La réponse logique fut donc d'offrir un langage de définition de méta-modèles qui prit lui-même la forme d'un modèle : ce fut le méta-méta-Modèle MOF (Meta-Object Facility) [OMG, 2006]. Ce dernier doit avoir la capacité de se décrire lui-même, cela pour éviter d'avoir un autre modèle et limiter le nombre de niveau d'abstraction.

C'est sur ces principes que se base l'organisation de la modélisation de l'OMG généralement décrite sous une forme pyramidale représentée par la figure III.1 [Bézivin, 2003].

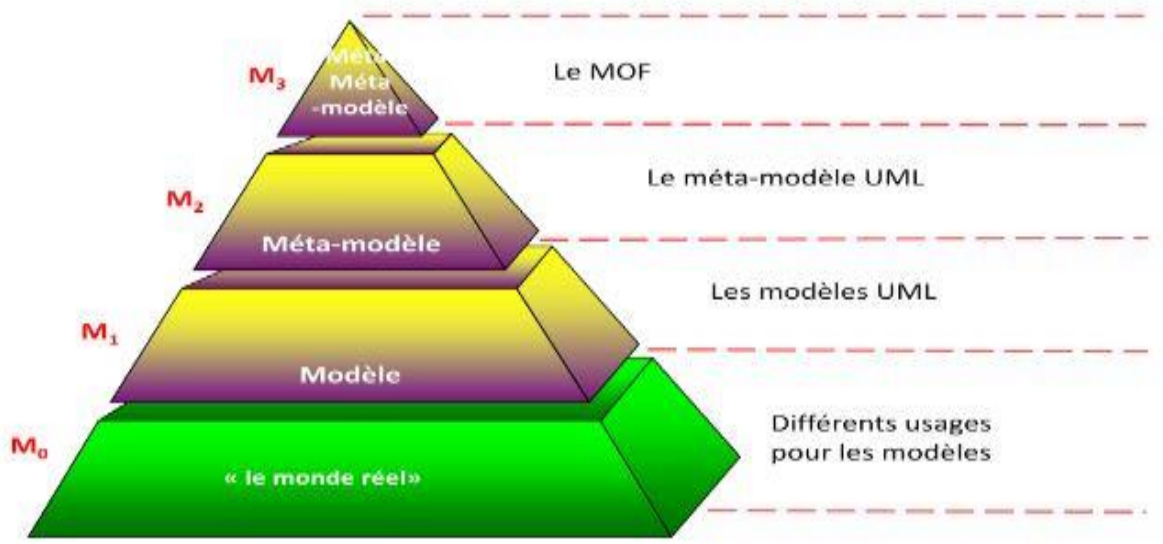


Figure III.1: **Niveaux d'abstraction de l'IDM [OMG, 2006].**

Les niveaux d'abstractions sont résumés dans les points suivants:

1. Le niveau M0 ou monde réel est représenté à la base de pyramide.
2. Le niveau M1 ou la couche modèle dans laquelle nous pouvons définir les modèles en utilisant les langages de modélisation.
3. Le niveau M2 ou la couche de méta-modélisation où les méta-modèles sont définis en utilisant les langages de méta-modélisation.
4. Le niveau M3 ou la couche de méta-méta-Modèle où seul les langages de méta-modélisation sont auto-définis et décris.

L'ingénierie dirigée par les modèles est donc un paradigme dans lequel la modélisation est considérée comme l'élément central d'un logiciel.

III. L'approche de l'ingénierie dirigée par les modèles MDA

L'OMG a défini l'approche MDA (Model Driven Architecture) en 2000 [Soley, 2000] pour exploiter pleinement les avantages des modèles. Le MDA inclut pour cela la définition de plusieurs standards, notamment UML, MOF et XMI. [Elkouhen, 2015]

Le MDA (Model Driven Architecture) de l'OMG est une approche typique d'ingénierie dirigée par les modèles pour la conception d'application, elle est basée sur le standard UML pour la définition des modèles et l'environnement de méta-modélisation MOF (Meta Object Facility) pour la programmation au niveau modèle et la génération de codes.

III.1. Les modèles MDA

1. CIM (Computation Independent Model)

Les modèles d'exigence CIM décrivent les besoins fonctionnels de l'application, dans lequel nous devons représenter l'application dans son environnement afin de définir les services qu'elle offre et les entités avec lesquelles elle interagit. Les CIM peuvent servir de référence pour s'assurer que l'application finie correspond aux demandes des clients.

2. PIM (Platform Independent Model)

Les modèles d'analyse et de conception de l'application PIM représentent, indépendamment de toute plate-forme technique, une spécification neutre d'un système (modèle de métier et de service) qui ignore tous les détails de mise en œuvre et d'utilisation du système décrit.

3. PDM (Platform Description Model)

PDM (Platform Description Model) Il définit les différentes fonctionnalités de la plate-forme sur laquelle l'application va s'exécuter et précise comment les utiliser.

4. PSM (Platform Specific Model)

Les modèles de code PSM sont des modèles de métier et de service liés à une ou plusieurs plate-formes d'exécution.

5. Code source

Le code source est le résultat final de tout le processus MDA, il est généré automatiquement à partir du modèle PSM. Un code source d'une application mobile est compatible avec une ou plusieurs plate-formes d'exécution (iOS, android, WindowsPhone...).

III.2. Les transformations des modèles MDA

Les transformations sont au cœur de l'approche MDA. Elles permettent convertir un modèle en un autre modèle et d'obtenir différentes vues de ce dernier, de le raffiner ou de l'abstraire, de plus elles permettent de passer d'un langage vers un autre. Elle assure le passage d'un ou plusieurs modèles d'un niveau d'abstraction donné vers un ou plusieurs autres modèles du même niveau (transformation horizontale) ou d'un niveau différent (transformation verticale). Les transformations horizontales sont de type PIM vers PIM ou bien PSM vers PSM. Les transformations verticales sont de type PIM vers PSM ou bien PSM vers code. Les transformations inverses verticales (rétro-ingénierie) sont type PSM vers PIM ou bien code vers PSM. [Fourati, 2010]

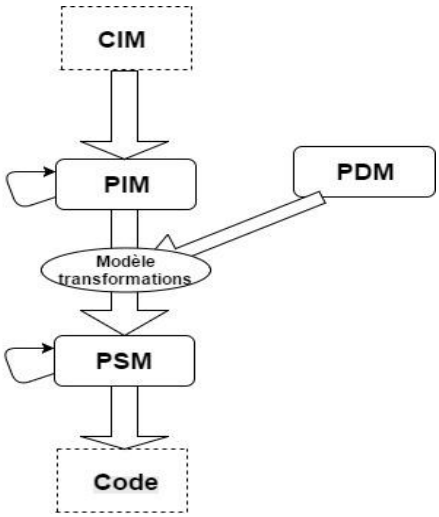


Figure III.2: Le processus de MDA en Y [Elkouhen, 2015].

Tout d'abord, nous trouvons le modèle CIM (Computation Independent Model), nous sommes exactement dans le niveau **M_0 [Figure III.1]** ou le monde réel qui résume les besoins de l'utilisateur et les fonctionnalités du système.

La suite du processus est la construction des modèles PIM (Platform Independent Model) partielles à partir des modèles CIM, cela permet de véhiculer l'information contenue dans les

CIM vers les modèles PIM. Ce dernier sera transformé [PIM→ PIM] pour enrichir les PIM partielles en ajoutant des informations et en spécifiant leur contenu.

Le modèle PSM (Platform Specific Model) est obtenu d'une transformation de modèle PIM. Il faut noter qu'un modèle PIM doit persister à l'apparition de nouveaux PSM.

Dans le cas où le modèle d'analyse et de conception (PIM), ainsi que le modèle de code (PSM) représentent le système, nous sommes exactement dans le niveau **M_1 [Figure III.1]** de l'architecture de l'approche MDA.

Remarque

Dernière étape est la transformation du modèle PSM vers un code source, cette étape n'est pas à proprement dit considérée comme une transformation par l'approche MDA car c'est une transformation modèle→ texte, nous pouvons plutôt la considérer comme une étape de retranscription textuelle du modèle PSM.

Le code source obtenu est compatible avec une ou plusieurs plate-formes d'exécution.

IV. Les objectifs de l'approche MDA

L'approche MDA de l'OMG a été définie pour répondre aux problèmes liés à l'évolution continue des technologies, parmi les objectifs de cette approche :

- **Les modèles pérennes:**

L'objectif majeur du MDA est l'élaboration des modèles pérennes (PIM), ces derniers doivent survivre le changement de plate-forme et être réutilisable pour plusieurs plate-formes.

Cette pérennité est garantie grâce à l'utilisation d'UML qui est un standard stable mais aussi grâce aux modèles qui sont par nature des entités pérennes.

- **Les gains de productivité :**

Les modèles sont la base du processus MDA, ils facilitent la communication entre experts du domaine et développeurs. Dans cette approche les modèles sont considérés comme un outil de production à l'aide de l'automatisation des transformations de modèles.

La génération de la totalité de modèles de code (PSM) devient alors automatique, à cause des modèles qui sont considérés comme des éléments qui véhiculent une information précise est nécessaire pour cette transformation. Le mappage automatique de la PSM permet d'enlever le risque d'erreur, il produit ainsi un gain de coût à la transformation de la PIM, c'est le gain de productivité.

- **La prise en compte des plate-formes d'exécution:**

La MDA permet d'intégrer les spécifications des différentes plate-formes[13] d'exécution pendant la transformation du modèle PIM vers PSM**.** Et de cette manière le développement des applications mobiles multi-plateformes est devenu facile, puisque tout est basé sur un modèle PIM qui est indépendant des détails techniques des plate-formes d'exécution (J2EE, .Net, PHP, etc.)

Il faut noter que toutes modifications dans le modèle PIM, conduit à un changement dans les fonctionnalités du système livré.

En plus des points cités, l'approche MDA a pour objectifs :

- **Évolution d'applications existantes.**
- **Maîtriser l'impact des nouvelles technologies.**

V. Conclusion

Dans ce chapitre nous avons détaillé l'approche d'ingénierie dirigée par les modèles MDA, mais ce n'est pas la seule approche de génie logiciel qui se base sur les modèles dans son processus de développement.

[13] **Plate-forme:** est un environnement permettant la gestion et/ou l'utilisation de services applicatifs, il peut être un système d'exploitation, un environnement d'exécution, ou encore un environnement de développement.

Chapitre IV : Atelier graphique de modélisation d'applications mobiles

I. Introduction

Le travail dans notre sujet à pour objectif de manipuler les différents outils : Xtext et Sirius ainsi que de comprendre la logique de la nouvelle technologie react-native, afin de développer notre propre générateur de code source final, qui génère du code react-native compatible avec trois plate-formes : iOS, android, WindowsPhone.

Dans ce chapitre, nous détaillons étape par étape la démarche de travail. Notre solution comporte trois parties, dont la première consiste à créer le modèle PIM indépendant de toutes plate-formes d'exécutions, en se basant sur une grammaire DSL développée avec l'outil Xtext.

La deuxième partie est la manipulation des modèles, dans cette étape nous utilisons l'outil Sirius pour la création d'un éditeur en se basant sur la représentation du DSL au sein du modèle EMF (Eclipse Modeling Framework).

La dernière partie est la génération du code de l'application, et en raison de l'absence d'un générateur de code react-native, nous développons une application java que nous nommons "MyAcceleo" pour faire la retranscription textuelle du modèle PSM.

Le code react-native obtenu sera exécuté via des lignes de commandes dans un fichier de format ".sh".

Le schéma suivant représente une vue globale de la solution proposée :

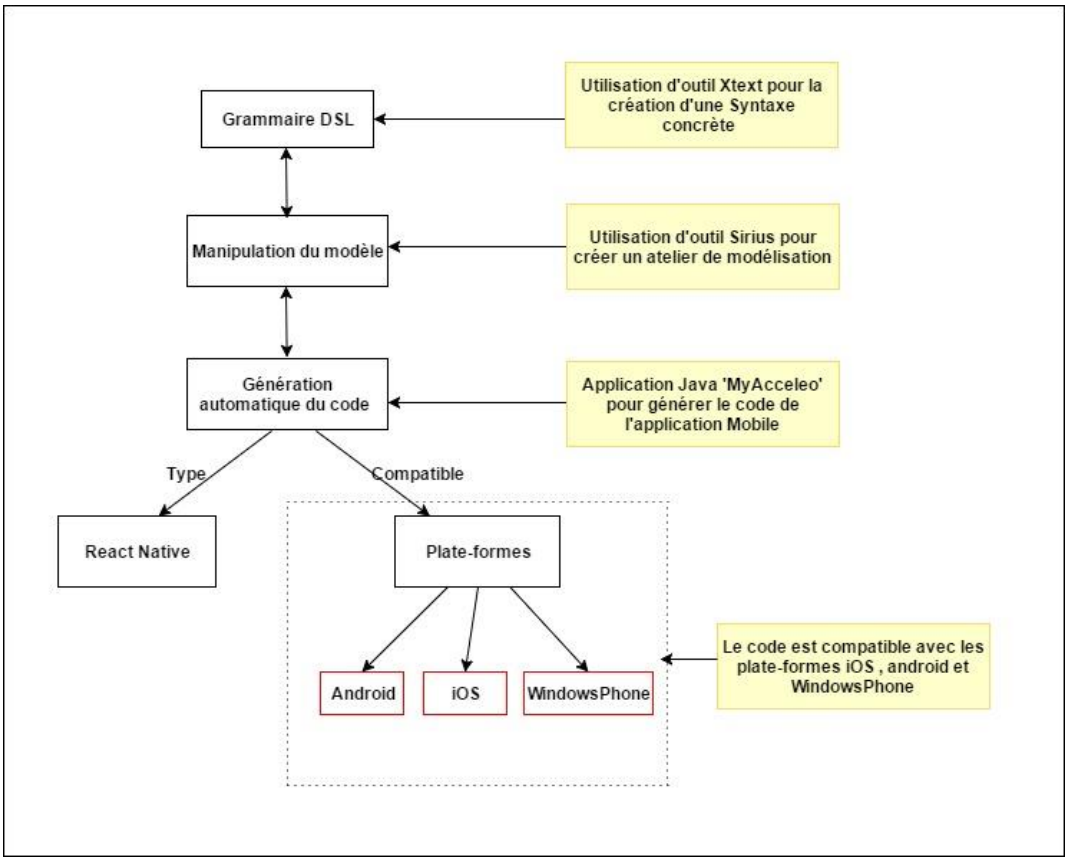


Figure IV.1: Vue de l'ensemble de la solution.

II. Détails de la solution proposée

II.1. Création du méta-modèle

Pratiquement, et comme déjà présenté dans le chapitre précédant, un modèle est une représentation, une image de la réalité. Toute réalité complexe a besoin d'être représentée pour être comprise et maîtrisée, il est supposé fournir une représentation des faits que l'on désire observer, excluant les faits annexes qui n'apparaîtront pas nécessaires.
Le résultat de cette partie est d'obtenir un méta-modèle qui nous permet de créer nos modèles PIM.

La première étape du processus consiste à établir le vocabulaire du (DSL) en utilisant le framework **Xtext.** Nous sommes positionnés exactement dans le niveau 1 de la solution comme le montre la Figure **IV.2** :

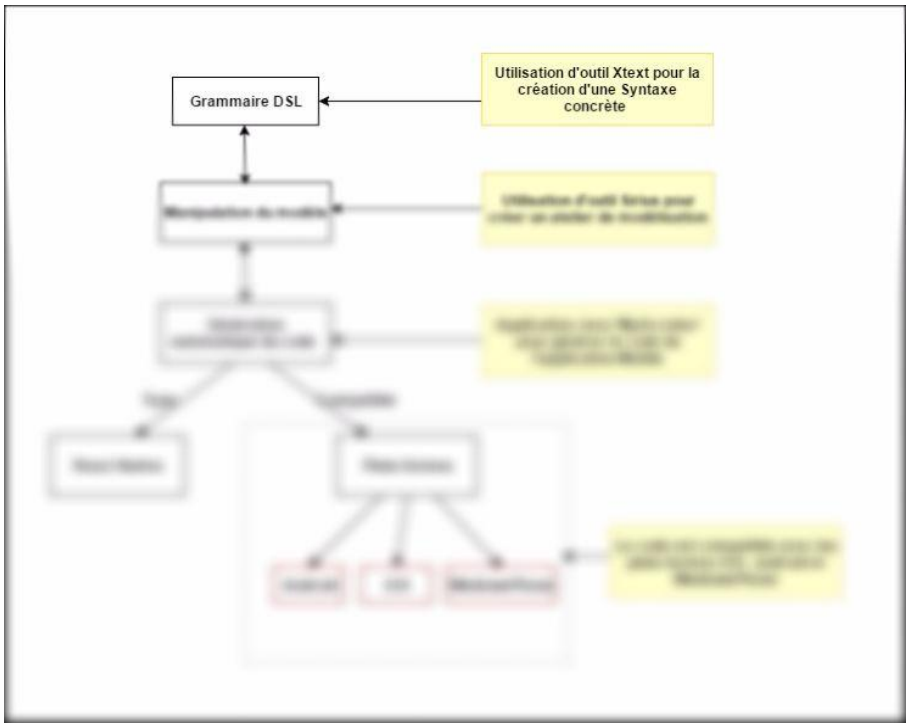


Figure IV.2: Niveau 1 de la solution (partie Xtext).

Le vocabulaire du DSL doit être simple et compréhensible pour garantir le plus grand respect des besoins des clients.

Dans un second temps, il faut définir le méta-modèle en spécifiant les règles syntaxiques de notre DSL qui vont suivre le vocabulaire déterminé dans l'étape précédente. Il est possible de représenter le DSL sous format graphique mais dans notre solution nous avons choisi l'utilisation d'une grammaire textuelle.

Notre grammaire Xtext est développée en trois grandes parties: **"Modèle"**, **"Vue"**, **"Contrôleur"**, à savoir le pattern **MVC**, l'utilisation de ce dernier permet de concevoir des applications de manière claire et efficace grâce à la séparation des intentions.

La figure **IV.3** est le diagramme de classe qui donne une vue globale de notre DSL :

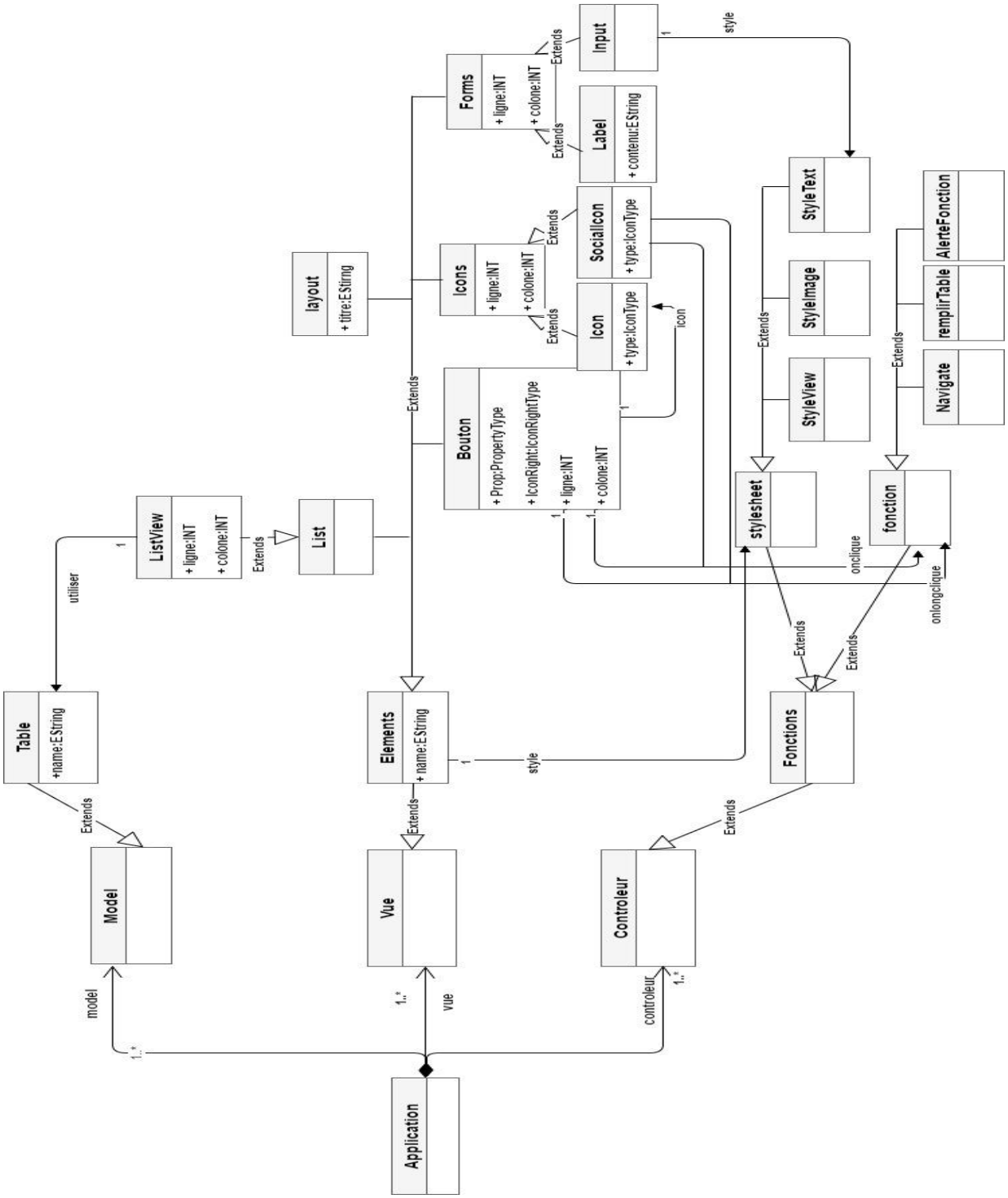


Figure IV.3: Diagramme de classe pour la syntaxe concrète DSL.

Expliquons un petit peu notre diagramme de classe :

- ❑ **"Application"** joue le rôle de la racine, elle est composée d'un ou plusieurs Modèles, vues ou contrôleurs.

```
Application:
    'application' name=STRING '=>' packageName=STRING
    'FirstLayout' ':'  (FirstLayout+=[Layout] )?
    '{'
        (model+=Model)*
        (vue+=Vue)*

        (controleur+=Controleur)+

     '}'
;
```

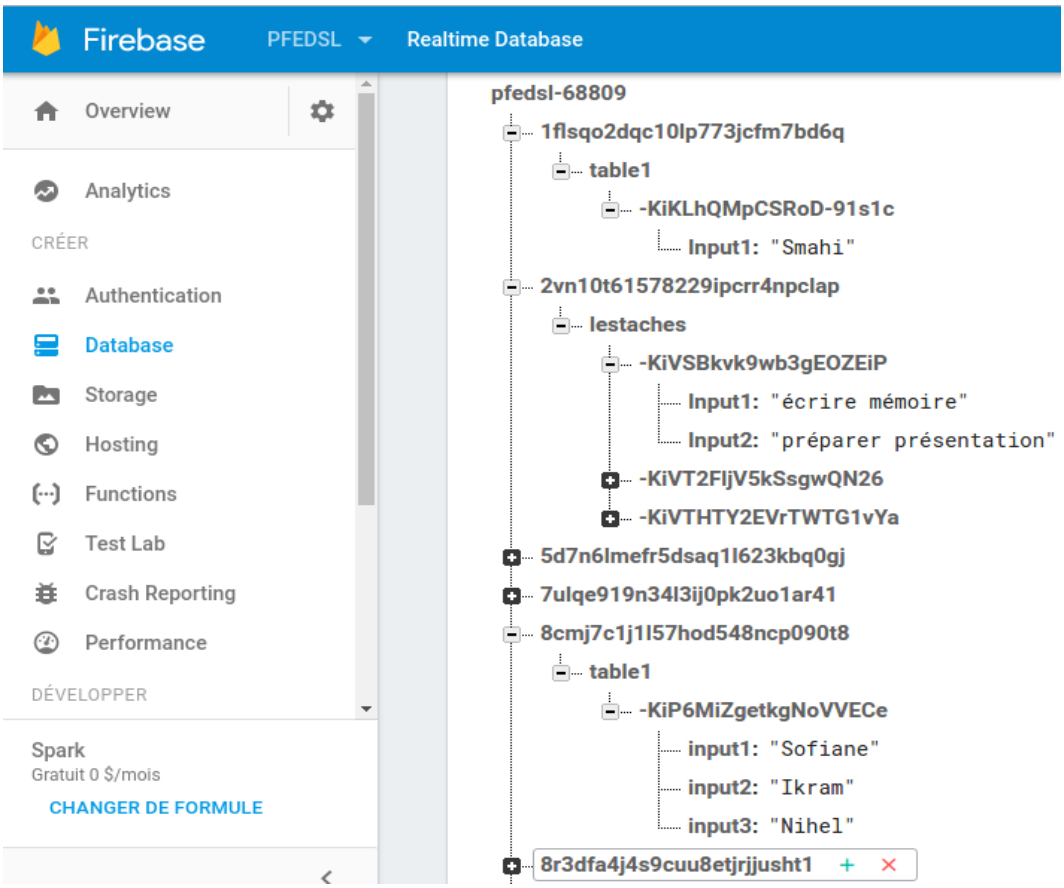
Capture IV.1: La racine de la grammaire Xtext.

- ❑ Le modèle représente la partie métier, elle contient les données à afficher. Dans notre solution, cette partie comporte une seule règle [Capture IV.2] qui permet de créer un élément **"Table"**, les champs de chaque table font référence à un input afin de récupérer les informations introduites par l'utilisateur.

```
Model: 'Le model : {'
    Table
'}';

Table returns TableDefinition:
    'Table' name=ID
('champ '   ':'   champ+=[Input](',' champ+=[Input])* ';' ) 'End.'   ;
```

Capture IV.2: Le modèle de la grammaire Xtext.

"La base de données" n'est pas déclarée dans notre grammaire en raison de l'utilisation du service d'hébergement des bases de données distantes supporté par le framework react-native : **"Firebase"**. Ce service nous a permis de créer une base de données centralisée à distance et à chaque création d'une nouvelle application nous lui affectons un item, ça sera en quelque sorte une base de données dans une base de données.



Capture IV.3 : Architecture firebase.

- La vue regroupe les éléments qui seront utilisés dans les interfaces d'une future application. Parmi ces éléments nous trouvons : les boutons, les listViews, les inputs, les layouts etc.., ainsi que des éléments supportés par react-native comme : les icones.

```
Elements:
    Bouton|Layout|Image|
    ContainerFonctions| ContainerStyle|
    lists |Icons|Forms
    ;
```

Capture IV.4 : La vue de la grammaire Xtext (Éléments).

- Dans le contrôleur nous définissons tout ce qui est fonction. Les éléments de la vue et les données de Modèle sont reliés avec les actions désignées dans le contrôleur. Dans notre contrôleur nous déclarons les fonctions (remplirTable, alerteFonction, naviguer) et le styleSheet (styleImage, styleText, styleView).

```
stylesheet:
    StyleView |StyleText|StyleImage
;

fonction:
    AlerteFonction|RemplirTable |Navigate
;
```

Capture IV.5 : Le contrôleur de la grammaire Xtext (Éléments).

Pour le styleSheet, nous avons défini les 3 types de styles existant dans react-native qui sont : StyleView, StyleText, StyleImage, afin de donner la possibilité aux utilisateurs de personnaliser le style des futures applications. Tous les composants principaux dans une application react-native acceptent un support nommé "**style**" qui sera rempli en utilisant le langage CSS. Les styles dans react-native se basent sur le concept d'héritage et la figure suivante montre l'architecture globale des styles dans react-native :

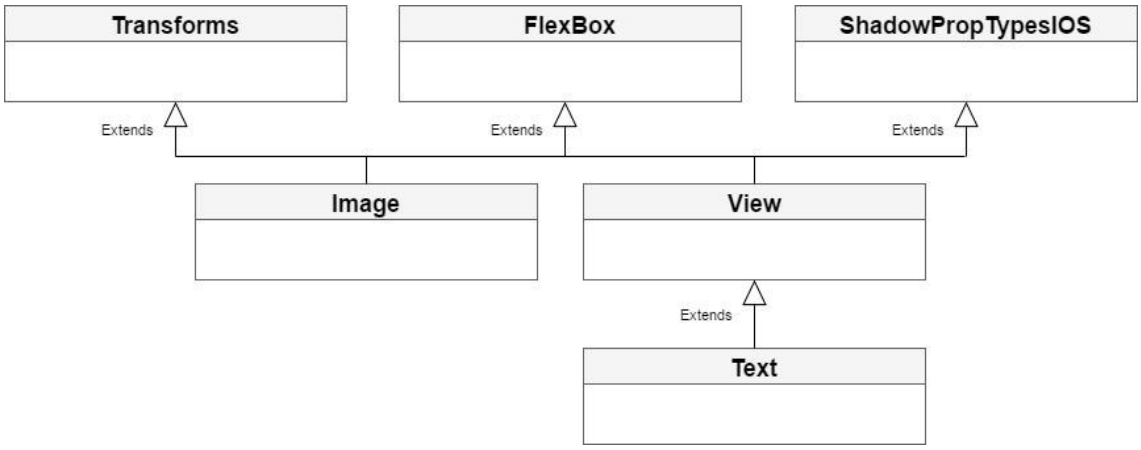


Figure IV.4: L'architecture des styles dans react-native.

Concernant les fonctions, nous détaillons le rôle de chacune dans les points suivants :

- **RemplirTable** : comme son nom l'indique, cette fonction est utilisée pour stocker l'information dans une table de base de données, elle relit la vue avec le modèle, ou en d'autres termes, elle récupère le contenu d'un input pour l'enregistrer dans une table précise, cette fonction prend deux paramètres, le premier pour donner un nom à la fonction et le deuxième pour référencer la table dans laquelle les informations seront stockées.

```
RemplirTable:
 'RemplirTable' name=ID ':'
('use '  ':'   utiliser+=[TableDefinition] (',' utiliser+=[TableDefinition])* ';') 'End.';
```

Capture IV.6: La fonction remplirTable.

➢ **Naviguer** : est la fonction qui permet de naviguer entre les différents layouts de l'application, Il suffit juste de donner la référence vers le layout où il faut aller.

```
Navigate:
    'Navigate' name=ID ':'
    ('Reference'   ':'    ref+=[Layout] (',' ref+=[Layout])*  ';')'End.'
;
```

Capture IV.7: La fonction Navigate.

➢ **AlerteFonction** : la fonction alerte est une fonction simple pour lancer un message d'alerte.

```
AlerteFonction:
    'AlertFunction' name=ID ':'
    ('Message'   ':'   message = ID   ';') 'End.'
   ;
```

Capture IV.8 : La fonction Alerte.

Les détails d'utilisation de l'outil Xtext ainsi que la création d'une grammaire DSL sont présentés dans [annexe A].

Après avoir terminé toutes les étapes de la création de la grammaire DSL, nous pouvons donc exécuter le projet Xtext pour obtenir le méta-modèle construit à partir de la syntaxe concrète de DSL qui se trouve dans le fichier « model/generated/Pfe.ecore » du projet, nous disposons aussi d'un analyseur syntaxique pour notre DSL, ainsi que d'un éditeur avec coloration syntaxique [14]et complétion automatique.

A partir de ce moment là, nous pouvons créer des instances du notre méta-modèle et les manipuler en utilisant l'outil Sirius, ces instances sont les modèles PIM.

[14] **Coloration syntaxique :** est une fonctionnalité proposée par certains éditeurs de texte, consiste à formater automatiquement les éléments du texte affiché en utilisant une couleur et un format.

II.2. Manipulation du modèle

Dans notre travail nous donnons la main aux utilisateurs de créer leurs propres représentations de leurs futures applications mobiles.

Le projet Eclipse Sirius facilite cette tâche en se basant sur la représentation du DSL au sein d'un modèle EMF (Eclipse Modeling Framework). Il permet la spécification des éditeurs de manière totalement graphique, et visualiser le rendu en temps réel.

Nous travaillerons dans cette partie dans une instance d'eclipse qui se base sur les plugins générés dans l'étape précédente. Nous sommes positionnés exactement dans le niveau 2 de la solution comme le montre la figure suivante :

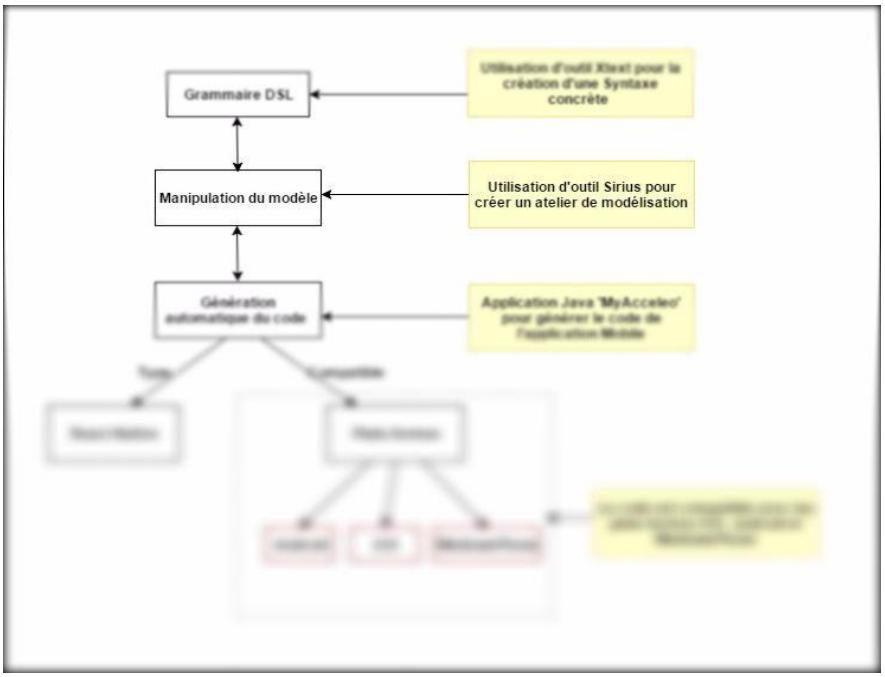


Figure IV.5: Niveau 2 de la solution (partie Sirius).

III.3. Instanciation du modèle

Pour manipuler le modèle indépendant de toutes plate-formes d'exécution le PIM nous allons suivre quelques étapes :

En utilisant l'outil Sirius, la première des choses est d'instancier le modèle en créant un projet de type « **Modeling** », le projet créé contiendra un fichier ".aird" qui permettra à Sirius

de stocker l'agencement des éléments dans l'éditeur. Ensuite nous créerons une instance de notre modèle, cette dernière est un fichier portant l'extension de notre langage : ".pfe", et qui contiendra les éléments du modèle.

Dans un second temps, nous initions notre éditeur à travers un projet que nous nommons « Viewpoint Specification Project ». Ce projet contiendra un fichier avec l'extension '.odesign' dans lequel nous allons définir nos éléments. [Annexe B]

III.4. Création des éléments

Comme déjà mentionné, ce modeste travail propose un atelier avec lequel l'utilisateur peut modéliser une future application mobile facilement, en créant des vues (Layout) et en spécifiant les différents éléments qui les composent comme par exemple : ListViews, Input etc..., ainsi que les relations qui existent entre ces éléments, nous pouvons même naviguer entre les vues. La figure suivante correspond au diagramme de cas d'utilisation de notre atelier Sirius :

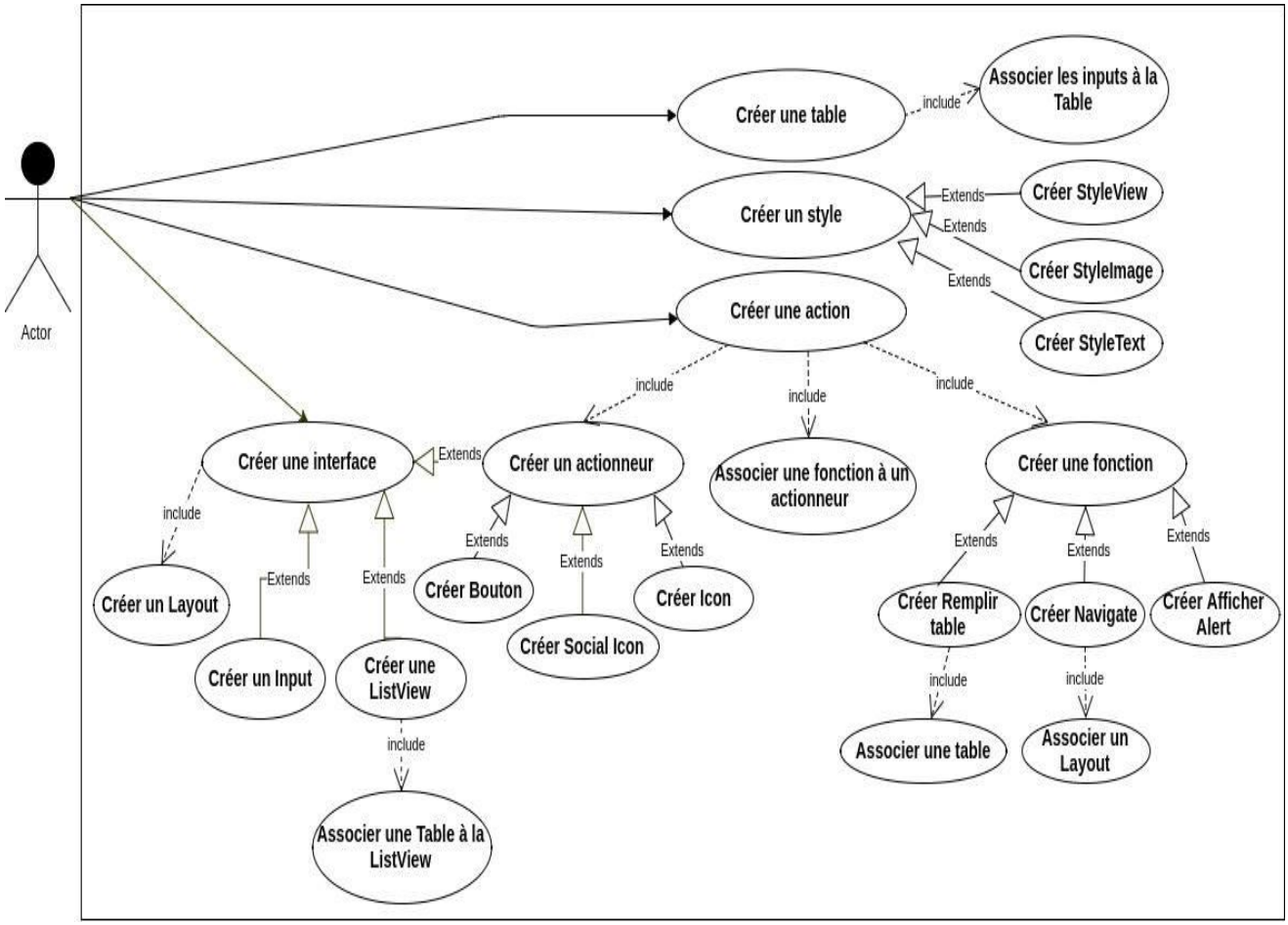


Figure IV.6: Diagramme de cas d'utilisation pour le modèle Sirius.

Description de diagramme de cas d'utilisation :

Ce diagramme se constitue d'un acteur, des cas d'utilisation et du périmètre du système. L'acteur est l'utilisateur de notre application Sirius et le périmètre du système est défini par le rectangle.

Les cas d'utilisation :

- ***Créer une interface*** : l'utilisateur doit créer un layout et il peut lui introduire des actionneurs, des inputs et des listViews afin d'obtenir une interface.

- ***Créer une listView*** : nous devons associer à notre listView une table, pour que notre listView pourra afficher les informations de cette table.

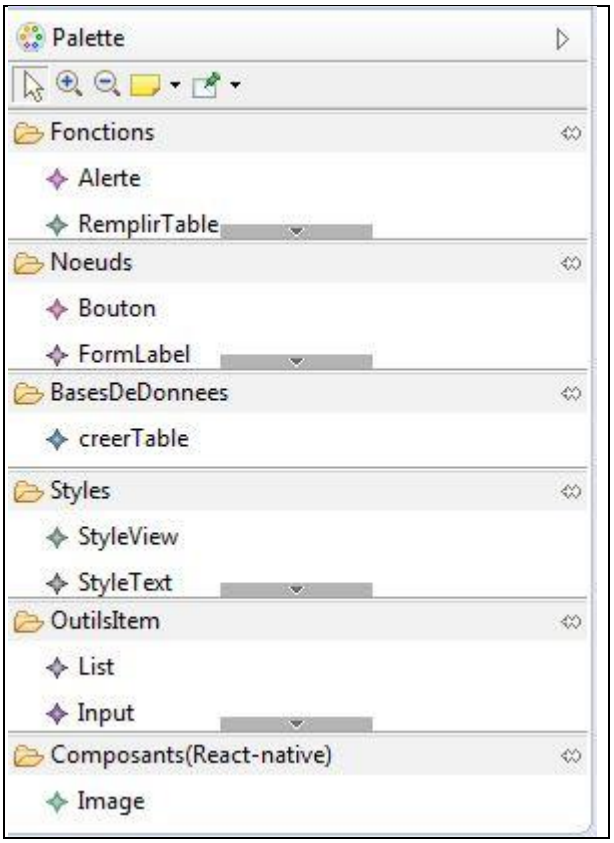
- ***Créer une action*** : ce cas d'utilisation permet de créer une action, l'utilisateur doit créer une fonction, ensuite il doit créer un actionneur et lui associer la fonction créée.

- ***Créer une fonction*** : il existe plusieurs types de fonction : afficher alerte, créer Navigate que nous devons lui associer le layout de destination. Et la fonction "remplirTable" qui reçoit en paramètre la table à remplir.

- ***Créer des styles*** : l'utilisateur peut créer des styles afin de les associer par la suite aux éléments de la vue (StyleImage pour les images, StyleText pour tous les éléments de type text, StyleView pour les éléments de type view).

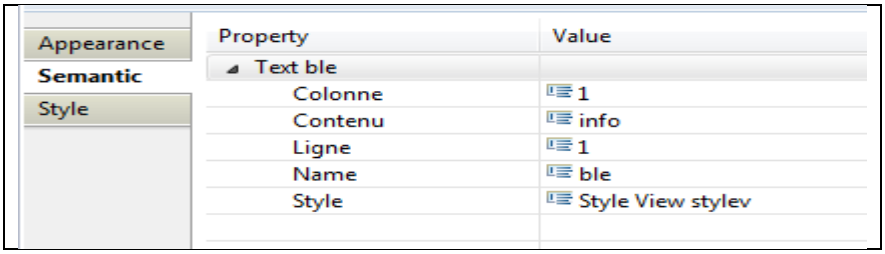
- ***Créer une table*** : L'utilisateur peut créer des tables dans lesquelles nous allons enregistrer des données, pour ce faire nous devons associer des inputs à ces tables lors de leurs créations.

Les éléments, les fonctions et les styles sont créés à partir de la palette d'outils comme le montre la figure suivante :



Capture IV.9: La palette d'outils.

Pour chaque élément créé dans Sirius nous devons spécifier ses propriétés. La capture ci-dessous, est un exemple des propriétés d'un label :



Capture IV.10: Exemple de propriétés d'un label dans Sirius.

Les éléments acceptent des styles créés et personnalisés séparément, comme ils peuvent contenir des fonctions. Il est nécessaire pour chaque élément de définir sa position dans le layout qu'il appartient, cela en spécifiant sa ligne et sa colonne dans les propriétés comme le montre la capture IV.10.

Comme les styles et les fonctions, nous créons les tables qui vont être insérées dans notre base de données distantes, elles prennent comme paramètre un ou plusieurs inputs qui contiennent l'information à stocker, elles seront utilisées par des listViews pour afficher les

informations enregistrées. Pour plus de détails sur la création d'éditeur graphique avec Sirius, consulter [annexe B]

II.3. Génération du code source

Généralement l'étape qui suit la manipulation du modèle est la génération du code de la nouvelle application, les développeurs utilisaient un outil pour cela qui est "**Acceleo**".

En cherchant sur internet, nous avons trouvé le framework react-native que nous avons déjà cité dans les chapitres [1 et 2], un seul code pour trois plate-formes différentes à la fois, le problème est que l'outil acceleo ne permet pas la génération d'un tel code react-native, c'est la raison pour laquelle nous avons créé notre propre générateur de code react-native.

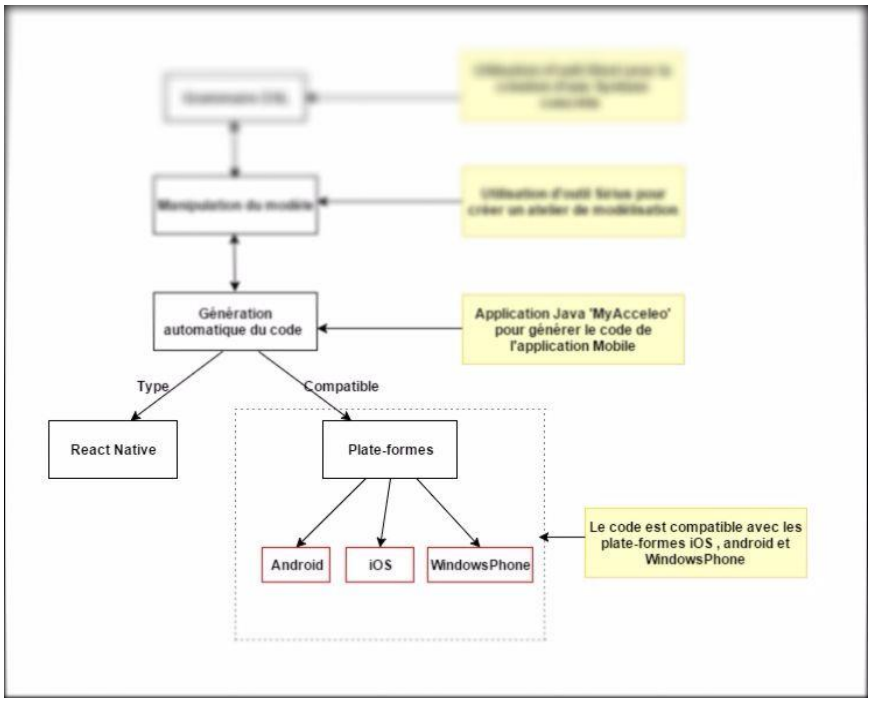


Figure IV.7 : Niveau 3 de la solution (génération du code).

La génération du code se fait par **"My Acceleo"** qui est une application java, pour la retranscription textuelle du modèle vers un code react-native. Elle est composée de six classes java représentées dans le diagramme de classes suivant :

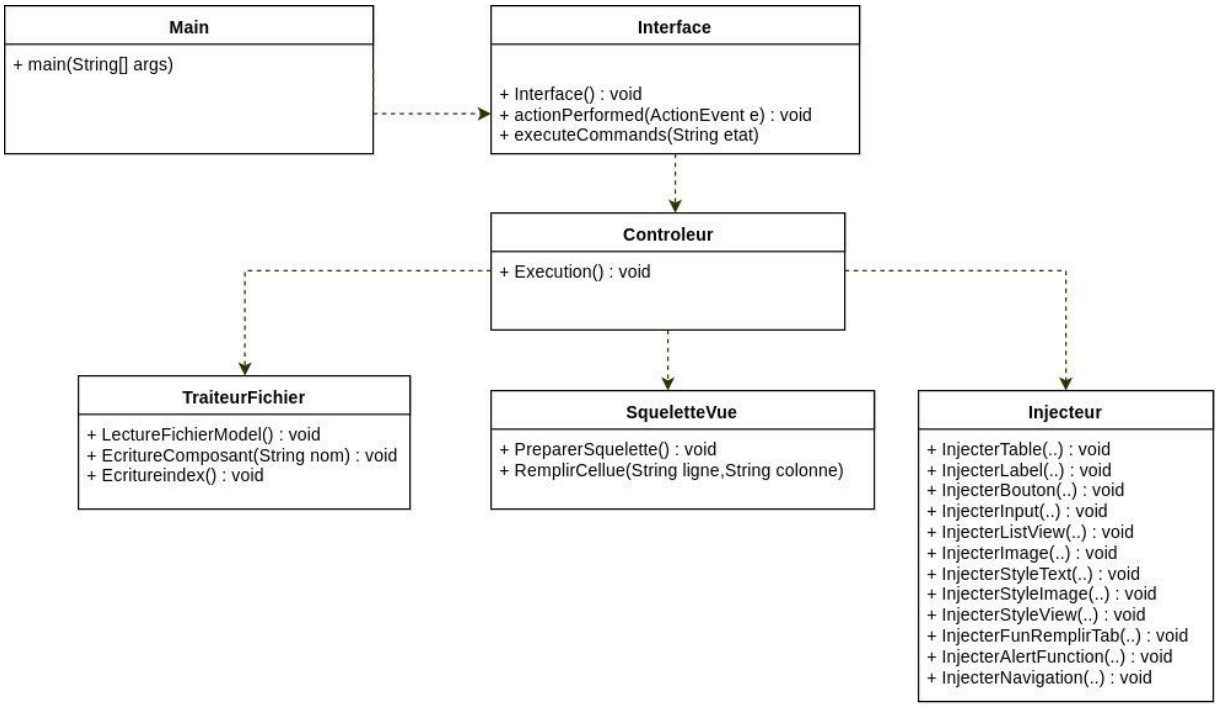


Figure IV.8: Diagramme de classe de l'application java "My Acceleo".

II.3.1 Fonctionnement de MyAcceleo

Nous expliquons dans ce qui suit la figure IV.8 :

Le projet **MyAcceleo** contient la classe "**contrôleur"** qui est le chef d'orchestre de notre application, dans un premier temps elle appelle la méthode "**LectureFichierModel()"** de la classe "**TraiteurFichier"** pour lire le modèle généré par l'outil sirius, ensuite elle lance le traitement et récupère les composants existants dans le modèle afin de construire la vue en appelant la classe "**Squellettevue"**.

Une fois la structure de la vue est construite, la classe "**contrôleur"** relie une deuxième fois notre fichier model et appelle les méthodes de la classe "**injecteur"** pour injecter les bouts de code (les fonctions dans le bloc fonction, les composants dans le bloc vue, les styles dans le bloc styleSheet…) dans les fichiers Javascript de l'application finale.

Pour terminer, la classe **"contrôleur"** appelle une dernière fois les méthodes de la classe "**TraiteurFichier",** mais cette fois-ci elle appelle les méthodes "**Ecriturecomposants()"** et "**Ecritureindex()"** pour obtenir le code complet dans un fichier sous format Js. Il ne reste qu'à copier ce code dans un projet react-native, le compiler et l'exécuter, tout ça se fait automatiquement en exécutant un script qui contient les lignes de commandes du shell.

Le script :

script.js

```
Le script :
#!/bin/bash
export JAVA_HOME=/home/sofiane/install/jdk1.8.0_121/
export ANDROID_HOME=/home/sofiane/Android/Sdk/
cd /home/sofiane/work/MyAcceleo/fichier/pfe
fuser -k 8081/tcp
npm run android .
```

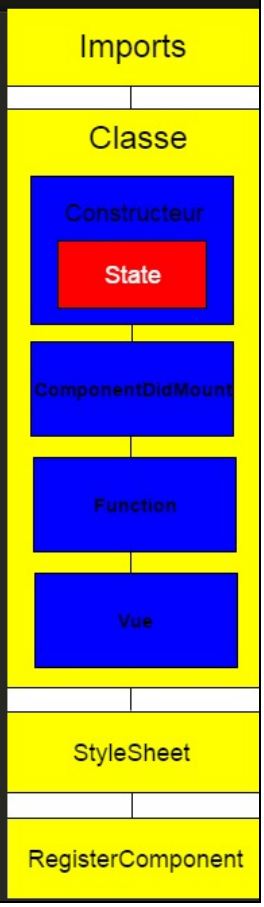
Capture IV.11 : Script pour lancer la nouvelle application.

Rentrons plus dans les détails et voyons ce que c'est un squelette et ce qui va être injecté dedans dans chacune de ses parties.

La figure suivante représente la forme du squelette:

squellette.js

```
import React, { Component } from 'react'
import {AppRegistry} from 'react-native';
import { Button } from 'react-native-elements'

export default class pfe extends Component {
constructor(props) {
//Constructeur
super(props);

this.state = {
//Debutdustate

}
}
componentDidMount() {

}
//function
render() {
//Vue
return(
<View>

</View>
);
}
}
 var styles = StyleSheet.create({
//StyleSheet
});
AppRegistry.registerComponent('pfe', ()=>pfe);

```



Capture IV.12: Représentation du fichier squelette.js.

Comme le montre la figure, le squelette comporte quarte blocs :

1. Le bloc "**imports**" comporte tous les modules que nous pouvons utiliser en générant notre application (firebase, react-native-router-flux, react-native-element…).

- Le bloc "**classe**" définit notre classe et il comporte un bloc "**constructeur**" qui lui-même contient un bloc "**states**".
- Dans le bloc "**states**", nous initions les variables que nous aurons à utiliser dans l'application générée.
- "**Le componentDidmount**" : est invoqué immédiatement après la montée d'un composant, dans ce bloc se fait la récupération des données qui se trouvent dans une base de données à distance.
- Dans le bloc "**function**" se fera l'injection de toutes les fonctions de notre application.
- Dans la partie "**vue**" nous mettrons tous les composants de notre application bien structurés.

2. Le bloc "**StyleSheet**" : tous les styles créés par l'utilisateur dans l'application sirius se trouveront dans cette partie et ils seront appelés depuis la partie vue.
3. "**RegisterComponent**" : c'est grâce à cette partie du fichier que notre classe va être visible dans toute l'application.

III. Application de la méthode proposée

III.1. Processus de développement

Pour appliquer notre étude, nous proposons une application de test "**TodoLIST**", cette dernière est une simple application pour gérer les tâches quotidiennes ou les tâches d'un projet de manière simple et rapide, donc pour cela nous créons un modèle PIM indépendant de toutes les plate-formes. Notre PIM contient :

I. Deux layout : "**TodoLIST**" et "**ListeTaches**", dont le premier représente l'interface d'accueil, il regroupe les éléments suivants:

1. Une image avec un style personnalisé qui est "**StyleImage**".
2. Trois inputs (Tache, Description, Date) qui vont récupérer les informations entrées par le mobinaute.
3. Deux boutons, le premier est "**AjouterTache**", il est utilisé pour enregistrer la nouvelle tâche dans la table "**Tache**" que nous allons détailler dans le point suivant et

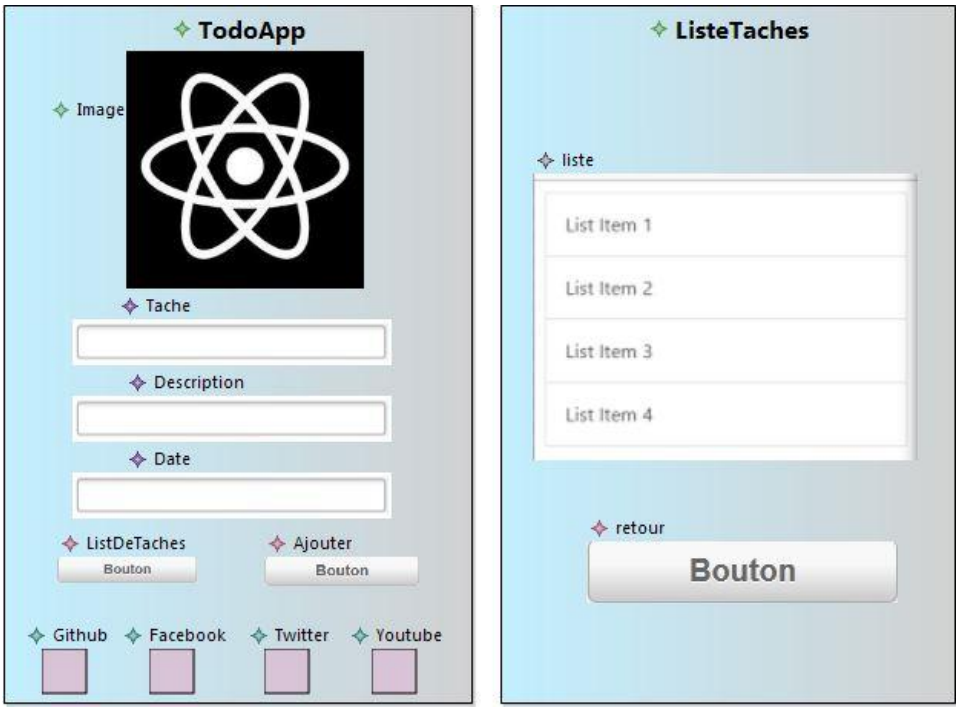
le bouton "**ListDeTaches**" qui permet d'aller vers le layout "**ListeTaches**" via la fonction de navigation "**Vliste**", pour consulter la liste des tâches créées.

4. Quatre icônes de type "**SocialIcon**" : facebook, github, twitter et youtube.

Tandis que le deuxième layout contient :

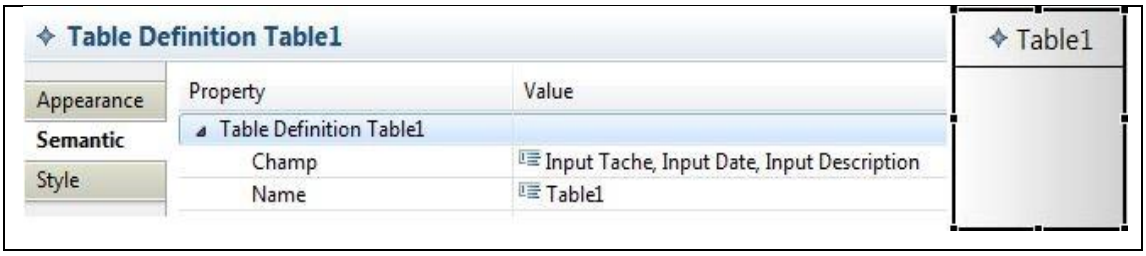
1. Une simple listView pour afficher les informations à partir de la table "**Tache**".
2. Un bouton "**retour**" qui va faire un
3. retour vers layout "**TodoLIST**" en utilisant la fonction de navigation "**Retour**", pour créer une nouvelle tâche.

Après avoir modélisé, nous obtenons le résultat suivant:



Capture IV.13: la modélisation des layouts.

II. Une table nommée "**Tache**", pour enregistrer les tâches créées, chaque tâche est définie par : un nom de tâche, une description, la date de la tâche. Notre table fait référence aux trois inputs que nous avons déjà créés dans le layout "**TodoLIST**" pour récupérer les informations et les enregistrer.



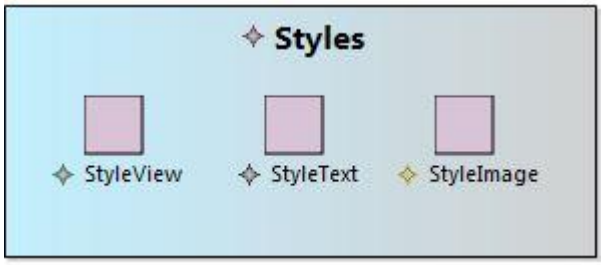
Capture IV.14 : La table "Tache" et ses propriétés.

III. Quatre fonctions :

1. Une fonction d'alerte : "**Alerte**" pour afficher une alerte en cas d'un clic prolongé sur le bouton "**AjouterTache**", pour décrire la fonctionnalité de ce bouton.
2. Une fonction remplirTable : "**EnregistrerTache**" qui permet de récupérer les entrées de chaque input et les stocker dans la table "**Tache**". Cette fonction permet aussi de faire le lien entre notre table et la listView pour afficher les informations dans cette dernière.
3. Deux fonctions de navigation : "**Vliste**" et "**Retour**", la première est utilisée pour naviguer vers le layout " **ListeTaches**" tandis que la deuxième est pour faire un retour vers le layout d'accueil.

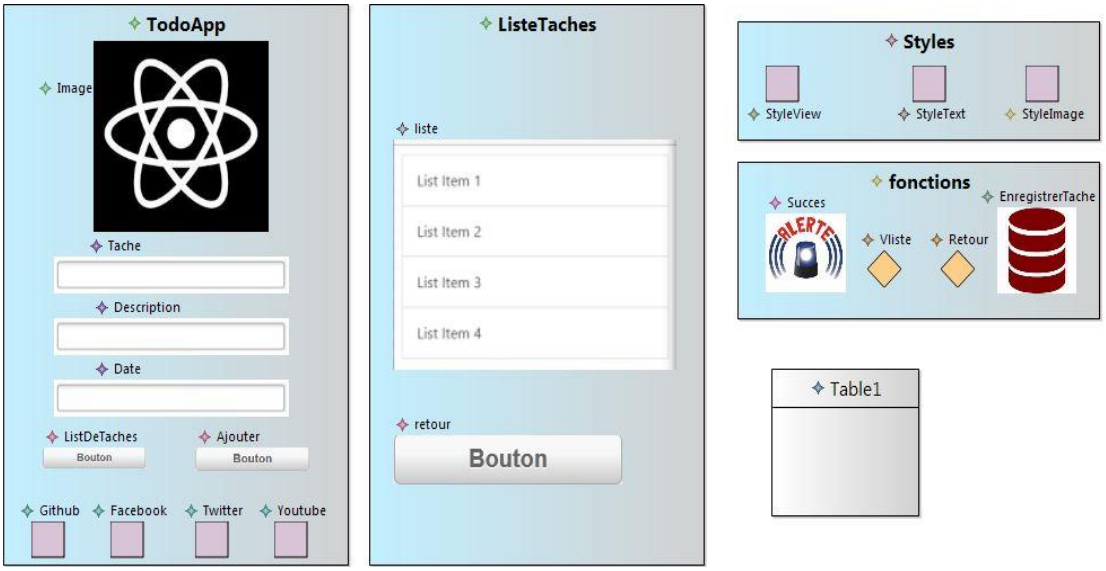
Capture IV.15: Les fonctions.

IV. Trois styles : un "**StyleView**" pour personnaliser les vues, un "**StyleText**" pour définir la forme du texte et un "**StyleImag**e" pour paramétrer l'image que nous avons insérée dans layout d'accueil.



Capture IV.16 : Styles personnalisés.

Le résultat de notre modélisation est le suivant :

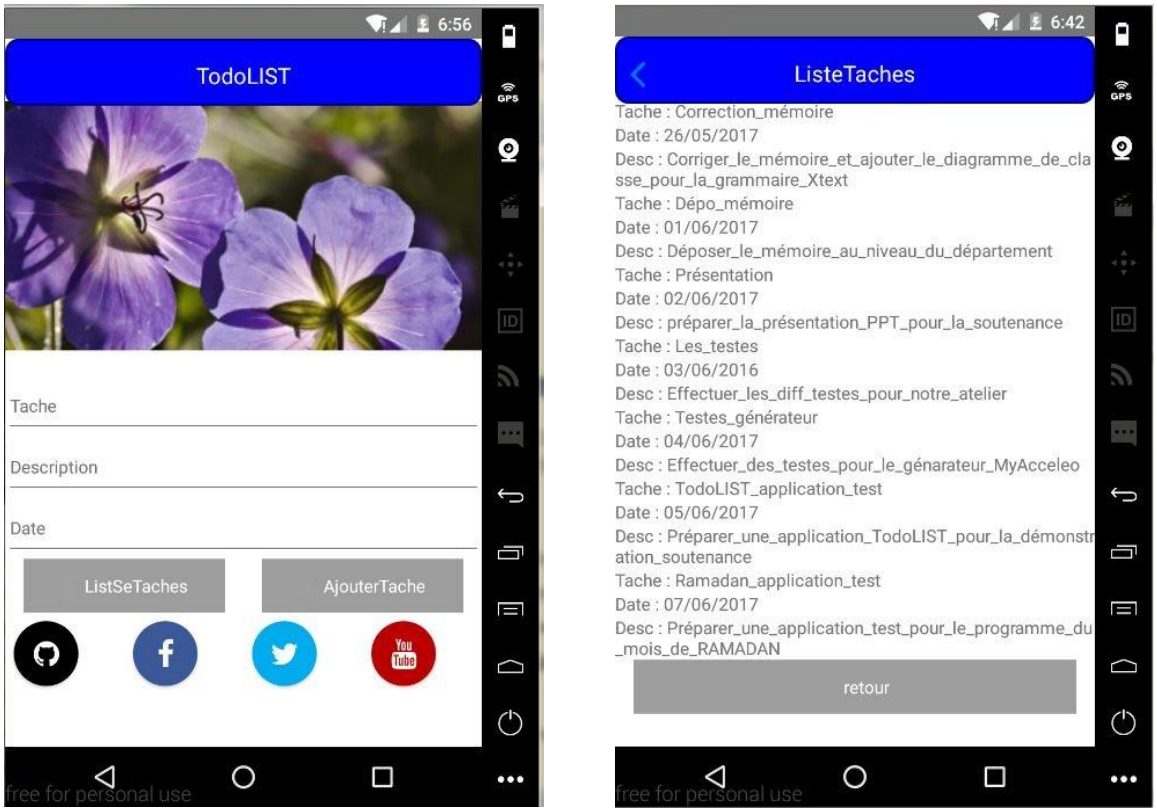


Capture IV.17 : la modélisation de l'application TodoLIST.

L'étape suivante est la génération du code source de l'application "**TodoLIST**", pour cela nous utilisons notre générateur du code "**MyAcceleo**", les étapes sont résumées dans les points suivants :

1. Spécifier le chemin du modèle conçu dans le générateur du code.
2. Exécuter l'application "**MyAcceleo**" comme toute application java.
3. La compilation de l'application créée se lance automatiquement dès que le code est généré.
4. Enfin, l'application est lancée dans l'émulateur (ou dans l'appareil mobile).

Le résultat est dans les captures suivantes:



Capture IV.18 : l'application "TodoLIST".

III.2. Evaluation

Notre atelier nous a permis de concevoir l'application de test "**TodoLIST**" personnalisée, comme elle permet de concevoir d'autres applications en utilisant les éléments de la palette, dans notre atelier, nous avons pu définir les différentes propriétés pour chaque élément, personnaliser leurs styles et les relier entre eux avec des fonctions.

Le générateur du code "**MyAcceleo**" a pu générer un code react-native pour l'application "**TodoLIST**", il a placé chaque élément dans sa position exacte, et il nous a permis de lancer l'application dans le mobile, et nous avons obtenu une application "**TodoLIST**" compatible avec les trois plate-formes : android, iOS et WindowsPhone sans toucher au code.

IV. Conclusion

Nous avons présenté les étapes importantes de la création d'atelier pour générer des applications mobiles, ainsi que l'application de teste "**TodoLIST**".

Conclusion Générale

I. Conclusion

Dans le cadre du mémoire de fin d'études, nous avons proposé une solution qui applique le principe de l'approche d'ingénierie dirigée par les modèles pour le développement des applications mobiles. Afin d'aider le développeur à développer des applications mobiles en réduisant le temps et le coût de développement grâce aux avantages des modèles.

Nous avons présenté une partie du développement mobile ainsi que les solutions qui existent, puis nous avons présenté les trois différents frameworks en les comparant et en faisant un choix final. Dans le chapitre trois, nous avons étudié l'approche MDA. Enfin nous avons présenté les étapes de notre démarche et l'application test.

Nous avons donc créé un langage DSL, ce dernier était utilisé pour générer un modèle PIM indépendant de toutes plate-formes d'exécution. Nous avons pu réaliser un éditeur graphique simple à utiliser avec lequel l'utilisateur peut générer une application qui s'exécute sur ANDROID, IOS et WINDOWS PHONE.

II. Perspectives

L'achèvement de ce travail offre plusieurs projets de fin d'études aux étudiants des années suivantes comme :

- **Enrichir la palette avec des nouveaux composants** : il suffit de faire une liste de composants comme (Slidebar côté vue, AfficherDate coté contrôleur, ajouter des relations entre les tables côté modèle) et de les intégrer dans la grammaire Xtext, la palette sirius et dans le programme java "MyAcceleo" qui est claire, modulaire et extensible.
- **Transformer le programme java "MyAcceleo" en un plugin eclipse.**
- **Réaliser le même travail que ce mémoire mais avec d'autres frameworks mobiles (native, hybride ou web) et faire une comparaison à la fin.**

Pour atteindre ces perspectives nous mettons à la disposition des lecteurs dans l'annexe un tutoriel complet pour la manipulation des outils Sirius et Xtext.

Références Bibliographique

[Alain, 2014] Alain B., « Tutoriel : créer un éditeur de diagramme facilement avec Eclipse Sirius », tutoriel pour l'utilisation de sirius, 24/05/2014.
Dernière consultation 06/06/2017 sur :
http://alain-bernard.developpez.com/tutoriels/eclipse/sirius-intro/#LV-A-1

[Antoine, 2017] Antoine C.D., « React Native : le framework JavaScript de Facebook au crible », article scientifique sur le framework react-native, 14/04/2016.
Dernière consultation 06/06/2017 sur :
http://www.journaldunet.com/web-tech/developpeur/1176787-react-native-la-declinaison-de-react-pour-developper-des-apps-natives/

[Bathelot, 2016] Bathelot B., « Définition : SDK », article scientifique pour la définition SDK, 24/10/2016.
Dernière consultation 06/06/2017 sur :
http://www.definitions-marketing.com/definition/sdk/

[Bathelot, 2017] Bathelot B., « Définition : Application mobile », article scientifique pour la définition d'application mobile, 8 mars 2017.
Dernière consultation 06/06/2017 sur :
http://www.definitions-marketing.com/definition/application-mobile/

[Bézivin, 2003] Bézivin J., « La transformation de modèles. Ecole d'Eté d'Informatique CEA EDF INRIA », cours #6, INRIA-ATLAS & Université de Nantes. 2003.

[Bézivin, 2006] Bézivin J., « Introduction à l'ingénierie dirigée par les modèles », cours universitaire, université de Nantes, 2006.

[Cédric, 2017] Cédric L., « Développement mobile : avantages et limites de React Native », article scientifique pour décrire les avantages du framework react-native, 11/01/2017.
Dernière consultation 06/06/2017 sur :
https://fr.linkedin.com/pulse/d%C3%A9veloppement-mobile-avantages-et-limites-de-react-native-c%C3%A9dric-loue

[Cihad, 2014] Cihad h., « Hybrid Apps with Angular & Ionic Framework».Présentation cours, Istanbul Coders, 2014.
Dernière consultation 06/06/2017 sur :
https://www.slideshare.net/cihadhoruzoglu/hybrid-apps-with-angular-ionic-framework

[Elkouhen, 2015] Elkouhen A., « Specification d'un Méta-modèle pour l'adaptation des outils UML », thèse de doctorat, université de Lille 1 France, 2015.

[Emilie, 2015] Emilie C., « Développement d'une application mobile native », article d'explication de la conception d'une application mobile, 28/04/2015.
Dernière consultation 06/06/2017 sur :
http://www.mobizel.com/2015/04/developpement-dune-application-mobile-native-13/

[Estublier 2006] Jean-Marie Favre Jacky Estublier et Mireille Blay. L'ingénierie dirigée par les modèles : au-delà du mda. Hermes Science, Lavoisier édition, 2006.

[Fourati, 2010] Fourati F., « Une approche IDM de transformation exogène de Wright vers Ada», mémoire de master informatique, Ecole nationale d'ingénieurs de Sfax, 11/06/2010.
Dernière consultation 06/06/2017 sur :
https://arxiv.org/ftp/arxiv/papers/1207/1207.6831.pdf

[Gabriel, 2015] Gabriel H., **«** Le développement d'application hybride », article scientifique sur le développement d'application hybride, 23/10/2015.
Dernière consultation 06/06/2017 sur :
http://www.supinfo.com/articles/single/802-developement-application-hybride

[Laine, 2013] Laine, « Généralités sur l'approche MDA », article scientifique sur l'approche MDA, mai 2013.
Dernière consultation 06/06/2017 sur :
http://laine.developpez.com/tutoriels/alm/mda/generalites-approche-mda/#LIX

[Laurence, 2017] Laurence G., « Développement d'applications mobiles avec Cordova », article scientifique, 14/12/2015.
Dernière consultation 06/06/2017 sur :
https://www.ideematic.com/actualites/2015/12/developpement-dapplications-mobiles-cordova/

[Ludovic, 2012] Ludovic T., « Apache Cordova : La future génération d'applications mobiles », article scientifique, 08/05/2012.
Dernière consultation 06/06/2017 sur :
https://www.geeek.org/apache-cordova-la-future-generation-d-applications-mobiles-545.html

[Marwa, 2017] Marwa B., « développement mobile hybride Ionic», présentation de mémoire master II, université Hassan II Maroc, 27/03/2017.
Dernière consultation 01/04/2017 sur :
https://fr.slideshare.net/marwabaich1/ionic-angularjscordovanodejssass

[MaxLab, 2015] MaxLab, « React Native : la nouvelle bombe de chez facebook », article scientifique, 02/02/2015.
Dernière consultation 06/06/2017 sur :
http://maxlab.fr/javascript/react-native-la-nouvelle-bombe-de-chez-facebook/

[MRN, 2017] MRN, « A Material Design style React Native component library », Document matériel de l'API react-native 2017.
Dernière consultation 06/06/2017 sur :
http://mrn.js.org/?ref=producthunt

[Olfa, 2006] Olfa-Hamrouni, « généralité sur le développement d'applications mobiles ». Présentation cours, 2006.
Dernière consultation 05/06/2017 sur :
http://www.isetjb.rnu.tn/jwmu2017/docs/Developpement-Mobie-OLFA-HAMROUNI.pdf

[OMG, 2006] Object Management Group OMG.
UML Diagram Interchange (DI) 1.0, 2006. Dernière consultation 06/06/2017 sur : http://www.omg.org/spec/UMLDI/.

[Rédaction, 2016] La Rédaction, « Développement d'applications mobiles: web, natif, Hybrides », article scientifique pour le développement mobile, 31/05/2016.
Dernière consultation 06/06/2017 sur :
http://www.servicesmobiles.fr/developpement-dapplications-mobiles-web-natif-hybrides-32198/

[Ripkens, 2014] Ripkens B., «Ionic: An AngularJS based framework on the rise». Article scientifique, 28/11/2014.
Dernière consultation 06/06/2017 sur :
https://blog.codecentric.de/en/2014/11/ionic-angularjs-framework-on-the-rise/

[Robert, 2016] Robert, « Five reasons why web developers love React Native», article scientifique, 23/11/2016.
Dernière consultation 06/06/2017 sur :
https://blog.shoutem.com/five-reasons-web-developers-love-react-native/

[Rodrigo, 2016] Rodrigo R., « REACT-NATIVE TOURS.JS #1». Présentation react-native, 2016. Dernière consultation 06/06/2017 sur :
https://reyesr.github.io/toursjs-1-react-native/#/9

[Sandrine, 2017] Sandrine A., « React native : une bonne alternative au développement natif ? », article scientifique dans les applications mobile, 04/04/2017.
Dernière consultation 06/06/2017 sur :
http://www.arca-computing.fr/react-native-une-bonne-alternative-au-developpement-natif/

[Soley, 2000] Soley R., « Model Driven Architecture (MDA) draft 3.2», Object Management Group, 2000.

[Stéphanie, 2015] Stéphanie M., « Présentation d'Ionic Framework » article scientifique, 20/03/2015.
Dernière consultation 06/06/2017 sur :
http://www.duchess-france.org/presentation-de-ionic-framework/

[Vincent, 2016] Vincent, « Cordova : Applications mobiles hybrides », article sur cordova, ionic, phonegap, hybride, 25/02/2016.
Dernière consultation 06/06/2017 sur :
https://vincent-g.fr/2016/02/25/ionic-apache-cordova-developpement-mobile-hybride/

[Yannick, 2009] Yannick's B., « definir-son-dsl-avec-xtext », article scientifique pour la création du DSL, 27/07/2009.
Dernière consultation 01/04/2017 sur :
https://ylizzi.wordpress.com/2009/07/27/definir-son-dsl-avec-xtext/

[Zoontek, 2016] Zoontek, « Votre première app React Native », article scientifique, 04/10/2016.
Dernière consultation 06/06/2017 sur :
http://putaindecode.io/fr/articles/js/react/native/introduction/

[TECFA, 2003] Technologies Internet et Education, «2. Document Object Model (DOM) définition », article scientifique, 2003.
Dernière consultation18/06/2017 sur :
http://tecfa.unige.ch/guides/tie/html/php-dom/php-dom-2.html

Annexes

I. Annexe A : Définir une grammaire DSL avec Xtext

I.1. Généralité sur les DSLs

Un DSL (Domain Specific Language) est un langage dédié à une problématique métier spécifique. C'est une notion qui s'oppose aux langages de programmation dits généralistes, de type Java ou C, ou bien aux langages de modélisation généralistes de type UML.

L'avantage des DSLs réside dans leur puissance d'expression, puisqu'ils sont conçus pour une problématique précise, et dans leur facilité de traitement lorsqu'il s'agit d'effectuer de la validation ou de la transformation (génération de code par exemple).

En contrepartie, un DSL nécessite de définir un méta-modèle et une notation, ce qui représente un temps de développement et d'apprentissage du langage supplémentaire.

I.2. Types de DSL

Il y a plusieurs manières de concevoir un DSL.

Tout d'abord, la définition du DSL peut être envisagée de différentes manières.

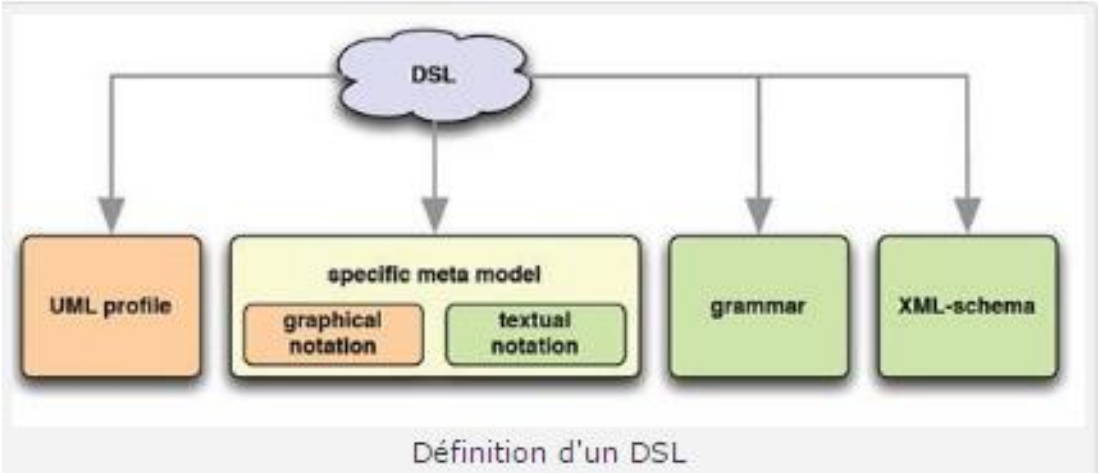


Figure A1.1 : Les différentes manières de concevoir une DSL [Yannick, 2009].

De même, la notation du DSL peut être graphique ou textuelle sous Xtext.

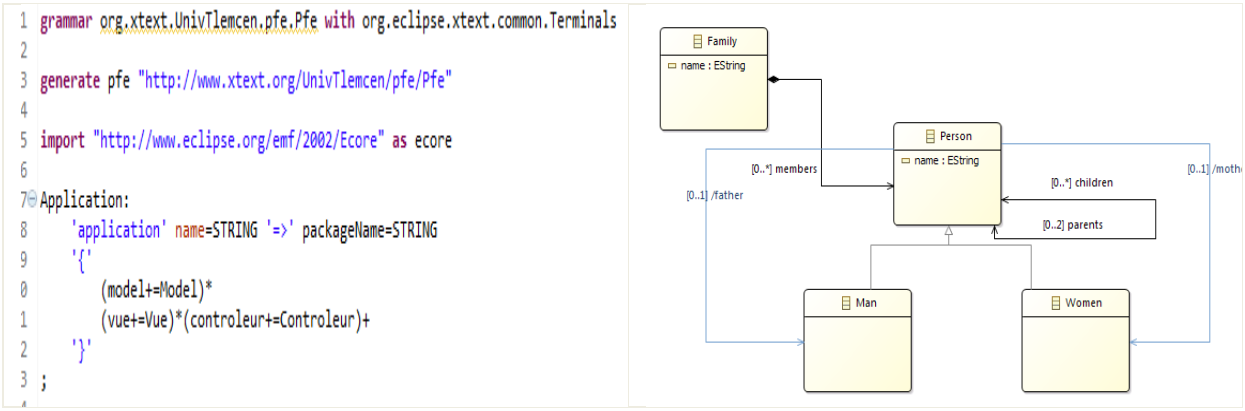


Figure A1.2: Les notations d'un DSL [Yannick, 2009].

Les notations graphiques sont bien sûr plus visuelles et efficaces pour représenter des relations entre éléments. Par contre, dès qu'il s'agit de décrire et d'éditer des détails, la notation textuelle est souvent plus simple à utiliser. D'autre part, elle permet de mettre en œuvre plus simplement des mécanismes de diff et merge.

Il est également possible de mixer les deux notations.

Exemple : Grammaire textuel avec Xtext

Tout d'abord et avant de commencer, il faut préparer l'environnement de travail.

Donc il faut :

1. Télécharger eclipse modeling sur votre ordinateur, rendez vous sur le site officiel: http://www.eclipse.org/downloads/eclipse-packages/
2. Ajouter le plugin Xtext à votre bibliothèque eclipse comme suit :

Dans eclipse : help→ Install New software et terminer le processus d'intégration.

Les étapes :

1. Etape 1: Créer un nouveau projet Xtext

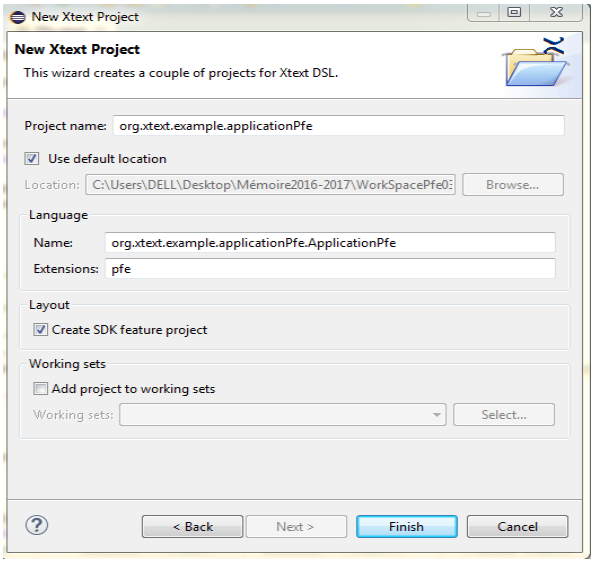
Pour commencer, nous devons d'abord créer des projets Eclipse. Utilisez l'assistant Eclipse pour le faire:

File → New → Project… → Xtext → Xtext project

Choisissez un nom de projet significatif, un nom de langue et une extension de fichier, par exemple :

ProjectName :	org.example.applicationPfe
Language name:	org.example.applicationPfe.ApplicationPfe
Language extensions:	Pfe

Cliquez sur Terminer pour créer les projets.



Capture A1.1 : Création d'un projet Xtext.

Après avoir terminé l'assistant, vous trouverez quatre nouveaux projets dans votre espace de travail.

org.xtext.example.applicationPfe	La définition de grammaire et tous les composants spécifiques au langage (analyseur, validation, etc.).
org.xtext.example.applicationPfe.sdk	Fonctionnalité IDE indépendante de la plate-forme**.**
org.xtext.example.applicationPfe.tests	Tests unitaires pour la langage.
org.xtext.example.applicationPfe.ui	L'éditeur Eclipse et d'autres fonctions connexes.

2. **Etape 2: Ecriture de la grammaire**

- Le fichier **« applicationPfe.pfe »** s'ouvrira automatiquement, dans lequel vous allez écrire votre grammaire DSL.

Maintenant vous pouvez modifier ce fichier avec votre propre grammaire :

ApplicationPfe.xtext

```
1  grammar org.xtext.example.applicationPfe.ApplicationPfe
2   with org.eclipse.xtext.common.Terminals
3
4  generate applicationPfe "http://www.xtext.org/example/applicationPfe/ApplicationPfe"
5
6  Application:
7          (View+=Vue)*;
8  Vue:
9      elements+=Element+;
10 Element:
11     Input | Label ;
12 Input:
13       'Input' (name=ID)?
14       'TextType'  text=TypeH? ';'
15       'End.';
16 Label:
17     'Label' (name=ID)?
18     'Contenu' contenu=ID ';'
19     'End.';
20 enum TypeH:h1 | h2 | h3 | h4 |fontFamily;
```

Capture A1.2 : Exemple de grammaire Xtext.

Examinons plus en détail ce que signifient les différentes règles de grammaire, nous les regroupons dans la figure suivante:

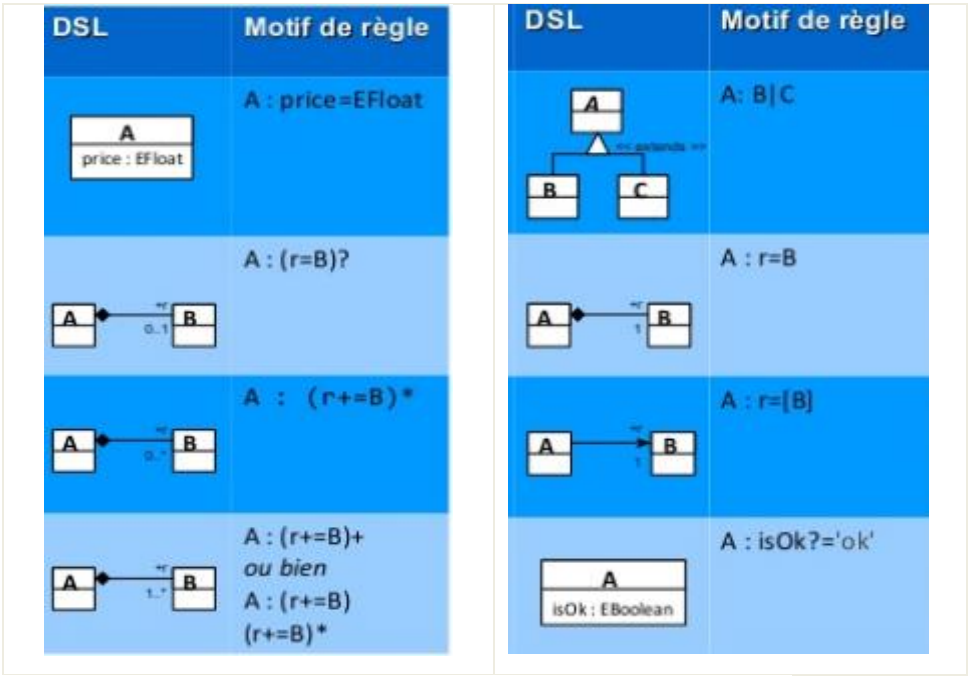


Figure A1.3 : les différentes règles pour la grammaire Xtext [Yannick, 2009].

I.3. Conclusion

Et voilà maintenant vous pouvez créer une simple grammaire Xtext en utilisant l'outil Xtext sous eclipse.

II. Annexe B : Création d'un éditeur graphique avec sirius

II.1. Généralité sur Sirius

La création d'éditeurs graphiques (éditeurs UML, représentation de processus, éditeurs de concepts métier, etc.) est difficile à intégrer dans un cycle itératif court avec le client, car coûteuse en temps et nécessitant souvent l'apprentissage de technologies ou frameworks complexes.

Le projet Eclipse Sirius se propose de faciliter la création d'éditeurs en se basant sur la représentation d'un DSL au sein d'un modèle EMF (Eclipse Modeling Framework).

Sirius est un projet Open Source de la Fondation Eclipse. Cette technologie permet de concevoir un atelier de modélisation graphique sur-mesure en s'appuyant sur les technologies Eclipse Modeling, en particulier EMF et GMF.

L'atelier de modélisation créé est composé d'un ensemble d'éditeurs Eclipse (diagrammes, tables et arbres) qui permettent aux utilisateurs de créer, éditer et visualiser des modèles EMF.
Le site officiel : http://www.eclipse.org/sirius/ .

II.2. Commencer avec sirius

Dans la pratique deux manières possible pour arriver à votre nouvelle atelier :

- **Méthode 1** : Utiliser le projet de type « Empty EMF Project » pour créer le modèle.
- **Méthode 2** : Utiliser les outils Xtext (syntaxe textuelle) ou bien GMF (syntaxe graphique) pour la création d'une syntaxe et générer le modèle.

Nous allons détailler les deux méthodes dans la suite de se tutoriel.

1. Méthode 1 : Syntaxe abstraite avec Ecore

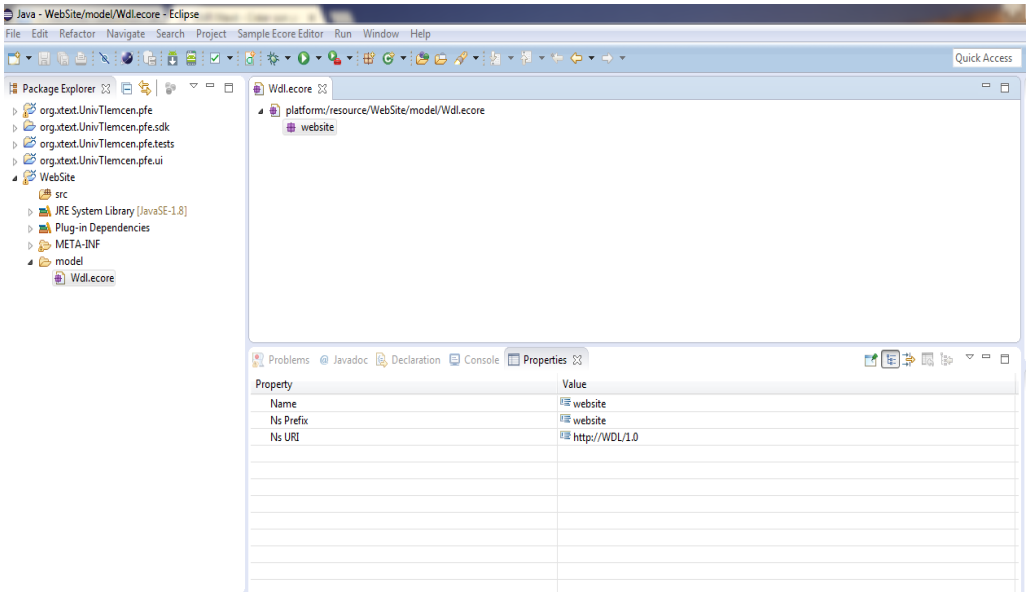
Le langage ECORE est un langage de méta modélisation qui repose sur des notions issues de l'orienté-objet et ne devrait donc pas dérouter les ingénieurs habitués à la programmation par objet et à UML.
L'exemple qui illustre cet annexe est un site web (exemple de base).

1.1. Création d'un fichier Ecore :

Tout d'abord commencez par créer un nouveau projet EMF vide (File → new→ other → EMF → Empty Emf Project), puis dotez-le d'une nature Xtext (Clic droit sur le projet→ Configure → Add Xtext Nature/Convert to Xtext Project).

Dans le dossier model, créez un fichier Ecore (New→ Other→ EMF→ Ecore model), nommé Wdl.ecore.

N'oubliez surtout pas de renseigner les propriétés de votre langage, du genre **name:website**, **nsPrefix:website** et **nsUri:http://WDL/1.0**. Désormais, cet URI servira d'identifiant unique à chaque version de votre langage à travers la plate-forme EMF.



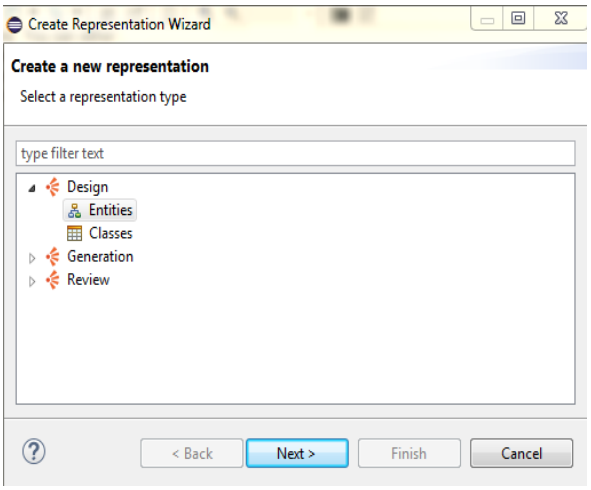
Capture A2.1 : Ecore modèle.

A partir de là, vous avez le choix entre une édition graphique de votre fichier Ecore, ou bien une édition textuelle.

1.2. Méta modélisation Via éditeur graphique :

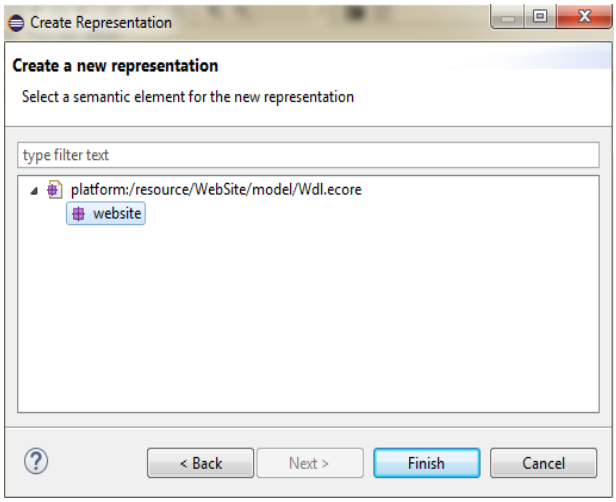
Lancez l'éditeur graphique de diagramme Ecore (clic droit sur le fichier → Initialize Ecore Diagram).

Vous choisissez une représentation, dans notre cas nous avons choisi Design (Entities)**.**



Capture A2.2: Création d'une nouvelle représentation.

Ensuite vous devez spécifier la racine website :



Capture A2.3: Spécification de la racine.

En fin de processus, vous obtenez un fichier Wdl.aird. Double clique dessus et vous verrez apparaître une palette d'outils « à la UML ». La méta modélisation graphique du domaine des sites web est alors triviale **:**

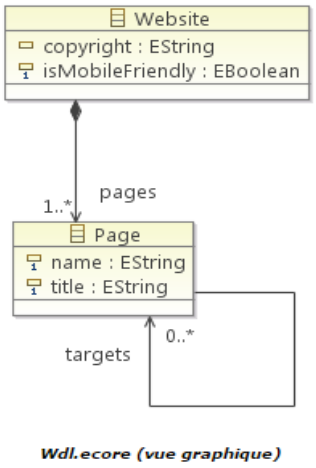


Figure A2.1:Exemple de modélisation graphique avec UML [Alain, 2014].

L'élément racine est webSite, il est décrit par deux Attributs. Il est composé d'une ou plusieurs pages. Une page est décrite par deux attributs aussi et par des références à d'autres pages.

1.3. Méta modélisation Via éditeur textuelle:

L'utilisation d'éditeur textuelle nécessite l'installation d'un composant supplémentaire dans eclipse qui est **OCL Tools**. Vous pouvez l'installer comme tout autre composant (Help → Install Modeling Components).

Une fois l'installation est terminée, vous pouvez éditer votre fichier de manière textuelle. Il vous suffit juste de lancer l'éditeur (clic droit sur le fichier → Open with → OCLinEcore Editor).

1.4. Le choix d'un type de méta-modélisation

Chacune des deux types de Méta Modélisation présentés possèdent ses avantages et inconvénients.

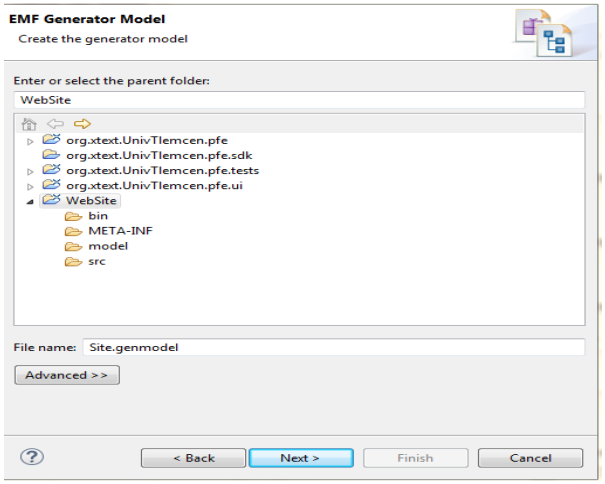
L'avantage majeur de la vue textuelle est de pouvoir exprimer des contraintes OCL directement au niveau du méta modèle.

Comme compromis, vous pouvez prendre le meilleur des deux mondes : commencez le travail avec l'éditeur graphique puis attaquez les finitions avec l'éditeur textuel. Veillez juste à ne pas être contradictoire entre ces deux points de vue sur le même fichier Ecore.

1.5. Génération du modèle

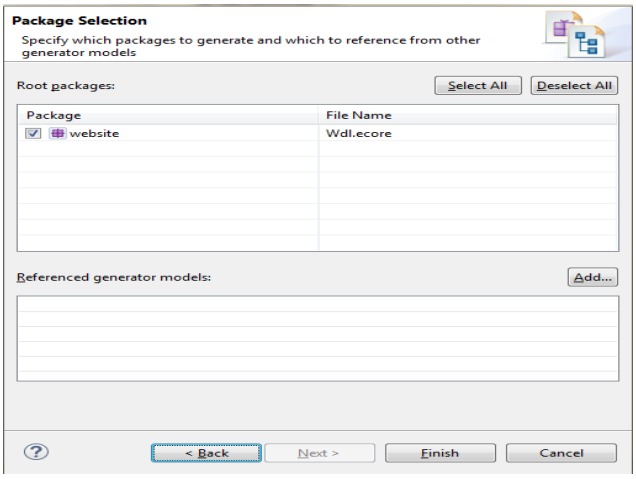
Dans cette étape, vous allez créer un fichier avec l'extension « .genmodel », à partir de l'assistant « EMF Generator Model » pour notre fichier « Wdl.ecore ».

Sur la deuxième page « Select Model importer », sélectionnez « Ecore model », puis sur la page suivante, sélectionnez votre modèle « .ecore ».



Capture A2.4: Sélection de modèle ecore.

Sélectionnez ensuite le package «website » pour la génération :



Capture A2.5 : Sélection du package.

Une fois le fichier est généré, vous pouvez obtenir les trois plugins, « website », « website.edit » et « website.editor ».

Maintenant tout est près pour créer votre éditeur graphique avec sirius, cette étape sera détaillée après avoir parlé de la deuxième méthode.

2. Méthode 2 : Génération du modèle à partir d'une syntaxe concrète

La deuxième méthode consiste à utiliser des outils basés sur les métas modèles comme Xtext et GMF pour la création d'une syntaxe concrète et générer le modèle.

Dans le monde actuel, l'outil Xtext est le framework le plus utilisé pour le développement de langages de programmation et de DSL (Domain-Specific programming Language), surtout quand il s'agit de décrire et d'éditer des détails.

Dans l'annexe précédent **' Définir une grammaire DSL avec Xtext '** nous avons déjà présenté les étapes à suivre pour la création d'un projet Xtext.

Après avoir créé votre projet Xtext vous aurez qu'à définir votre grammaire Xtext.

Dès que vous terminez cette étape, vous pouvez générer l'infrastructure associée à la grammaire.

2.1. Génération de l'infrastructure associée à la grammaire

a) Fichier ".mwe2"

Le fichier « GenerateWdl.mwe2 » situé dans le même dossier que le fichier « Wdl.Xtext » définit la liste des fragments qui vont être utilisés par Xtext pour la génération de l'éditeur. Pour plus de détails vous pouvez consulter le lien suivant : https://eclipse.org/Xtext/documentation/306_mwe2.html .

b) Génération du modèle

La génération de votre grammaire peut se faire de deux façons différentes:

- Soit par clic droit sur le fichier Xtext → « Run As → Generate Xtext Artifacts »
- Soit par clic droit sur le fichier MWE2→ « Run As → MWE2 Workflow »

Cette action va lancer la génération de tous les éléments nécessaires à votre éditeur.

La première fois que vous lancez la génération, le message suivant apparaît dans la console Eclipse :

> **ATTENTION* It is recommended to use the ANTLR 3 parser generator (BSD licence - http://www.antlr.org/license.html). Do you agree to download it (size 1MB) from 'http://download.itemis.com/antlr-generator-3.2.0.jar'? (type 'y' or 'n' and hit enter)*

Il vous suffit de répondre par 'y' et le processus de génération continue.

À la fin de la génération, tous les dossiers « src-gen » de vos plugins devraient être fournis avec un certain nombre de classes.

Et maintenant vous arriver à l'étape de création de votre éditeur graphique avec sirius que nous allons la détailler dans le titre suivant.

2.2. Création de l'atelier

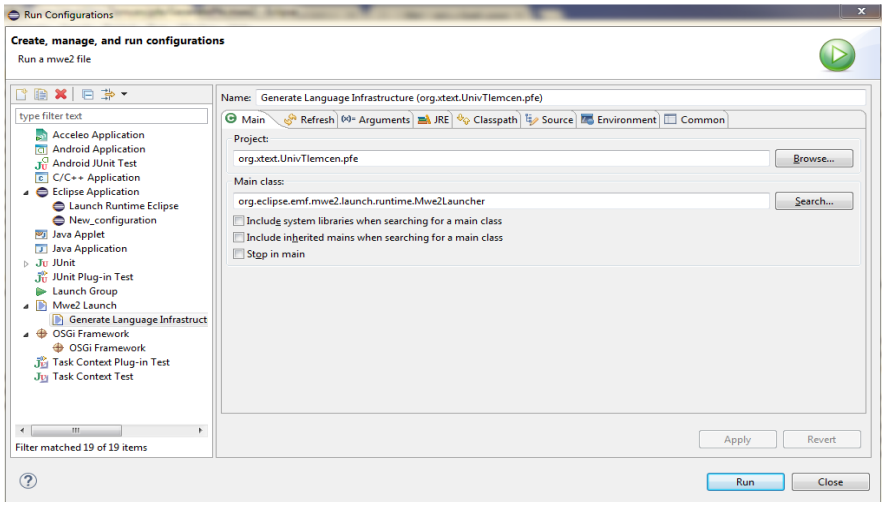
Parmi les points forts de sirius est que le rendu peut être observé en temps réel Parallèlement à la spécification, à partir de l'instanciation du modèle que vous voulez le représenter.

Nous allons voir maintenant toutes les étapes pour la construction d'un atelier sous sirius.

Les étapes :

1. **Etape 1: lancez un Eclipse avec vos plugins**

Cette étape consiste à lancer un Eclipse avec nos plugins. Pour cela, ouvrez d'abord le panneau « Run configurations » via le menu « Run→ Run configurations », puis créez une nouvelle configuration de type « Eclipse Application ».

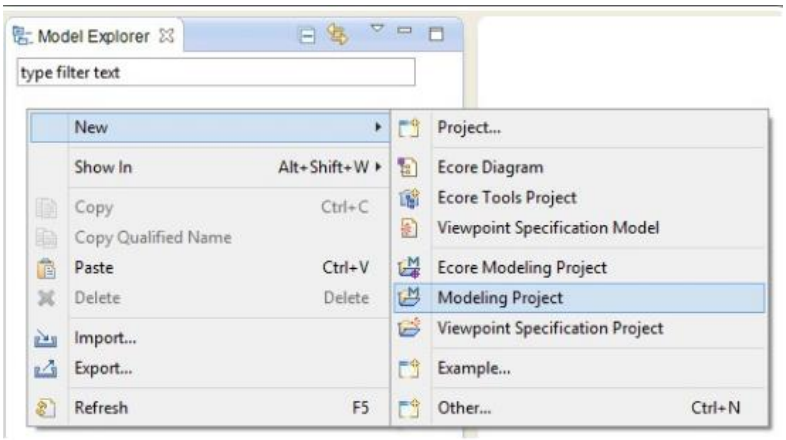


Capture A2.6: Configuration d'exécution.

Puis appuyez sur "Run", une nouvelle instance d'eclipse s'apparaitre, et vous allez travailler uniquement dans cette dernière.

2. **Etape 2: Instanciation du modèle**

Pour instancier le modèle, vous devez créer un projet de type modeling, donnez un nom au projet par exemple "WebSite.example".



Capture A2.7: Modeling project.

Dans le projet créé, on peut identifier un fichier appelé « representations.aird ». Ce dernier permettra à Sirius de stocker l'agencement des éléments dans l'éditeur que l'on va créer, de manière indépendante du modèle lui-même.

Notez qu'il existe un seul fichier « .aird » par « Modeling Project », mais chaque fichier peut stocker plusieurs représentations.

Créez ensuite une instance de votre modèle, pour cela existe deux manières :

- Si vous avez utilisé la méthode 1 '**Syntaxe abstraite avec Ecore** '

Créez une instance du modèle grâce au menu (New → Example EMF Model Creation Wizards → Website model). Donnez un nom à votre modèle, puis sélectionnez comme objet racine.

- Si vous avez utilisé la méthode **2 'Génération du modèle à partir d'une syntaxe concrète**'

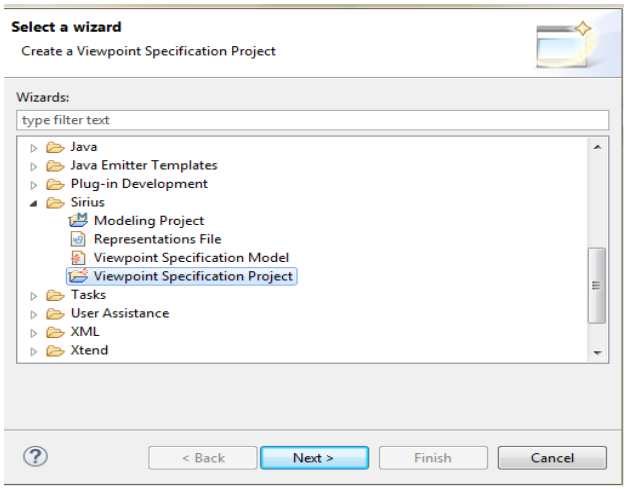
Créez une instance du modèle comme suit : (New → File) et donnez un nom à votre nouveau fichier qui va prendre obligatoirement l'extension donnée dans la création du projet Xtext, exemple : '.Wdl'.

Vous pouvez ensuite créer les éléments du votre modèle.

3. Étape 3: Initialisation de l'éditeur

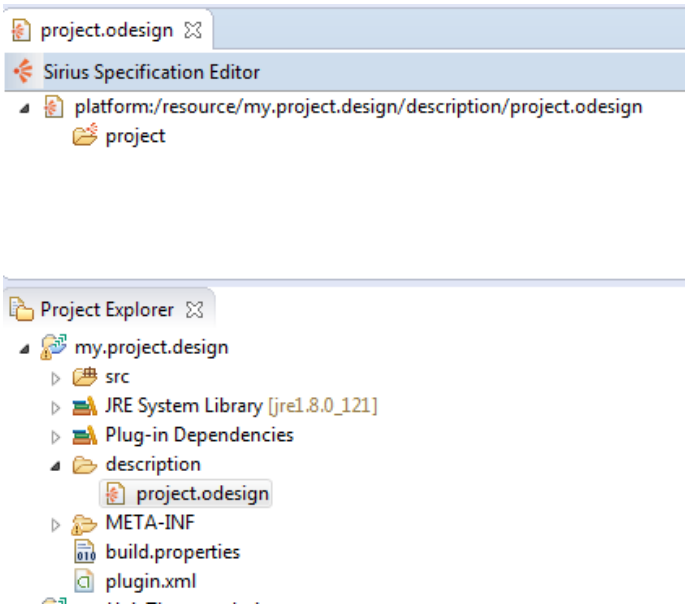
La création des éditeurs de Sirius se fait à travers des projets que l'on nomme « Viewpoint Specification Project ».

Vous devez créez un projet de type 'Viewpoint Specification Project' (File→ new → other → Sirius → Viewpoint Specification Project).



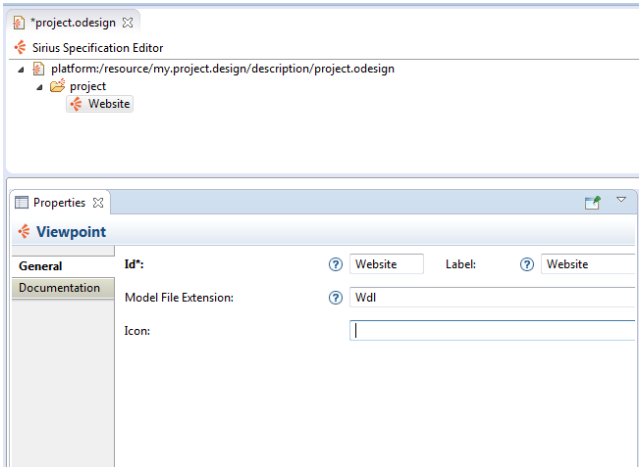
Capture A2.8 : Création d'un projet " Viewpoint Specification Project".

Enfin de processus vous trouverez un fichier avec l'extension '.odesign' dans lequel vous allez définir vos éléments.



Capture A2.9 : Exemple de fichier ".odesign".

La première étape est de créer une 'Viewpoint', avec un clic droit sur l'élément « Projet », créez un ViewPoint. Comme son nom l'indique, un viewpoint sert à définir une manière de représenter un modèle.



Capture A2.10 : Création d'un ViewPoint.

Dans les propriétés, vous donnez un Id et un label, et Dans le champ « Model File Extension », mettez l'extension des fichiers de modèle que vous voulez associer à votre viewpoint.

Vous devez ensuite associer une représentation à votre Viewpoint afin d'indiquer à Sirius quel type d'éditeur vous voulez créer. Il faut noter qu'il existe plusieurs types d'éditeur : des éditeurs sous forme d'arbre, de tableau, de diagramme ou encore de diagramme de séquence.

Pour se faire, clic droit sur 'viewpoint', puis choisissez (New Representation → Diagram Description), et dans les propriétés, renseignez un Id, un label et l'objet de base représenté par le diagramme dans le champ 'Domain class'.

2.3. Spécification des éléments et la création de la palette

2.3.1. Spécification des éléments

Cette section décrit comment spécifier les éléments graphiques qui devraient apparaître sur votre diagramme et comment les organiser.

Notez que tous les éléments graphiques et les outils sur un diagramme de Sirius font partie d'une couche, chaque diagramme doit avoir une couche par défaut, cette couche par défaut est créée lors de la création du diagramme. Lorsqu'une couche est activée, tous les éléments graphiques et outils qu'il définit sont également activés.

Pour plus de détails :

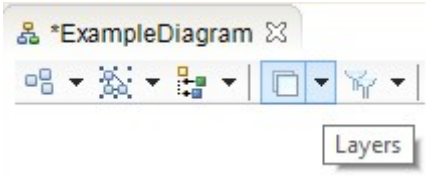
https://www.eclipse.org/sirius/doc/specifier/diagrams/Diagrams.html#layers_graphical_elements

a. Les noeuds & les conteneurs:

Les diagrammes de Sirius peuvent contenir plusieurs types d'éléments graphiques, qui sont décrits dans les éléments de la couche. Les plus utilisées sont: les noeuds et les conteneurs.

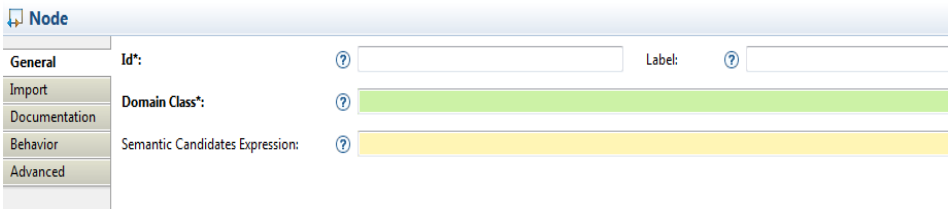
Mais avant de choisir une représentation aux éléments, vous devez d'abord spécifier un "Layer".

Le "Layer" permettra à l'utilisateur d'afficher ou non certains éléments à la volée grâce au bouton idoine dans la barre d'outils de l'éditeur généré.



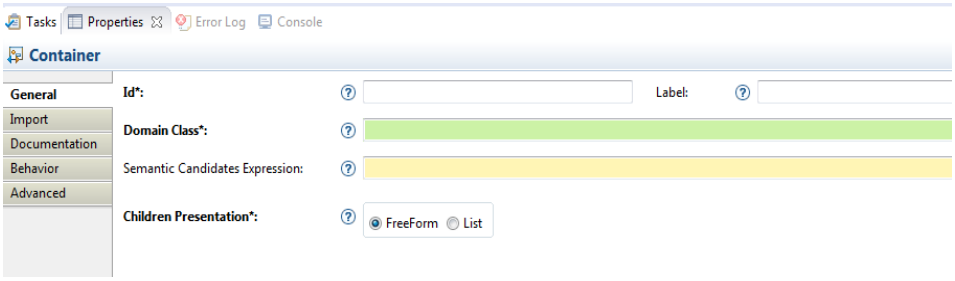
Pour le créer, sélectionnez votre diagramme, clique droit et sélectionnez « New Diagram Element → Default Layer Layer ». Puis, donnez à cet élément un ID et un label.
Maintenant il est possible de créer des noeuds et des conteneurs.
Voici les deux manières pour les créer :

- Nœuds pour les éléments qui ne peuvent contenir d'autres éléments. Pour se faire, clique droit sur le layer que vous avez créé et choisissez (New Diagram Element → Node). Il reste qu'à spécifier les propriétés du noeud en donnant l'ID, label, domain Class (la classe du domaine que le nouveau noeud représentera) et finalement le champ "semantic Candidat" pour indiquer l'expression permettant d'accéder aux éléments à afficher.



Capture A2.11 : Propriétés d'un nœud.

- Conteneurs pour éléments pouvant contenir d'autres éléments, y compris d'autres conteneurs, de la même manière que Node vous créez un conteneur : clique droit sur le layer que vous avez créé et choisissez (New Diagram Element → Container), puis remplissez les propriétés.



Capture A2.12 : Propriétés d'un conteneur.

2.3.2. La création de la palette

Pour éditer le modèle, vous devez créer une palette, cette dernière va contenir des sous-sections des outils de manipulation d'éléments. Pour se faire, clique droit sur le layer que vous venez de créer et choisissez (New Tools → Section).

Dans cette nouvelle section, vous pouvez placer des noeuds et des conteneurs en cliquant un clique droit sur la section et choisir soit:

- **Noeud:** "New Element Creation → Node creation".
- **Conteneur:** "New Element Creation → Container creation".

Après l'ajout de toutes les propriétés, vous trouvez la palette et vous pouvez la remplir au fur et à mesure.

II.3. Conclusion

Dans cet annexe nous avons donné les détailles de la création d'un éditeur graphique sous sirius, vous saurez maintenant manipuler un modèle PIM et créer un premier éditeur.

Et pour plus d'informations consulter le site officiel de sirius :

https://wiki.eclipse.org/Sirius/Tutorials/StarterTutorial

Liste des figures

Liste des tableaux

Master deux en Génie Logiciel 2016/2017	Réalisé par : BELHADJ KACEM Soufyane & SENOUSSAOUI Ikram	جامعة أبو بكر بلقايد UNIVERSITÉ DE TLEMCEN

Titre :

Développement d'un atelier graphique pour la génération d'applications mobiles multi-plateforme.

Mots clés : MDA, développement multi-plateforme, react-native, applications mobiles, DSL, XTEXT, SIRIUS.

Résumé: **C**e sujet de mémoire vise à étudier et appliquer l'approche d'ingénierie dirigée par les modèles pour le développement des applications mobiles. **E**ntre des applications pour Android, iOS et Windows Phone, la demande en développeurs n'a jamais été aussi grande. Chacun a un langage de développement qui lui est propre. Cela oblige les entreprises à avoir un panel de personnel possédant les capacités de développement dans différentes plate-formes pour but de créer une application sur chacune d'entre elles. **D**'où la naissance des applications multi-plateformes. Et vu le nombre important des plate-formes existantes, ce type de développement est devenu très complexe. **C**ela nous ramène à penser à une solution pour améliorer le développement des applications mobiles multi-plateformes en se basant sur l'approche **MDA**. Le but est d'offrir un atelier permettant de manipuler des interfaces utilisateurs compatibles avec les différentes plate-formes et qui utilisent des bases de données (utilisations des bases de données distantes) afin de générer des applications react-native qui s'exécuteront sur les systèmes d'exploitation ANDROID, iOS et WindowsPhone.

Abstract: **T**he subject of this memory is in the field of Software Engineering, and is designed to study the model-driven engineering approach (MDA). **B**etween applications for Android, iOS and WindowsPhone, the demand for developers has never been greater. These three big players compete for the top of the standings. Each of them has its own programming language, this requires companies to form a team of developers with development capabilities in differents platforms in order to create applications on each of them, and this is the reason for the appearance of cross-platform applications. Development has become very complex because of this multiplicity of platform execution. **T**his leads us to think of a solution to improve the development of cross-platform mobile applications using the model-driven engineering approach (MDA). The object is to generate user interfaces that are compatible with differents platforms and that use databases (remote databases) in order to generate react-native applications that will be running on ANDROID, iOS and WindowsPhone operating systems.

ملخص : موضوع هذه الذاكرة في مجال هندسة البرمجيات, و هو يهدف إلى دراسة الهندسة الموجهة بالنماذج من أجل تطوير التطبيقات النقالة.
بين تطبيقات الأندر ويد , أيفون و ويندوزفون , لم يحدث أن عرفنا طلبا كبيرا على مطوري المجال. يتنازع العمالقة الثلاث على رأس الصدارة, لكل منهم لغته الخاصة في البرمجة . هذا ما يفرض على الشركات تكوين فريق من المطورين ذوي قدرات برمجية في كل منصات التنفيذ لإنشاء تطبيقات على كل منهم .و هذا ما أدى إلى ظهور ما نسميه في مجال تطوير التطبيقات النقالة, بالتطبيقات متعددة المنصات. و بالنظر إلى العدد الهائل من المنصات الموجودة حاليا أصبح هذا النوع من التطور معقدا جدا.
هذا ما دفعنا إلى التفكير في إيجاد حل لتعزيز تطوير التطبيقات النقالة متعددة المنصات باستعمال الهندسة الموجهة بالنماذج.الهدف من ذلك هو صنع واجهات المستخدم تتوافق مع منصات مختلفة إضافة إلى دمج قواعد البيانات فيها (استخدامات قواعد البيانات عن بعد) المحمول من خلال تقديم ورشة عمل لإنشاء تطبيقات من شأنها أن تعمل على أنظمة التشغيل: الأندر ويد ,أيفون وَ ويندوزفون.

.