



UNIVERSITÉ DE LARBI TÉBESSI, TÉBESSA



FACULTÉ DES SCIENCES EXACTES ET SCIENCES DE LA NATURE ET DE LA VIE

DÉPARTEMENT DES MATHÉMATIQUES ET INFORMATIQUE

LABORATOIRE DES MATHÉMATIQUES, INFORMATIQUE ET SYSTÈMES (LAMIS))

Thèse

PRÉSENTÉE EN VUE DE L'OBTENTION DU DIPLÔME DE

DOCTORAT LMD

INTITULÉE :

Contribution au développement d'un Processus de Composition d'Applications Mobiles Adaptatives à base d'Entités Logicielles Hétérogènes

Discipline : Informatique

Spécialité : Système d'information coopératif

Présentée et soutenue publiquement par

AFRAH DJEDDAR

devant le jury ci-dessous :

Pr. Mohammed Rida Laouar
Pr. Nacira Ghoualmi
Dr. Nabiha Azizi
Dr. Makhlof Derdour
Dr. Hakim Bendjenna
Pr. Abdelkrim Amirat

Université de Larbi Tébessi
Université de Badji Mokhtar
Université de Badji Mokhtar
Université de Larbi Tébessi
Université de Larbi Tébessi
Université de Souk-Ahras

Président
Examinatrice
Examinatrice
Examineur
Encadreur
Co-encadreur

20 MAI 2016

CONTRIBUTION AU DÉVELOPPEMENT D'UN PROCESSUS DE
COMPOSITION D'APPLICATIONS MOBILES ADAPTATIVES
À BASE D'ENTITÉS LOGICIELLES HÉTÉROGÈNES

AFRAH DJEDDAR

UNIVERSITÉ DE LARBI TÉBESSI



LABORATOIRE LAMIS
LE 20 MAI 2016

RÉSUMÉ

De nos jours, la plupart des activités humaines reposent sur l'utilisation des appareils mobiles. Ces derniers sont caractérisés par des configurations matérielles et logicielles différentes. Cependant, l'adoption massive et récente de cette technologie explique la demande croissante des logiciels spécifiques qui dépendent fortement de leurs environnements d'exécution. L'hétérogénéité des appareils mobiles montre que la portabilité des entités logicielles qui représentent des fonctionnalités à effectuer pour combler les exigences des utilisateurs joue un rôle important dans le domaine de développement des applications mobiles. Malgré la disponibilité d'un grand nombre des applications mobiles avec ces différentes formes d'implémentations (composant, service, application, etc.), les besoins des utilisateurs différents les uns des autres. Ces issues illustrent qu'il y a de facto un besoin d'un processus de composition qui permet la réutilisation des entités logicielles déjà existantes afin de développer des applications mobiles selon les besoins des utilisateurs ainsi que ses comportements soient spécialisés selon les informations contextuelles des appareils mobiles qui les supportent.

L'objectif principal de cette thèse est de proposer un processus pour la composition d'applications mobiles contextuelles en utilisant des entités logicielles pré-existantes. Dû au fait que ces entités logicielles ainsi que leurs environnements d'exécution présentent différents points d'hétérogénéité, le processus proposé doit prendre en compte ces issues en raison de conduire à des applications mobiles cohérentes et adaptatives. Afin d'atteindre cet objectif, nous adoptons l'approche de modélisation - l'ingénierie dirigée par les modèles - pour réaliser notre processus de composition baptisé CAMAP (Composition of Adaptive Mobile Application Process).

Afin d'illustrer l'applicabilité et l'efficacité de notre processus de composition, nous développons un prototype pour le processus de composition proposé CAMAP. Nous présentons dans un premier temps pour les architectes des logiciels une palette qui leur donne la possibilité de définir graphiquement l'aspect fonctionnel de l'application mobile désirée. Ensuite, le processus de composition proposé a pour objectif de générer l'architecture détaillée d'une application mobile composite adaptable à un contexte bien défini en se basant sur la description fonctionnelle donnée. Ce processus est basé sur un ensemble de sous-algorithmes de composition dirigés par le contexte des objets de composition et de l'environnement d'exécution choisi. Ces sous-algorithmes proposés reflètent une succession de transformations de modèles pour réaliser les différents passages entre les représentations proposées pour l'application mobile composite.

MOTS CLÉS : Application mobile, Appareil mobile, Réutilisation, Entité logicielle, Processus de composition, Hétérogénéité, Description du contexte, Modélisation.

ABSTRACT

Today, most human activities based on the use of the mobile devices which are characterized by different hardware and software configurations. However, the recent widespread adoption of this technology explains the increasing demand of specific software which depends heavily on its execution environment. This software reflects mobile applications representing a set of software entities to perform in order to meet the user's needs. The heterogeneity of the mobile devices demonstrates that the portability of the software entities which represent the functionalities to perform in order to fill the user's requirements play an important rôle in the field of the mobile applications development. Despite the availability of a large number of mobile applications with their various forms of implementations (component, service, application, etc.), the needs of users differ from each other. These challenges illustrate that there is de facto a need for a composition process which allows the reuse of existing software entities to develop mobile applications according to the user's needs as well as their behaviors must be specialized according to the contextual information of the mobile devices they support.

The main objective of this thesis is to propose a process to compose adaptive mobile applications using existing software entities. To the fact that these software entities and their execution environment present different points of heterogeneity, the proposed process must take into consideration these issues for leading to consistent and adaptive mobile applications. We adopt the modeling approach to implement our composition process baptized CAMAP (Composition of Adaptive Mobile Applications Process).

In order to illustrate the applicability and the effectiveness of our composition process we develop a prototype for the proposed composition process CAMAP. We present at first to the software architects a palette that gives them the ability to define graphically the functional aspect the desired mobile application. After, the proposed composition process aims to generate the detailed architecture of the composite mobile applications that are adaptable to a well-defined context based-on a given functional description. This process is based on a set of sub-algorithms headed by the context of the composition objects and the chosen execution environment. The proposed sub-algorithms reflect a succession of model transformations to realize the different passage between the proposed representations for the composite mobile application.

KEY WORDS : Mobile application, Mobile device, Reuse, Software entity, Composition process, Heterogeneity, Description context, Modeling.

DÉDICACE

Je voudrai exprimer ma gratitude à mes parents, à mes sœurs Rania et Ilhem et à mon frère Badis pour l'amour qui me témoignent et pour la part certaine qu'ils ont joué dans la réalisation de cette thèse. En fait, ils ont vécu tous le rôle d'être un doctorant en train de réaliser sa thèse, ils étaient toujours inquiètes et ils faisaient beaucoup de soucis pour moi tout en me posant toutes sortes de questions : Est-ce que ça te fait du bien Afrah? T'as fini la rédaction de ton article? T'as déjà eu l'acceptation dans un journal ou pas encore? Est-ce que ça va encore être long? Quand est-ce que tu vas déposer la thèse? Pour quand sera ta soutenance?. Un merci spécial est adressé à ma mère -ma vie- pour toutes les tasses de breuvages chauds qu'elle m'a offertes, la grande attention qu'elle m'a apportée surtout quand je reste plusieurs heures derrière mon bureau devant mon ordinateur en train de travailler. Un grand merci à mon père pour tous les efforts qu'il a entrepris pour moi, pour son aide et pour son précieux soutien qui m'ont permis de devenir ce que je suis aujourd'hui. Comme je n'oublierai jamais l'inquiétude et l'encouragement de ma très chère sœur Rania tout au long de ces années. Je remercie aussi très chaleureusement les autres membres de ma famille (oncles, cousines et cousins, etc.) qui m'ont soutenue de près ou de loin durant mes études doctorales.

Je ne saurai passer sous silence l'apport inestimable de mon encadreur Dr. Hakim Bendjenna pour sa grande disponibilité et ses encouragements tout au long de la préparation de cette thèse. Je veux adresser tous mes sincères remerciement à lui pour la confiance qu'il m'a accordé ainsi que la grande liberté d'idées et de travail qu'il m'a donnée.

*A mes parents
A mon frère Badis
A mes soeurs Rania et Ilhem
A mon encadreur Dr.Hakim Bendjenna*

REMERCIEMENT

Je fais partie des personnes qui croient mordicus qu'il n'y a de force ni de puissance que par Dieu. Mes remerciements vont tout premièrement à dieu le Tout-Puissant de m'avoir aidé toute ma vie et pour la volonté, la santé et la patience qui m'a donné durant tous ces années d'études, sans lui ce manuscrit n'aurait pu voir le jour.

Ces trois années de thèse sont passées bien plus vite que je ne l'avais jamais imaginé au départ et puis finalement. Elles m'ont considérablement changé et sont sans doute les plus riches que j'ai vécues jusqu'alors en terme de rencontres, de voyages et d'expériences et ne me laissent vraiment que des bons souvenirs. Ces années de travaux de recherche furent pour moi une réelle expérience scientifique, professionnelle et humaine. De nombreuses personnes ont contribué à l'aboutissement de ces travaux et je souhaite ici leur exprimer ma gratitude.

Le travail de recherche présenté dans cette thèse a été effectué dans l'équipe de recherche 3SI (Ingénierie des systèmes d'information et imagerie) du laboratoire LAMIS (Laboratory of Mathematics, Informatics and Systems) de l'Université de Larbi Tébessi. Les divers travaux de recherche ont été réalisés sous la direction scientifique du Dr. Hakim Bendjenna, Pr. Abdelkrim Amirat que je tiens à leurs exprimer ma profonde reconnaissance pour tout ce qu'ils m'ont apporté durant ces années. Leurs conseils et leurs encouragements m'ont permis de devenir ce que je suis aujourd'hui et je les remercie infiniment.

Je remercie le plus sincèrement possible les membres de jury : le Pr. Mohammed Rida laouar, Pr. Nacira Ghoulmi, Dr. Nabiha Azizi et le Dr. Makhoulf Dardour pour leur attention et l'intérêt porté à cette thèse et qui ont eu l'obligeance d'accepter de juger ce travail. Comme je présente mes sincères remerciements à tous les membres de LAMIS soit les professeurs, les docteurs et les doctorants qui ont contribué à élaborer ce laboratoire de recherche et qui ont su rendre mon travail agréable à travers leurs conseils, leur simple présence ainsi que les différentes journées doctorales et les conférences élaborées par eux.

Je souhaite exprimer ma plus profonde reconnaissance à Pr. Phillippe Roose pour l'aide précieuse qu'il m'a apportée durant ma présence au sein du laboratoire LIUPPA de l'Université de Pau et des Pays de l'Adour. Il a su diriger mes travaux avec beaucoup d'efforts, de tact et d'intérêt. Il m'a toujours accordé généreusement le temps nécessaire pour partager avec moi ses idées et sa grande expérience. Ainsi mon collègue Sahraoui Abdelatif qui m'a aidé sans hésitation dans mon projet de recherche et qui fut présents à mes côtés lorsque j'en avais besoin et je le remercie infiniment pour le travail collaboratif qu'on a effectué ensemble.

Les mots me manquent pour remercier à sa juste valeur mes amis : Hiouani Rima, Hussam Ateeq, Aysa ouasel, Mahmoud al Quaysi, Ziad al Jariri, Nasri Ahlem, Berramla Karima ainsi que leurs familles pour leurs soutiens moral et psychologique indispensables pour maintenir ce projet à flot au travers des aléas de la vie et pour avoir cru en mes capacités intellectuelles et à mon sens de l'organisation pour le réaliser.

Je ne saurais terminer sans souligner le soutien amical et chaleureux de mes copines et copains de tous les jours qui m'ont soutenue durant ce parcours doctoral. Je m'abstiens de les nommer tellement la liste est longue. Je nommerai tout de même mes collègues du notre laboratoire Gherari Manel, Benatia Imene et Marzoug Soltane que je remercie spécialement pour ses conseils et ses appuis.

TABLE DES MATIÈRES

Liste des Figures	10
Liste des Tables	11

CHAPITRE

INTRODUCTION GÉNÉRALE _____ PAGE 13

1	Contexte du travail et problématique	13
2	Objectifs et démarche	14
3	Structure de la thèse	15

CHAPITRE 1

LES APPLICATIONS MOBILES SENSIBLES AU CONTEXTE _____ PAGE 18

	Introduction	19
1	Les applications mobiles	20
	1.1 Historique et définitions	20
	1.2 Les plateformes de téléchargements	20
	1.3 Etude de marché : statistiques	22
2	Les appareils mobiles	23
	2.1 Historique et définitions	23
	2.2 L'hétérogénéité des appareils mobiles	24
	2.3 Les ressources limitées	28
3	Le contexte dans un environnement mobile	30
	3.1 Définition du contexte	30
	3.2 Vers la notion d'adaptation	32
	3.3 Modélisation du contexte	34
	Conclusion	37

CHAPITRE 2

LA COMPOSITION POUR LE DÉVELOPPEMENT D'APPLICATIONS MOBILES _____ PAGE 38

	Introduction	39
--	--------------	----

1	L'Ingénierie Dirigée par les Modèles (IDM) pour le développement mobile	40
1.1	Les principes généraux de l'IDM	40
1.2	L'intérêt d'une approche IDM	44
1.3	Pourquoi l'IDM pour le développement mobile ?	45
2	Le développement mobile par la composition de l'existant	47
2.1	Représentation du noyau fonctionnel	47
2.2	Processus de composition	49
2.3	Composition de services/composants	54
2.4	Composition basée composants VS composition orientée services	59
3	Les approches de composition : Synthèse	63
	Conclusion	66

CHAPITRE 3

CAMAP : UN PROCESSUS POUR LA COMPOSITION D'APPLICATIONS MOBILES PAGE 67

	Introduction	68
1	Scénario de composition : ShopReview Mobile App	69
2	CAMAP : Le contexte global	71
3	CAMAP : Étapes et déroulement	72
4	La description du contexte de composition	74
4.1	La nécessité de décrire le contexte	74
4.2	Modélisation du contexte de l'appareil mobile cible	75
4.3	Modélisation du contexte des objets de composition	76
5	Modélisation de l'application mobile composite	78
5.1	La représentation fonctionnelle	78
5.2	La description architecturale	79
6	Mécanisme opératoire du processus : passage entre modèles	83
6.1	Du modèle fonctionnel au modèle fonctionnel raffiné	83
6.2	Du modèle fonctionnel raffiné au modèle architectural	85
6.3	Du modèle architectural au modèle architectural détaillé	86
	Conclusion	88

CHAPITRE 4

IMPLÉMENTATION ET EXPÉRIMENTATION PAGE 89

	Introduction	90
1	Implémentation des méta-modèles proposés	91
1.1	Méta-modèles pour la représentation fonctionnelle	91
1.2	Méta-modèles pour la description architecturale	94

2	Implémentation du processus de composition	101
2.1	Extraction des informations contextuelles	103
2.2	Paramétrage des règles ATL	104
2.3	Ordonnancement automatique des règles ATL	105
3	Résultats et Évaluation	106
3.1	Les résultats du processus de composition CAMAP	106
3.2	L'évaluation du processus de composition CAMAP	111
	Conclusion	114

CHAPITRE

CONCLUSION GÉNÉRALE PAGE 116

1	Contribution	116
2	Perspectives	117
	Production Scientifique et Projets de Recherche	119
	Bibliographie	121

LISTE DES FIGURES

FIGURE 1:	Travaux réalisés	15
FIGURE 2:	Plan de la thèse	16
FIGURE 1.1:	Organisation d'applications mobiles dans une plateforme de téléchargements	21
FIGURE 1.2:	Statistiques de téléchargements d'applications mobiles (Statista, 2015)	22
FIGURE 1.3:	Comparaison des plateformes de téléchargements <i>Google play</i> , <i>Apple store</i> et <i>Windows Phone</i> (Nomade, 2012 ; Statista, 2015)	23
FIGURE 1.4:	Caractéristiques des normes IEEE 802.11 du Wi-Fi (Gupta, 2015)	26
FIGURE 1.5:	Test d'autonomie (Rozier, 2015)	27
FIGURE 1.6:	Informations pertinentes des plateformes mobiles	28
FIGURE 1.7:	Le contexte d'interaction dans un environnement mobile	32
FIGURE 1.8:	Application tenant compte du contexte	33
FIGURE 1.9:	Application auto-adaptable	33
FIGURE 2.1:	Pyramide de modélisation de l'OMG (Minsky, 1965)	42
FIGURE 2.2:	Principe de transformation de modèles (De Castro et <i>al.</i> , 2011)	43
FIGURE 2.3:	Aperçu générale du processus de composition global proposé	50
FIGURE 2.4:	Orchestration de services (Lopez-Velasco, 2009 ; Hock-Koon, 2011)	55
FIGURE 2.5:	Chorégraphie de services (Lopez-Velasco, 2009 ; Hock-Koon, 2011)	56
FIGURE 3.1:	Scénarios de composition pour l'application mobile <i>ShopReview</i>	70
FIGURE 3.2:	Contexte global du processus proposé (CAMAP)	71
FIGURE 3.3:	Le fonctionnement global du processus proposé (CAMAP)	73
FIGURE 3.4:	Positionnement de notre processus de composition dans le monde de transformations	73
FIGURE 3.5:	L'ontologie du contexte de l'appareil mobile	75
FIGURE 3.6:	L'ontologie du contexte de l'entité logicielle	77
FIGURE 3.7:	Profil d'exécution des entités logicielles	77

FIGURE 3.8:	Le méta-modèle CMA-FD	79
FIGURE 3.9:	Le méta-modèle CMA-FD	80
FIGURE 3.10:	Spécification d'un <i>médiateur exogène</i>	82
FIGURE 3.11:	Spécification d'un <i>médiateur endogène</i>	82
FIGURE 3.12:	Le moteur de transformation 1	85
FIGURE 3.13:	Le moteur de transformation 2	86
FIGURE 4.1:	Le méta-modèle CMA-FD exprimé en EMF (.ecore)	91
FIGURE 4.2:	Le méta-modèle H^2 CMA-AD.ecore exprimé en EMF (.ecore)	95
FIGURE 4.3:	Implémentation d'un moteur de passage (les choix technologiques)	101
FIGURE 4.4:	Transformation endogène via le langage ATL	102
FIGURE 4.5:	Transformation exogène via le langage ATL	102
FIGURE 4.6:	Règle SPARQL avec des paramètres génériques (code java)	103
FIGURE 4.7:	Méta-modèle de paramètres	104
FIGURE 4.8:	Modèle de paramètres	105
FIGURE 4.9:	La règle ATL <i>AddImpType</i>	105
FIGURE 4.10:	La règle <i>AddImpType</i> en code Java	106
FIGURE 4.11:	Le modèle fonctionnel de l'application mobile <i>ShopReview</i>	107
FIGURE 4.12:	Le modèle fonctionnel raffiné de l'application mobile <i>ShopReview</i>	108
FIGURE 4.13:	Le modèle architectural de l'application mobile <i>ShopReview</i>	110
FIGURE 4.14:	L'architecture détaillée de l'application mobile <i>ShopReview</i>	111
FIGURE 4.15:	Topologie de l'application <i>ShopReview</i> sur la plateforme OMNET++	111
FIGURE 4.16:	Un plan d'évaluation pour le processus de composition proposé	113

LISTE DES TABLES

TABLEAU 2.1: Taxonomie des transformations	44
TABLEAU 2.2: Un cadre de comparaison d'approches de composition	64
TABLEAU 3.1: Les types de composition traités par le processus de composition CAMAP	83
TABLEAU 4.1: Syntaxe graphique d'éléments architecturaux du modèle fonctionnel . .	92
TABLEAU 4.2: Syntaxe graphique des éléments architecturaux du modèle fonctionnel raffiné	93
TABLEAU 4.3: Pseudo code : Algorithme de la génération du modèle fonctionnel raffiné	94
TABLEAU 4.4: Syntaxe graphique des éléments architecturaux du modèle architectural	96
TABLEAU 4.5: Pseudo code : Algorithme de la génération du modèle Architectural . .	97
TABLEAU 4.6: Les éléments architecturaux de l'architecture détaillée de la CMA . . .	99
TABLEAU 4.7: Pseudo code : Algorithme de la génération du modèle architectural détaillé	99
TABLEAU 4.8: Les règles de passage déclenchées	109

CHAPITRE : INTRODUCTION GÉNÉRALE

1. CONTEXTE DU TRAVAIL ET PROBLÉMATIQUE
2. OBJECTIFS ET DÉMARCHE
3. STRUCTURE DE LA THÈSE

1 Contexte du travail et problématique

De nos jours, l'adoption massive des appareils mobiles pour effectuer nos tâches dans la vie quotidienne prouve la croissance exponentielle de l'utilisation des applications mobiles ainsi que leur développement. Selon *Gartner*, le nombre de téléchargements d'applications mobiles pourrai atteindre près de 269 milliards en 2017 (Oeillet, 2013).

Le domaine mobile présente de nouveaux défis dans l'ingénierie logicielle (Zhang and Adipat, 2005) car les différents appareils mobiles disponibles tels que les Smartphones, tablettes, etc. qui supportent ces logiciels mobiles sont caractérisés par plusieurs configurations différentes. Autrement dit, ils possèdent des caractéristiques matérielles et logicielles hétérogènes. En outre, les plateformes mobiles sont en évolution rapide y compris diverses capacités comme par exemple : GPS, les modes d'entrées/sorties, les capteurs, etc. (Cugola et al., 2014). Cela montre que les applications mobiles, contrairement aux applications classiques, doivent être spécialisées et donc leur construction doit être soumise à plusieurs contraintes. Ces contraintes sont exprimées par : les caractéristiques de déploiement (ex. le type de la plateforme d'exécution, la taille d'écran) et les ressources limitées offertes par les appareils mobiles (ex. des batteries à autonomie limitée, capacité de stockage limitée, service Wi-Fi avec une bande passante restrictive) ainsi que la variabilité de leur état d'exécution (ex. l'état actuel du service Wi-Fi est désactivé, le niveau actuel de la batterie est de 42%, l'espace de stockage restant est de 1.9 Go)

En plus, la prise en compte des ressources limitées lors du développement des applications mobiles empêche le déploiement de manière identique de la même application sur plusieurs appareils mobiles hétérogènes, à savoir la limitation de l'utilisation. Par conséquent, le développement logiciel pour l'environnement mobile doit faire face à plusieurs défis en raison des caractéristiques hétérogènes des appareils mobiles. Pour cela, les applications mobiles doivent être aussi adaptatives que possible avec leur environnement d'exécution, i.e. les conditions d'exécution de l'application mobile désirée doivent être conformes aux informations contextuelles de l'appareil mobile où elle sera déployée.

Sachant que la majorité des utilisateurs (85%) préfèrent utiliser les applications mobiles principalement plutôt que les sites web mobiles¹ où autres pour effectuer leurs tâches quotidiennes puisqu'elles sont plus pratiques, plus rapides et plus faciles à parcourir (Compuware, 2012), les différentes applications mobiles disponibles risquent ainsi de ne pas couvrir toutes les exigences des utilisateurs selon leurs demandes. Ce qui montre la nécessité de prendre en considération les exigences des utilisateurs lors du développement d'applications mobiles. Cela peut les aider à être plus productifs (Jones, 2013). A cet effet, la demande de nouvelles applications reste toujours en croissance exponentielle dans le but de combler les exigences de chaque utilisateur dans la vie quotidienne.

Les différentes issues citées ci-dessus motivent le besoin d'avoir un processus de composition pour construire des applications mobiles via la réutilisation de l'existant. L'objectif est d'atteindre les exigences des utilisateurs tout en considérant les informations contextuelles de l'environnement d'exécution, i.e. le contexte courant de l'appareil mobile ou l'application désirée sera déployée. Plusieurs travaux ont été proposés pour considérer cet axe de recherche parmi eux nous citons celui de (Rosa and Lucena Jr, 2011) et celui de (Furno and Zimeo, 2014). Les auteurs dans ces travaux de recherche traitent la tâche de composition en utilisant soit des composants soit des services selon les informations contextuelles de l'environnement d'exécution. Ces travaux limitent les objets de composition à un seul type et l'application composite sera dédiée seulement pour être exécutée dans un environnement spécifique, i.e. limiter l'utilisation de l'application.

1. Le terme *Site web mobile* réfère le mot *Mobile website* en anglais : "*est un site Web qui est spécifiquement conçu pour être consulté sur un appareil mobile*"

Pour cause des exigences des utilisateurs et aussi du contexte de l'appareil mobile cible, un développeur se trouve parfois obligé de combiner des entités logicielles hétérogènes. Cependant, et au meilleur de nos connaissances, aucun processus de composition des applications mobiles n'a été proposé pour considérer à la fois plusieurs facteurs comme : les exigences des utilisateurs, le type des entités logicielles constituantes, les caractéristiques de l'appareil mobile à utiliser et aussi l'état courant des ressources disponibles sur cet appareil. Cela ouvre la porte à la nécessité de tenir compte et essayer de résoudre des issues importantes comme l'hétérogénéité présentée par les appareils mobiles à utiliser ainsi que l'hétérogénéité des entités logicielles à composées lors de la composition.

2 Objectifs et démarche

Nous proposons dans ce manuscrit un processus de composition d'applications mobiles adaptatives (CAMAP : Composition of Adaptive Mobile Applications Process) dont l'objectif visé est double :

- Composer des applications mobiles en utilisant des entités logicielles pré-existantes indépendamment de leurs détails d'implémentation afin de tirer profit des services existants pour satisfaire les besoins des utilisateurs,
- Obtenir des applications mobiles adaptables à leurs environnements d'exécution, i.e. envisager les différentes informations contextuelles de l'appareil mobile lors de la composition.

Atteindre ces objectifs conduit éventuellement à résoudre un problème d'hétérogénéité entre les entités logicielles reliées que ce soit en terme de leurs formes d'implémentation (ex. un composant, un service, etc.) ou les données échangées entre elles. Pour cela, nous proposons une solution d'un point de vue architectural où nous permettons une composition endogène (i.e. relier des entités du même type) et/ou exogène (i.e. relier des entités du différents types) tout en intégrant des médiateurs pour assurer une composition cohérente et un échange valable des données. Donc, cela engendre un sous objectif de ce travail de recherche qui est la détection des points d'hétérogénéité lors de la composition tout en déterminant les différents médiateurs nécessaires, à savoir gérer et remédier les combinaisons hétérogènes.

Notre processus de composition est fortement basé sur des notions largement acceptées dans le domaine de l'ingénierie dirigée par les modèles (IDM) dont l'objectif principale est d'aider les concepteurs/développeurs en fournissant des niveaux d'abstractions élevés pour pouvoir gérer facilement la composition des entités logicielles hétérogènes et automatiser leurs implémentations. Il vise à effectuer la tâche de composition : (a) en commençant par l'identification de l'ensemble des fonctionnalités souhaitées tout en ignorant comment elles seront implémentées ainsi que les différentes dépendances entre ces fonctionnalités. Ces dernières décrivent l'ordre d'invocation des fonctionnalités identifiées ainsi que les données échangées entre elles dont l'objectif est d'avoir une application mobile fonctionnelle indépendante de n'importe quel domaine d'application. (b) ensuite, il vise à composer les entités logicielles contextuelles qui sont correspondantes aux besoins identifiés tout en incluant les adaptateurs nécessaires dans le cas d'une coordination hétérogène dont l'objectif est d'obtenir une application mobile composite dédiée à un contexte spécifique.

Vis-à-vis de nos objectifs qui regroupent à la fois le domaine de composition et celui de l'IDM, nos contributions portent principalement un aspect conceptuel et une autre pratique (cf. Figure 1). L'aspect conceptuel prend la forme de :

- Un langage de modélisation pour l'identification des fonctionnalités nécessaires à la construction de l'application mobile désirée en fournissant un méta-modèle nommé CMA-FD (Composite Mobile Application - Fonctionnel Description). Ce dernier permet donc de fournir une représentation fonctionnelle de l'application mobile composite.
- Un langage de modélisation pour une description architecturale à base d'entités hétérogènes de l'application mobile composite en fournissant un méta-modèle nommé H^2 CMA-AD (Heterogeneous or Homogeneous Composite Mobile Application - Architectural Description). Ce dernier permet de décrire une architecture logicielle exprimant l'application mobile composite comme une collection d'entités logicielles qui peuvent être homogènes ou hétérogènes.
- La modélisation du contexte en fournissant un modèle d'ontologie pour définir le contexte de l'appareil mobile et un autre pour décrire les entités logicielles et leurs conditions d'exécution afin d'assurer le déploiement correct et le bon fonctionnement de l'application composite désirée sur l'appareil mobile à utiliser.

En revanche, l'aspect pratique de nos contributions prend la forme d'une réalisation d'un prototype pour le processus de composition proposé CAMAP tout en fournissant un algorithme de composition dirigé par le contexte ; celui-ci regroupe trois sous-algorithmes de composition où chacun d'entre eux représente un ensemble de règles de passage. Ces différents passages visent à générer l'architecture détaillée de l'application mobile composite incluant tout adaptateur nécessaire à partir d'un modèle fonctionnel donné. Les sous-algorithmes proposés reposent sur l'utilisation des descriptions contextuelles et les représentations proposées.

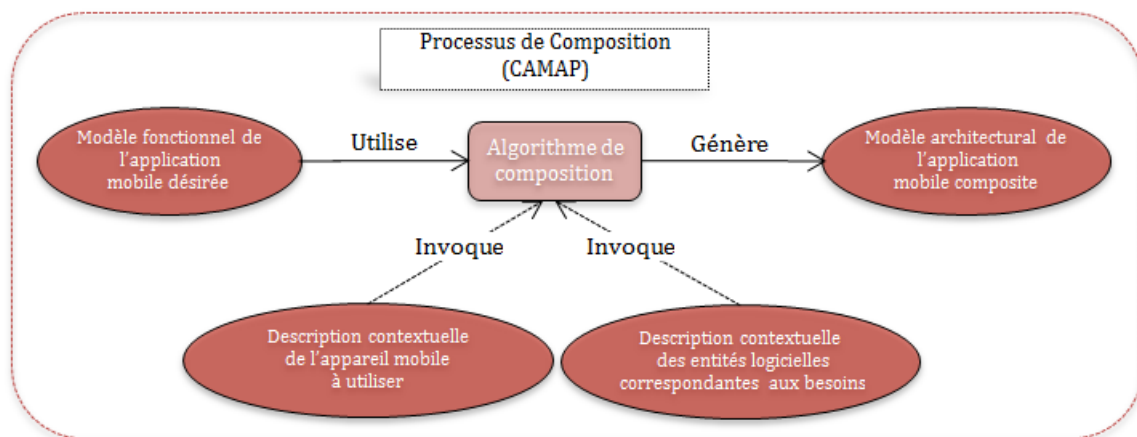


FIGURE 1 – Travaux réalisés

3 Structure de la thèse

Cette thèse est organisée en quatre chapitres commençant par une introduction générale décrit le contexte, les problématiques et les objectifs de notre travail de recherche et s'achève par une conclusion générale. Les chapitres de ce manuscrit décrivent respectivement l'état de l'art et nos contributions comme décrit la Figure 2 où les deux premiers chapitres définissent le cadre général et posent le socle bibliographique de notre travail de recherche tandis que les deux autres chapitres sont consacrés à nos contributions et ses réalisations :

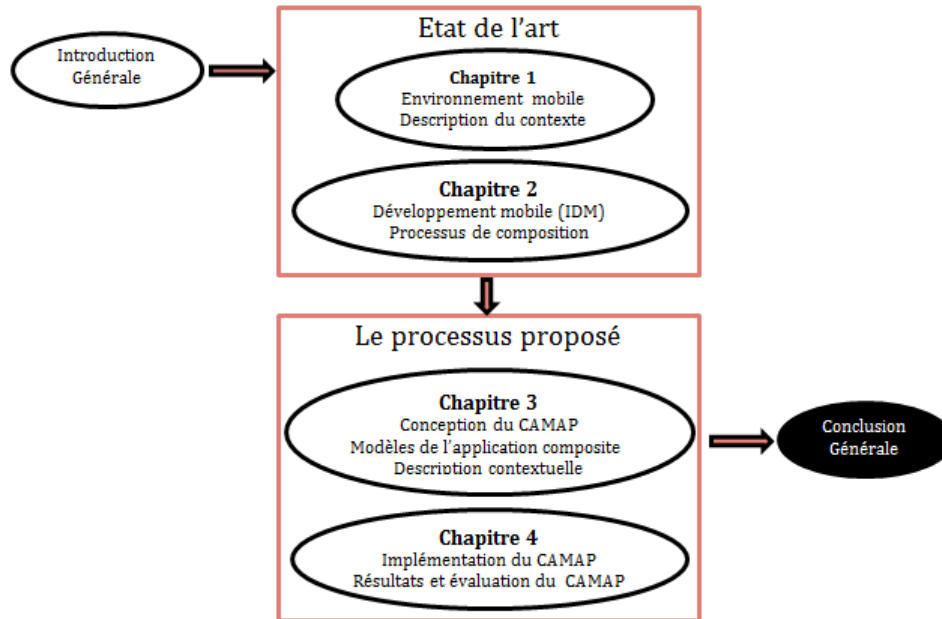


FIGURE 2 – Plan de la thèse

Chapitre 1 : Les Applications Mobiles Sensibles au Contexte

Dans ce chapitre, nous mettons l'accent sur les environnements mobiles en présentant les notions clés liées à ce domaine :

Dans un premier temps, des définitions générales concernant les applications mobiles et ses plateformes de téléchargements sont introduites ainsi qu'une étude de marché avec des statistiques concernant le nombre de téléchargements des applications mobiles est effectuée. Puis, des définitions générales concernant les environnements d'exécution de ces applications, ses différentes caractéristiques ainsi que les ressources limitées offertes par eux sont présentées. Par la suite, nous mettons l'accent sur la description du contexte commençant par la définition de ce concept, sa relation avec les applications mobiles et les appareils mobiles jusqu'à ce que nous arrivions à la notion d'adaptation et finalement comment nous pouvons le modéliser tout en présentant quelques approches de modélisation qui sont dédiées pour décrire le contexte.

Chapitre 2 : La Composition pour Le Développement d'Applications Mobiles

Ce chapitre traite le développement d'applications mobiles par le biais du mécanisme de composition. Il est constitué de 3 sections principales :

La première section porte sur l'ingénierie dirigée par les modèles pour le développement mobile. Dans un premier temps, nous présentons les principes généraux de cette ingénierie incluant la notion de modélisation et la notion de transformation de modèles. Puis, nous mettons l'accent sur l'intérêt de cette approche de développement. Nous concluons cette section en répondant à pleines de questions qui peuvent se poser autour de notre choix d'adopter cette ingénierie pour réaliser la solution proposée. Tandis que la deuxième section traite la construction d'applications mobiles par la réutilisation de l'existant. Dans un premier temps, nous introduisons les deux types de représentation les plus adoptées pour décrire le noyau fonctionnel d'une telle application. Puis, nous décrivons les tâches nécessaires pour obtenir une application composite tout en présentant dans un premier lieu le processus de composition global proposé selon les objectifs traités dans ce manuscrit.

Dû au fait que notre travail repose principalement sur la tâche de composition, nous détaillons dans les sous sections qui suivent tout ce qui concerne cette notion en relation avec les descriptions architecturales présentées précédemment. A ce stade, nous abordons les différents types de composition. Ensuite, nous introduisons un cadre comparatif entre la composition qui porte sur les services et l'autre qui porte sur les composants tout en mettant l'accent sur les problèmes d'hétérogénéité posés lors de la composition les multi-paradigmes émergents. La dernière section de ce chapitre présente une synthèse de travaux de recherches articulés autour de la problématique de composition d'applications mobiles. Cette synthèse a pour objectif de mettre en relief leurs apports et leurs manques afin de préciser les axes d'étude de notre proposition.

Chapitre 3 : CAMAP : Un Processus pour la Composition d'Applications Mobiles

Le chapitre 3 introduit le socle conceptuel du CAMAP en décrivant toutes les contributions élaborées :

Il introduit dans un premier temps un scénario de composition pour une application mobile nommée *ShopReview* dont l'objectif est de faire expliciter les motivations derrière notre travail de recherche et de préciser beaucoup plus les principaux problèmes qui sont traités dans ce manuscrit. Après, ce chapitre présente une section qui décrit et résume le contexte du processus proposée incluant toutes les notions clés abordées dans ce travail de recherche. Ensuite, il présente une autre section qui décrit la conception du processus de composition proposé en indiquant les points contributifs de ce manuscrit. Ainsi, nous mettons l'accent sur la relation entre ces contributions en expliquent le fonctionnement de ce processus de composition. Ce chapitre fait aussi le point sur la modélisation du contexte où nous proposons pour décrire le contexte de composition des modèles de description à base d'ontologies.

Dans la section qui suit, nous introduisons la représentation que nous proposons pour décrire l'aspect fonctionnel d'une telle application ainsi que la description architecturale à base d'entités logicielles hétérogènes que nous offrirons pour modéliser l'application mobile composite. Une dernière section dans ce chapitre se focalise sur le mécanisme opératoire de notre processus de composition qui est dirigé par le contexte et qui repose sur l'utilisation des représentations et des modèles d'ontologies proposés.

Chapitre 4 : Implémentation et Expérimentation

Le chapitre 4 décrit la mise en œuvre de nos contributions à travers le développement d'un prototype pour le processus de composition proposé (CAMAP) tout en présentant les résultats des expérimentations. Ce chapitre est organisé en trois sections :

La première section met en évidence la mise en œuvre des différents méta-modèles proposés. Ainsi, elle présente les différents sous-algorithmes de composition qui sont proposés pour faire passer l'application mobile à plusieurs niveaux d'abstractions. Dans la deuxième section nous décrivons les technologies adoptées pour réaliser le passage entre les différentes représentations de l'application mobile composite ainsi que les différentes méthodes proposées pour avoir un processus de composition générique. Dans la dernière section, nous montrons d'abord les résultats obtenus en appliquant notre mécanisme de composition selon deux scénarios proposés pour construire l'application mobile *ShopReview*. Ensuite, nous présentons un plan d'évaluation basé sur les scénarios décrits afin d'illustrer l'applicabilité et l'efficacité de notre processus ainsi que sa capacité à gérer l'hétérogénéité des applications mobiles à composer.

La thèse s'achève par une conclusion générale présentant les conclusions constatées où les points contributifs soulignés avec un retour sur nos objectifs et ouvrant cette thèse vers de nouvelles perspectives.

Les Applications Mobiles Sensibles au Contexte

SOMMAIRE: CHAPITRE 1

Introduction	19
1 Les applications mobiles	20
1.1 Historique et définitions	20
1.2 Les plateformes de téléchargements	20
1.3 Etude de marché : statistiques	22
2 Les appareils mobiles	23
2.1 Historique et définitions	23
2.2 L'hétérogénéité des appareils mobiles	24
2.3 Les ressources limitées	28
3 Le contexte dans un environnement mobile	30
3.1 Définition du contexte	30
3.2 Vers la notion d'adaptation	32
3.3 Modélisation du contexte	34
Conclusion	37

INTRODUCTION

Les utilisateurs sont de plus en plus mobiles et veulent pouvoir accéder à leurs systèmes d'information quel que soit le lieu où ils se trouvent. Pour ce faire, plusieurs appareils mobiles ont été développés pour soutenir ces systèmes afin de répondre aux exigences des utilisateurs en mobilité. Les constructeurs de ces appareils mobiles, notamment les Smartphones, fournissent des appareils ayant tous des architectures matérielles et logicielles différentes. Cependant, le développement d'applications mobiles se heurte au frein majeur qui est l'hétérogénéité des appareils mobiles et les ressources limitées offertes par eux. Pour cela, avant de se lancer dans le développement d'une application mobile, il est important de connaître les caractéristiques techniques et logicielles de l'appareil sur lequel elle va s'exécuter. Donc, elle doit être adaptée à son contexte d'utilisation afin de pouvoir répondre correctement aux besoins souhaités. Ce chapitre vise à faire une présentation des notions clés permettant de définir et comprendre le cadre général de notre travail de recherche à savoir les applications mobiles, les appareils mobiles et la description du contexte.

1 Les applications mobiles

1.1 Historique et définitions

Les applications mobiles sont parvenues dans les années 1990 (Horn et *al.*, 1999). Ces logiciels sont liées aux développements d'Internet et des télécommunications, des technologies agents (Benmammar and Krief, 2003), des réseaux sans fils et à l'émergence et la popularisation des terminaux mobiles. Avec le temps, le domaine mobile a connu une évolution considérable notamment depuis le 28 novembre 2007 où il est devenu un domaine de l'informatique à part entière (Perchat, 2015).

Le Petit Larousse en ligne nous apprend qu'une application est « Un programme ou un ensemble de programmes destinés à aider l'utilisateur d'un ordinateur pour traiter et effectuer une tâche précise ». Contrairement aux applications classiques qui sont dédiées à être exécutées sur des ordinateurs de bureau ou portables, les applications mobiles sont des logiciels applicatifs développés pour un appareil électronique mobile tel qu'un Smartphone, une tablette tactile ou encore certains ordinateurs qui fonctionne avec le système d'exploitation *Windows Phone*. Les applications mobiles sont développées sur des ordinateurs où le langage utilisé dépend du système sous lequel l'application sera exécutée. Les applications pour les appareils *Apple* sont développées par l'*objective C*, un langage principalement dédié à ces applications mobiles. Celles pour le système d'exploitation *Windows Mobile* sont développées par le langage *C#*. Le système *Android* utilise, quant à lui, un langage universel qui est le *Java*. Ce dernier est utilisé même pour les ordinateurs et le développement Web (JEE). Les applications mobiles ont pour objectif de faciliter la récupération d'informations où elles visent en particulier la productivité dans un environnement mobile.

Depuis peu les téléphones mobiles intelligents deviennent les appareils les plus courants et largement utilisés non seulement devant l'ordinateur personnel mais aussi le téléphone portable simple. A cet effet, les applications mobiles sont devenues de plus en plus complexes. Auparavant, une application mobile servait à effectuer des tâches simples ; elle se contentait par exemple d'acheter un ticket de métro, d'afficher des données provenant du web ou de vérifier un compte bancaire. Actuellement, en plus de cela, ces applications servent à combler des besoins plus complexes en communiquant avec le monde extérieur tels que les montres, les écrans de télévisions, etc. D'autres permettent le scan des codes-barres pour des produits que l'utilisateur veut acheter ou encore l'interaction avec des objets réels à travers la réalité augmentée. Ainsi, comme par exemple le calcul de la fréquentation cardiaque via des capteurs cardiaques placés sur les appareils mobiles.

Bien que les applications mobiles soient presque devenues infinies, le domaine mobile reste toujours une porte ouverte à des nouvelles innovations. Grâce à la créativité des développeurs, les exigences croissantes des utilisateurs ainsi que les évolutions technologiques de nombreuses inventions sont attendues : des applications commandées par la pensée, des applications liées à *Google Glass*, etc. Cela démontre la demande de nouvelles applications mobiles en tirant profit de celles qui sont déjà existantes, à savoir le besoin de composer et non pas de développer.

1.2 Les plateformes de téléchargements

A l'arrivée des Smartphones, téléphones portables embarquant un système d'exploitation complet et avec l'évolution du web, il est apparu un grand nombre d'applications mobiles. Afin de rendre accessible toutes ces applications développées via la multitude des appareils mobiles disponibles, les fournisseurs des systèmes d'exploitations mobiles se sont dotés chacun d'entre eux de ses propres magasins d'applications. Ces magasins portent aussi d'autres noms comme les marchés d'applications, les boutiques en lignes ou les plateformes de téléchargements.

Parmi elles nous trouvons : l'*App Store* de la plateforme d'*Apple/iOS*, le *Google Play* de la plateforme de *Google/Android*, le *Windows Phone* de la plateforme de *Microsoft/Windows phone*, *OVI* pour *Symbian* ou encore *BlackBerry App World*. Mis à disposition de l'utilisateur et à partir de ces magasins, celui-ci peut alors y accéder pour chercher et choisir d'installer plusieurs applications répondantes à ses besoins ; comme il peut aussi les mettre à jour, les noter, etc. Ces applications sont accessibles par *Internet* et leurs téléchargements peuvent être soit gratuits ou payants. Les plateformes mobiles ont pour objectif de créer un nouvel écosystème afin de rendre les applications mobiles pérennes. Ces applications mobiles sont de différents types et organisées selon des catégories où chacune d'entre elles regroupe celles qui sont similaires, à savoir elles ont les mêmes directives et possèdent des objectifs communs. Par exemple, la catégorie « Réseaux sociaux » regroupe « Tweeter, Facebook, LinkedIn, etc. ». La catégorie « Productivité » regroupe « les outils du bureau tels que Dropbox, Microsoft outlook, Antivirus, etc. ». La catégorie « Vie pratique » regroupe « Météo, Google maps, Google earth, etc. ». La Figure 1.1 illustre comment les applications mobiles peuvent être localisées et organisées dans une plateforme de téléchargements.



FIGURE 1.1 – Organisation d'applications mobiles dans une plateforme de téléchargements

Si nous nous référons au classement des applications les plus téléchargées, nous trouvons suivant des études déjà effectuées (Nimmith and Blandine, 2011) (Padellec, 2015) (AppBrain, 2015) que les utilisateurs expriment toujours un intérêt marqué pour les jeux, les réseaux sociaux, l'éducation et aussi la vie pratique pour les plateformes *iOS* et *Android* confondus. En particulier, l'*iPhone* a connu une forte progression des applications d'affaires et professionnelles tandis que *BlackBerry* a eu d'autres tendances vers des applications de livres électroniques, de musique et de divertissement.

Chaque appareil mobile est doté par un système d'exploitation spécifique et donc peut supporter des applications mobiles dédiées juste pour ce système et avec un nombre limités. Cependant, un utilisateur d'un tel appareil mobile peut avoir besoin d'y accéder et d'utiliser des applications mobiles fonctionnant sur d'autres types d'appareils, la question qui peut se poser est « Est ce qu'on peut avoir une application mobile fonctionnant sur tout type d'appareils mobiles ? ». Dans ce manuscrit nous traitons cette issue où nous proposons de générer plusieurs versions de la même application mobile dont l'objectif est d'augmenter la productivité dans le monde mobile et combler les besoins des utilisateurs en permettant le fonctionnement d'applications sur tout type d'appareils mobiles.

1.3 Etude de marché : statistiques

Les applications mobiles téléchargeables précisément sur les Smartphones ont le vent en poupe (cf. Figure 1.2). Plus de 7 milliards de ces logiciels ont été téléchargés en 2009. Et ce chiffre a été multiplié par 7 au cours des deux années suivantes, pour atteindre environ 50 milliards en 2011, d'après le cabinet *Chetan Sharma*. Par ailleurs, une étude réalisée par *MarketsandMarkets* a dévoilé que le marché des applications mobiles aura plus que triplé en 2015 (Nomade, 2012). En juillet 2013, la barre de 100 milliards d'applications téléchargées a été passée dans le monde. Selon la firme de recherche *Statista*, environ 180 milliards d'applications mobiles ont été téléchargées jusqu'en 2015. Selon les prévisions, le compteur devrait atteindre plus de 269 milliards de téléchargements en 2017 (Statista, 2015). Suivant un aspect financier, une étude de *Gartner* a dévoilé que les applications gratuites représentaient 91% des téléchargements en 2013 et que ce chiffre devrait atteindre 94,5% en 2017. Ainsi, les applications payantes ne rencontrent pas un grand succès (Padellec, 2015).

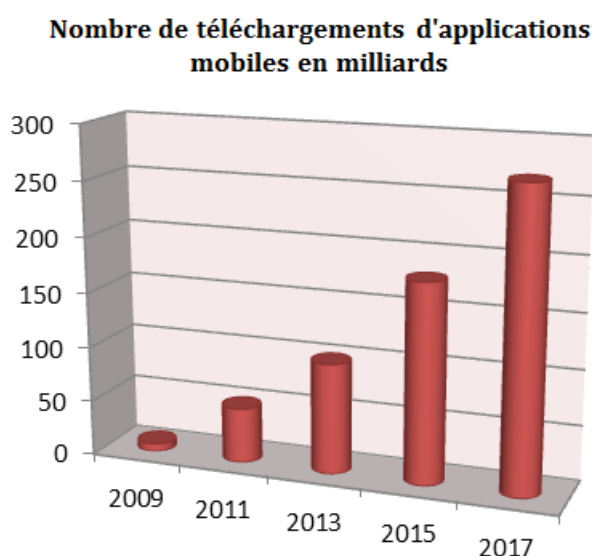


FIGURE 1.2 – Statistiques de téléchargements d'applications mobiles (Statista, 2015)

Au niveau des plateformes de téléchargements, l'*App Store* d'*Apple* a ouvert le 10 juillet 2008 avec 500 applications pour *iPhone* et *iPod Touch*. Deux ans et demi plus tard, plus de 350 000 d'applications dont 60 000 dédiées à l'*iPad* sont disponibles et 10 milliards de téléchargements ont été effectués. En 2013 sur cette plateforme la firme de *Cupertino* a déclaré qu'il y a plus de 50 milliards d'applications *iOS* téléchargées pour 800 000 applications. L'*App Store* propose aujourd'hui 1.4 millions de programmes avec 100 milliards de téléchargements (Statista, 2015).

Parmi les magasins applicatifs pour les terminaux mobiles qui ont fait leurs preuves en plus de celle d'*Apple*, nous trouvons *Google Play* de la plateforme *Google* qui a été lancé trois mois et demi après l'*App Store*. Cette plateforme de téléchargements a connu plus de 2,77 milliards de téléchargements avec plus de 230 000 programmes en 2010. En 2013, le nombre de téléchargements d'applications sur cette plateforme a été augmenté jusqu'à plus de 48 milliards. Aujourd'hui, plus de 1,8 millions d'applications sont présentées dans le *Google Play Store* avec 170 milliards de téléchargements.

A côté de ces deux magasins les plus connus et les plus couramment utilisés actuellement, nous citons aussi *Windows Phone Store* de *Microsoft* : lancé le 21 octobre 2010 en Europe. Cette boutique en ligne qui est dédiée au téléchargement des applications pour le système d'exploitation *Windows Phone*, propose plus de 200 000 applications en décembre 2013. Le nombre d'applications disponibles sur le *Store Microsoft* ne cesse d'augmenter (plus de 150

000 applications *Windows* contre 600 000 pour *Android* et *Apple*) et cela favorise également le nombre de téléchargements sur cette plateforme. *Windows Phone* a enregistré en 2013 environ de 200 millions d'applications téléchargées par mois. Elle propose aujourd'hui 669 000 programmes avec 3 milliards de téléchargements. Ces nombres d'applications disponibles ne cessent d'augmenter d'année en année (Nomade, 2012).

La Figure 1.3 présente une comparaison entre les plateformes de téléchargements précédemment citées en fonction du nombre de téléchargements d'applications associé avec le nombre des programmes développés pour chacune d'entre elles dans l'année 2015. Cette comparaison montre que *Google Play* touche le plus grand nombre des utilisateurs. Par conséquent, il se démarque et gagne 84.4% du marché ce qui le place comme étant la plus grosse boutique d'applications au monde. Étonnement, *Apple Store* ne représente que 11.7%. Le très discret *Windows Phone* complète le tableau avec 2.9% de parts de marché (Geronimo, 2015). Par le biais de notre proposition dans ce manuscrit nous espérons à permettre aux utilisateurs de profiter de toutes les applications mobiles disponibles quelque soit le type de l'appareil mobile utilisé.

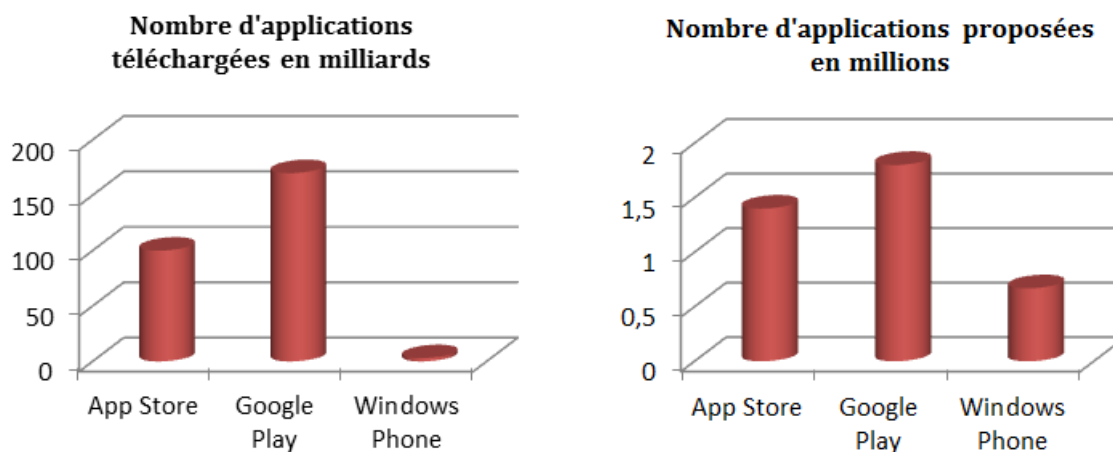


FIGURE 1.3 – Comparaison des plateformes de téléchargements *Google play*, *Apple store* et *Windows Phone* (Nomade, 2012 ; Statista, 2015)

2 Les appareils mobiles

2.1 Historique et définitions

De nos jours, la population mondiale est mieux équipée en Smartphones et tablettes qu'en ordinateurs de bureau. Le cabinet américain *Gartner* a publié une étude présentée dans un rapport intitulé « Utilisateur final prévision pour 2010 » affirmant qu'en 2013 dans le monde entier le mobile aura supplanté le PC comme moyen d'accès le plus couramment utilisé au web et voilà qu'aujourd'hui les appareils mobiles prédominent le monde de la technologie.

L'histoire de cette technologie a commencé avec l'apparition des terminaux mobile classiques qui sont définis comme des appareils informatiques portatifs utilisables de manière autonome lors d'un déplacement. Ils sont typiquement d'usage personnel dérivés des téléphones mobiles et permettent de prendre des photos, d'écouter de la musique, etc. La plupart des appareils mobiles sont connectés à *Internet* ce qui rend possible d'envoyer des courriels, d'accéder au Web et aux messageries instantanées.

Au fil du temps, la technologie mobile a connu un développement remarquable avec

l'apparition des appareils intelligents. Ces derniers, ne sont que des appareils mobiles évolués nommés Smartphones. L'arrivée de ces dernières a changé les habitudes d'utilisations où la saisie des données se fait le plus souvent par le biais d'un écran tactile d'un stylet ou rarement d'un clavier. Selon le principe d'un ordinateur, un Smartphone peut exécuter divers logiciels/applications grâce à un système d'exploitation dédiés notamment pour mobiles. Outre les appels, les messages SMS, la navigation sur web, les Smartphones sont enrichis par des applications plus complexes et dotés de plusieurs autres fonctionnalités : GPS, la reconnaissance vocale et de la synthèse vocale pour consulter les prévisions météo, la navigation, la messagerie vocale/visuelle et le dictaphone, etc.

Le premier Smartphone qui a révolutionné le domaine mobile est l'*iPhone* d'*Apple*. Lors de son lancement en 2007, il était le premier téléphone mobile à utiliser une interface *multi-touch* (ou *multi-tactile*) permettant une communication humain-ordinateur enrichie par des gestes intuitifs constitués de plusieurs points de contact.

Ces machines intelligentes ont marqué de leur empreinte tout au long du temps et d'une façon incrémentale. En 2013, le cabinet de recherche *Gartner* estime à 821 millions le nombre d'appareils mobiles de ce type vendus au cours de l'année 2012 dans le monde et à 1,2 milliard en 2013. Une autre étude a été effectuée par ce cabinet en 2013 montrant qu'en 2014, le nombre des Smartphones et tablettes vendus dans le monde s'élèveront à 2.1 milliards d'appareils contre 300 millions de PC vendus. Ainsi, selon l'estimation de l'équipementier réseaux *Cisco*, le monde comptera 25 milliards d'appareils connectés en 2015 et 50 milliards en 2020.

2.2 L'hétérogénéité des appareils mobiles

La construction d'applications mobiles est soumise à plusieurs contraintes liées à l'hétérogénéité des appareils mobiles que ce soit du point de vue matériel ou logiciel. Avant de se lancer dans le développement d'une application mobile, il est important de savoir les spécifications techniques de l'appareil mobile pour lequel elle sera dédiée.

■ Les caractéristiques matérielles

Aujourd'hui, les Smartphones et tablettes sont fabriqués par des constructeurs différents (ex. les plus populaires sont *Samsung*, *Apple* et *LG*) où chacun d'entre eux fournit des architectures matérielles différentes : des écrans différents en terme de la résolution et de la taille, des processeurs différents en terme de la vitesse, des caméras différentes en terme de la qualité des images obtenues, côté connectivité (GPS, Wi-Fi), les espaces de stockages, l'énergie de la batterie, etc.

- **La Technologie d'écran** : les appareils mobiles ont tous le point commun d'avoir un écran mais tous ces écrans ne se valent pas que ce soit par leur conception ou par la technologie qu'ils utilisent. Ces appareils disposent généralement des écrans tactiles pour les manipuler. Cependant, il existe deux types de technologie pour rendre une dalle tactile : (a) la dalle résistive qui adopte une sensation peu naturelle et déplaie souvent les utilisateurs. Cette technologie n'est plus utilisée aujourd'hui et nous la retrouvons uniquement sur les premiers appareils tactiles. Elle a été remplacée par les dalles capacitives, (b) la technologie capacitive est celle qui est utilisée depuis plusieurs années et a été connu du grand public notamment avec l'arrivée de l'*iPhone*. La dalle d'un écran est équipée de plusieurs éléments. Ces derniers sont les technologies d'affichage et parmi eux nous trouvons : LCD, PMOLED et AMOLED. Par rapport à l'écran LCD, AMOLED offre une meilleure qualité d'affichage avec un bon rendement énergétique en vertu de son mécanisme unique d'éclairage. De plus, les écrans se différencient par leur taille, leur définition et la résolution de l'écran. La taille se calcule en mesurant la diagonale de l'écran et est exprimée en pouce ou cm (un pouce équivaut à 2.54 cm) tandis que la

définition représente la dimension informatique d'un écran qui est le nombre de pixels en largeur et en hauteur. Contrairement à la définition d'un écran, la résolution est un rapport de densité. Elle représente le nombre de pixels affichés par pouce. C'est la densité de pixels qui est exprimée en Dpi (Dots Per Inch) ou Ppp (Points Par Pouce). La taille et la résolution d'écran d'un appareil mobile impactent la taille d'affichage du contenu visible sur ces appareils. Pour cela, les applications mobiles doivent être conçues pour une résolution spécifique. Pour garantir un meilleur affichage et surtout un propre fonctionnement pour les applications mobiles, il est recommandé de charger celles qui sont appropriées pour la taille et la résolution de l'écran de l'appareil mobile cible.

- **Capacité mémoire** : les Smartphones et tablettes actuels se contentent de 16 Go de mémoire au minimum et de 128 Go dans le meilleur cas ou grâce à une extension via carte micro SD. La mémoire de ces appareils commencera à saturer en sauvegardant les données et les applications souhaitées (i.e. à force d'installer des applications et d'ajouter des musiques, des photos ou des vidéos) sur l'espace de stockage restant parce qu'importe le mobile ou la tablette une partie de l'espace de stockage sera toujours occupée par le système d'exploitation installé sur cet appareil. La quantité de mémoire allouée aux applications et données de l'utilisateur se diffère d'un appareil à un autre et cela en fonction de la quantité de mémoire embarquée et de la taille du système d'exploitation installé. Elle est limitée que ce soit le type de l'appareil ou l'espace de stockage doté par celui-ci. Prenant l'exemple du Smartphone *Galaxy S6* doté de 32 Go de mémoire. Celui-ci n'aura qu'environ 25 Go d'espace libre. Contrairement à l'*iPhone 6* qui est doté de 16 à 64 Go de mémoire où le système d'exploitation *iOS* installé sur cet appareil occupe entre 4 et 8 Go en moyenne Go du mémoire selon la version du système installée. 25% de l'espace de stockage pourrait donc être consommé par l'installation de l'OS mobile d'*Apple*.
- **Connectique sans fil** : les Smartphones sont équipés de multiples interfaces réseaux sans fils qui leur permettent de répondre aux différentes demandes de communications et de réseaux. Parmi ces interfaces de connectivité nous trouvons : Bluetooth, Wi-Fi, GPS, GSM, GPRS, etc.
 - **Bluetooth** : est une norme de connexion sans fil peu énergivore, de courte portée et qui offre une bande passante de 1 Mb/s. Ce standard de transfert a pour objectif de transférer de fichiers d'informations, d'agendas électroniques, flux audio entre un téléphone et un casque, interconnexion sans fils des lecteurs mp3 à d'autres dispositifs pour le téléchargement, etc. Un appareil Bluetooth est caractérisé généralement par deux informations : la version de la norme et sa classe¹. Il définit aussi un certain nombre de profils d'applications (i.e. Bluetooth profiles²) permettant aux appareils Bluetooth de se connecter entre eux d'une façon spécifique. Donc, en plus de la classe et la version de la norme d'un appareil Bluetooth, les appareils mobile permettent aussi se différencier en terme de gestion de Profils Bluetooth. A titre d'exemple, l'*iPhone* et l'*iPad Touch* qui sont dotés par la version 8 d'*iOS* peuvent prendre en charge plusieurs Profils Bluetooth. Un tableau comparatif et illustratif présenté en (Apple, 2015) fournit des informations sur les profils pris en charge par chaque appareil.
 - **Wi-Fi** : Le Wi-Fi a vu le jour à la fin des années 90. Le terme Wi-Fi (pour Wireless Fidelity), largement popularisé aujourd'hui, se réfère à un ensemble de protocoles de communication sans fil (par onde radio) régis par les normes du groupe IEEE 802.11. Il couvre de nombreuses normes différentes qui ont toutes le préfixe 802.11 et un suffixe

1. La portée des équipements Bluetooth dépend en grande partie de la puissance en émission. Il existe trois classes de produits Bluetooth en fonction de la puissance émise. Les produits de la classe 1 émettent avec une puissance maximum de 100 *milliwatts* (10 mètres de portée), la puissance habituelle des produits de la classe 2 est d'environ 2.5 *milliwatts* (une dizaine de mètres de portée), tandis que la puissance de la classe 3 est limitée à 1 *milliwatts* (la portée est de quelques mètres)

2. Un Profil Bluetooth correspond à une spécification fonctionnelle d'un usage particulier. Il a pour but de s'assurer une interopérabilité entre tous les appareils Bluetooth.

sous forme de lettre permettant de faire distinguer les normes entre elles. Pour les particuliers, il existe en tout cinq normes différentes : 802.11a/b/g/n/ac où chacune d'entre elles désigne une évolution par rapport à la précédente en terme de plusieurs critères : bande de fréquence, débit théorique, portée, congestion, largeur du canal. Comme nous pouvons le voir dans le schéma suivant, il existe énormément de débits théoriques différents. Dû au fonctionnement du protocole 802.11, le débit déclaré en pratique³ est beaucoup plus faible au débit théorique maximal. L'évolution de la version 802.11a fonctionne dans la bande des 5 GHz, c'est qui lui permet d'avoir un débit élevé pour l'époque, à 54 Mbps. Cependant, sa portée est faible puisque plus une fréquence est élevée et plus sa portée diminue. En revanche, l'avantage de la bande des 5 GHz est sa faible congestion (i.e. moins d'interférences) qui permet dans les faits d'atteindre des débits plus élevés et une meilleure stabilité de la connexion. Contrairement à la bande de 2,4 GHz, elle est congestionnée puisque de nombreux appareils l'utilisent pareillement tels que les appareils *Bluetooth*, les micro-ondes, etc. Les normes (g) et (n) sont les deux normes largement répandues au sein des périphériques réseau sans fil. Le Wi-Fi 802.11g utilise la bande de fréquence radio de 2,4 GHz et affiche un débit théorique de 54 Mb/s, 25 Mb/s en pratique. Le Wi-Fi 802.11n est une évolution de la version 802.11g pouvant utiliser les fréquences de 2,4 ou 5 GHz et susceptible d'atteindre théoriquement un débit de 300 Mb/s grâce notamment à la technologie MiMo⁴. Les normes (b) et (g) sont très proches l'une de l'autre où la deuxième est une légère évolution de la première : même débit que la version (a) mais avec une meilleure portée alors que version (ac) c'est la version la plus récente et la plus améliorée avec une bonne portée et une faible congestion. Avec un Smartphone compatible à cette version de norme et un unique flux spatial, i.e. une antenne, le débit est pratiquement 10 fois plus élevé qu'en Wi-Fi 802.11g, i.e. atteindre le 433 Mbps et jusqu'à 1300 Mbps pour 3 flux spatiaux. Les versions des normes les plus adoptées par les Smartphones sont b/g/n (ex. *iPad*, *Galaxy S4*, *S5*, *S6*, *S6 Edge*, etc.). En plus de ces versions, les générations les plus récentes des Smartphones comme *Galaxy S2* et *S6 Edge* supportent la dernière norme (ac) (cf. Figure 1.4).

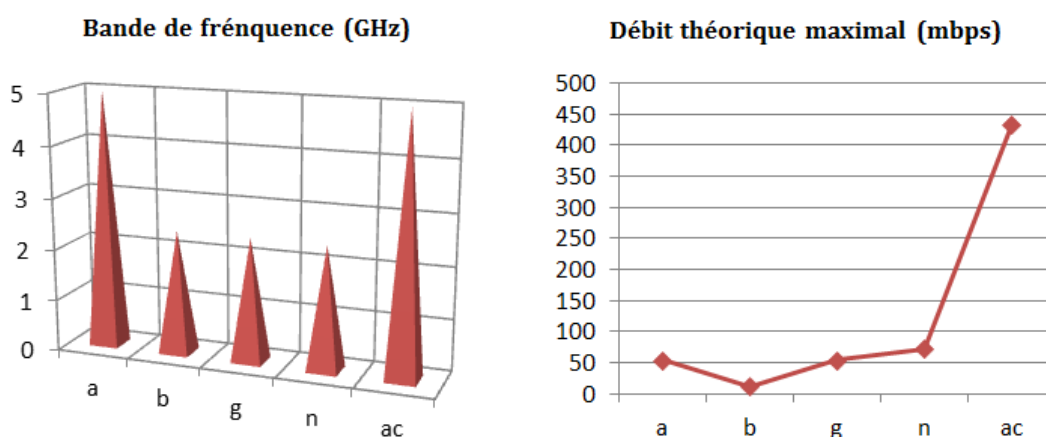


FIGURE 1.4 – Caractéristiques des normes IEEE 802.11 du Wi-Fi (Gupta, 2015)

- **L'autonomie** : c'est une des parties essentielles d'aspects, elle représente le potentiel de la batterie d'un Smartphone en terme de la consommation énergétique. L'autonomie diffère en fonction des tâches que nous effectuons sur les Smartphones. L'exécution d'une telle

3. Le débit pratique est fortement dépendant de la distance entre les appareils mais également des obstacles (comme les murs) qui se dressent sur le passage.

4. MIMO est l'acronyme de Multiple-Input Multiple-Output qui peut transmettre plusieurs signaux grâce à l'utilisation de plusieurs antennes (nécessite un émetteur et un récepteur compatible)

application mobile qui consomme beaucoup d'énergie implique que la vie de la batterie ne va pas durer beaucoup de temps. Donc c'est une question de taille de la batterie qui est mesurée en mAh. En dépendance avec cette capacité, l'autonomie est exprimée en fonction du temps par heures/minutes. La capacité de la batterie et donc l'autonomie dans les différents usages possibles (tel que la communication, la navigation web, la lecture vidéo, etc.) se diffère d'un Smartphone à un autre. A titre d'exemple, l'autonomie en communication, i.e. temps de conversation, de l'*iPhone6* dont la capacité de sa batterie 1810 mAh est de 12h 27 minutes, très éloignée des 26h du *Galaxy S5* ou sa batterie d'une capacité de 2800 mAh. Suivant un test d'autonomie effectué en (Rozier, 2015), durant 1 heure de vidéo YouTube 720 p en Wi-Fi avec la luminosité réglée à 200 cd/m² la batterie du *Galaxy S6* a perdue 14% de sa capacité. Un autre test consiste à activer le Wi-Fi et lancer une page web qui se rafraichit toute seule toutes les minutes, i.e. navigation sur web, tout en désactivant la veille automatique de l'écran et maximiser la luminosité de l'écran. Ce test est effectué sur les appareils suivant : *Samsung Galaxy*, *Sony Xperia Z3*, *iPhone 6*, *Nexus6* (cf. Figure 1.5). Donc, *Samsung Galaxy* est le plus autonome entre ces modèles.

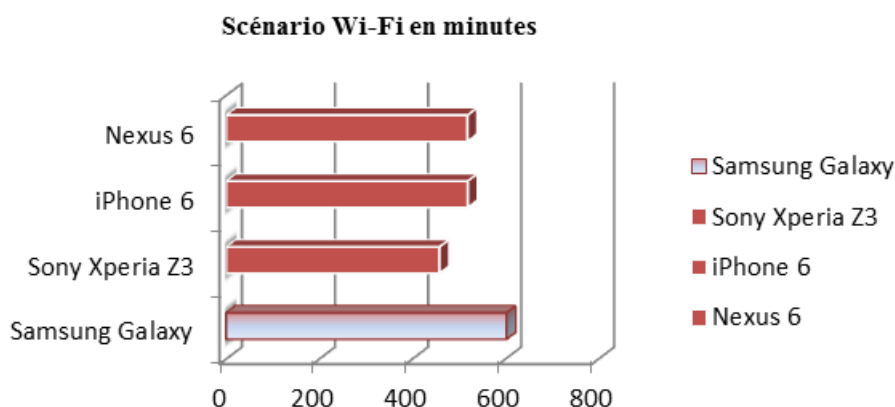


FIGURE 1.5 – Test d'autonomie (Rozier, 2015)

■ Les caractéristiques Logicielles

Les applications mobiles s'appuient sur un système d'exploitation qui permet la standardisation de leur fonctionnement et de leur développement. En plus de l'hétérogénéité engendrée par la multitude des fabricants de Smartphones, les plateformes ciblées (OS) installées sur ces appareils et qui sont dédiées pour faire exécuter les applications mobiles développées sont fournies par plusieurs fournisseurs : *Android* de *Google*, *iOS* d'*Apple*, *Windows Phone* de *Microsoft*, *Symbian* de *Nokia*, *BlackBerry OS* de *RIM*, etc. Ces OS présentent plusieurs di-similarités où chacun d'entre eux utilise son propre langage de programmation (ex. le *Java* pour *Android*, l'*objectif C* pour *iOS*, etc.) ainsi possède un environnement de développement spécifique (ex. *Eclipse* pour *Android*, *X Code* pour *iOS*, etc.). Cette différence réduit significativement la possibilité d'avoir des développeurs qui maîtrisent toutes ces plateformes ce qui influence sur la productivité des applications mobiles et augmente le coût de développement en terme de temps et d'argent. Dans le travail de recherche actuel nous visons à faciliter la tâche de développement d'applications mobiles par le biais du mécanisme de transformation *Modèle vers Code*.

En 2005, *Palm*, *Symbian* et *BlackBerry* ont dominé le marché. Un an plus tard, *Microsoft* les avait dépassé tous les trois. Sur la période 2008-2010, *BlackBerry* a réussi encore une fois à se hisser en première position avant de céder du terrain devant *Android* en 2011. En revanche, bien qu'il représente un acteur majeur du marché, *iPhone* n'a jamais été leader. A ce jour, *Android* et *iPhone* sont les deux plateformes principales du marché des smartphones :

- **Android** : *Android* (Hashimi et al., 2011) a été spécifié par un consortium : Open Handset Alliance (OHA) créée en novembre 2007 à l'initiative de *Google* dont l'objectif de créer un système complet, ouvert et gratuit pour les mobiles (i.e. distribué en open source). Aujourd'hui, c'est le leader en matière de système d'exploitation mobile. Il occupe 70% du marché européen. Le grand avantage d'*Android* est que cette plateforme n'a pas été conçue uniquement pour les Smartphones mais aussi pour être embarquée dans d'autres appareils tels que : des tablettes, des montres et même des voitures, etc. Pour ce faire, elle est basée sur un *Kernel linux* qui peut être installé sur n'importe quel appareil. Les entreprises de développement d'applications mobiles publient leurs applications sur les magasins dédiés aux systèmes d'exploitation mobiles. Donc, afin de pouvoir bénéficier d'applications mobiles d'*Android* il faut les télécharger du magasin *Google Play* comme il est pratique aussi de les rechercher sur les magasins d'applications officiels.
- **iOS** : *iOS* (Neuburg, 2013) est le système d'exploitation d'*Apple* développé pour l'*iPhone*, *iPod Touch*, *iPad* et *Apple Watch*. Contrairement à *Android*, *iOS* est propriétaire. En effet, l'intégralité du code source d'*iOS* n'est d'ailleurs pas disponible. Le portail *App Store* est dédié à l'exposition de toutes les applications développées pour ce système d'exploitation. Il est impossible de tirer profit d'une application en passant par un autre magasin de téléchargements que l'*App Store*.

Dans la Figure suivante (cf. Figure 1.6) nous présentons toutes les informations pertinentes pour chaque plateforme sous forme d'une petite comparaison entre elles selon trois axes : l'environnement de développement d'applications appropriées, le langage de programmation utilisé pour développer ces applications et les magasins dédiés pour stocker les applications développées.

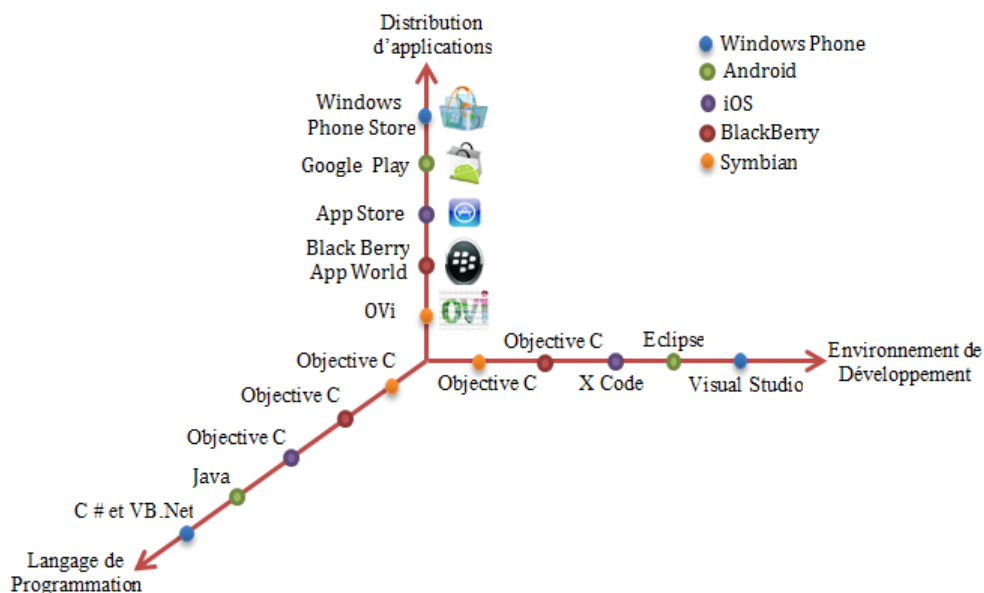


FIGURE 1.6 – Informations pertinentes des plateformes mobiles

2.3 Les ressources limitées

La majorité des appareils mobiles sont conçus pour être petits en terme de mémoire, de la taille de l'écran d'affichage et de la puissance de la batterie, etc. Cependant, les performances d'un téléphone mobile sont plus modestes, même si la tendance actuelle des constructeurs

est de proposer des appareils mobiles tout-en-un de plus en plus puissants disposant des mêmes fonctionnalités qu'un PDA⁵. Donc, en dépit de l'accroissement de la puissance de ces appareils (i.e. produire des batteries de plus en plus puissantes, des capacités de stockage de plus en plus grandes) et quelque soit leurs performances, leurs capacités restent limitées face à celles des ordinateurs de bureau et portables. Certaines limitations techniques inhérentes à leur taille d'écran, à la puissance des processeurs, aux interfaces de connectivité et à la capacité de mémorisation influencent le comportement des applications mobiles. Dans cette section nous discutons ces ressources limitées qui sont offertes par les appareils mobiles et qui doivent être prises en compte lors du développement d'applications mobiles (Meier, 2010). Comparés aux ordinateurs de bureau ou portables, les appareils mobiles ont :

- De petits écrans avec de faibles résolutions,
- De connexions réseaux moins fiables,
- Des capacités de stockage permanent limitées,
- Une RAM limitée,
- Des batteries à autonomie limitée,
- Une puissance processeur plus faible.

Les écrans des appareils mobiles à petite taille et avec une résolution faible limitent grandement le nombre d'informations qui peuvent être affichées à l'écran. Ils nous rendent se contenter juste de l'essentiel ce qui n'est pas forcément un mal. Les nouvelles générations de téléphones améliorent ce point en proposant de nouveaux écrans avec des résolutions incroyablement meilleurs permettant d'afficher beaucoup d'informations facilement compréhensibles. Cependant, en tenant compte du grand nombre d'appareils mobiles existants et afin de garantir un meilleur affichage du contenu des applications installées sur eux ; il est recommandé de prévoir dès la conception les caractéristiques les plus favorables, i.e. la taille d'écran recommandée, parce que l'interface de l'application mobile doit être adaptée à la taille de l'écran de l'appareil utilisé. Afin de s'assurer que l'application mobile est élégante et pourra se présenter bien sur pas mal d'appareils mobiles, il est possible de la concevoir pour de petits écrans mais également de prévoir que son interface s'adaptera à de plus grandes tailles en utilisant des techniques d'optimisation d'interfaces.

Avec les services de connectivité et plus particulièrement le service Wi-Fi, les déconnexions restent fréquentes, i.e. connectivité instable, et rien ne garantit que l'utilisateur sera toujours dans une zone couverte. De plus, la connexion sans fil utilisée par les appareils mobiles a généralement une bande passante limitée par rapport à une connexion filaire. Cependant, la limitation de la bande passante, la vitesse relativement lente et la non fidélité de cette technologie par rapport à un réseau câblé représentent des obstacles et donc influencent sur l'exécution des applications sur ces appareils. Pour cela, il est préférable de prévoir ces limitations dès la conception et de supposer lors du développement d'applications pour Internet que la connexion réseau sera lente, intermittente, coûteuse et de s'assurer également que l'application développée pourra gérer la perte de connexion.

Les progrès dans les mémoires flash et les disques SSD (Solid State Drive) ont entraîné un accroissement considérable des capacités de stockage des appareils mobiles, i.e. diminuer l'utilisation des disques flash ou des cartes SD et donner la possibilité de stocker même les collections mp3, photos, et jeux sur l'appareil. Les disques optiques de ces appareils proposent plus de 32 Go d'espace alors qu'en revanche le Téraoctet est maintenant courant sur les disques pour PC. Dû au fait que les applications mobiles doivent être installées dans la mémoire interne

5. PDA est un appareil numérique portable, souvent appelé par son sigle anglais « PDA » pour « Personal Digital Assistant »

et non sur des cartes SD externes ainsi que l'essentiel de l'espace de stockage d'un mobile sera probablement utilisé pour stocker de la musique et des films, les appareils mobile offrent à nos applications une capacité de stockage limitée.

Les appareils mobiles offrent des batteries qui ne sont pas assez performantes avec une énergie limitée. La manipulation des appareils mobiles consomme cette énergie avec le temps et tout dépend de l'utilisation que nous en faisons. La consommation énergétique d'un Smartphone reflète par la quantité d'énergie utilisée par le Smartphone afin de faire fonctionner les services qu'il propose. A titre d'exemple, les jeux ou les lecteurs vidéo videront bien plus vite la batterie que la consultation des sites web. Ce qui démontre que l'exécution d'une telle application ou d'une tâche quelconque nécessite forcément la consommation d'une proportion x % d'énergie disponible. Cela prouve que l'exécution des applications mobiles dépend fortement de l'énergie de la batterie disponible actuellement en fonction de sa capacité.

En terme de la puissance du processeur, les appareils mobiles les plus performantes dont la vitesse de ces processeurs dépassent le *Gigahertz*. La puissance du processeur a une dépendance directe avec l'exécution d'une telle application mobile où cette dernière ne doit pas lancer un traitement de plusieurs centaines de milliers d'interactions sur un appareil ayant un processeur très lent.

3 Le contexte dans un environnement mobile

Le contexte est considéré comme un aspect primordial pour le développement d'applications issues de trois domaines de l'informatique qui jouissent à présent d'une attention importante de la part de la communauté informatique : l'informatique ubiquitaire (ou ambiante) (Weiser, 1991), l'informatique pervasive (Saha and Mukherjee, 2003) et l'informatique sensible au contexte qui est connu sous le terme anglais "context-aware computing" (Chen and Kotz, 2000). Lopez (Lopez-Velasco, 2009) a déclaré que l'adaptation au contexte dans un système ubiquitaire a pour objectif de fournir aux utilisateurs des applications adaptées à la diversité des dispositifs qu'ils ont à leur disposition, tandis que l'objectif de l'adaptation au contexte dans un système pervasif est d'utiliser le plus de ressources possibles localisées autour de l'utilisateur afin de lui fournir une application adaptée sans qu'il intervienne dans le processus. Contrairement au paradigme de l'informatique mobile dans lequel les applications peuvent découvrir et utiliser l'information contextuelle, l'adaptation au contexte sert à fournir aux utilisateurs des applications adaptées qui prennent en compte le changement du contexte d'utilisation inhérent au déplacement de l'utilisateur.

Notre travail inclus dans ce cadre de recherche : l'adaptation au contexte. Plus précisément, il se focalise sur les applications mobiles déployées sur des Smartphones, tablettes, etc. qui font partie du domaine de l'informatique sensible au contexte. Dans cette section nous allons aborder la notion du contexte dans cet axe. Présentons dans un premier temps des définitions générales concernant cette notion, passons vers la notion d'adaptation. Après, et vu qu'un contexte quelconque nécessite une spécification nous présentons quelques modèles et approches de modélisation dédiés pour la description du contexte.

3.1 Définition du contexte

Dans les deux dernières décennies, avec l'évolution de l'informatique mobile, la notion du contexte est également devenue très importante dans la recherche. Dans cette section, nous nous intéressons aux définitions de deux notions précieuses *contexte* et *context-aware* qui existent dans la littérature.

■ Qu'est-ce qu'un contexte ?

De nombreux travaux issus de l'informatique sensible au contexte servent à donner une définition du contexte afin d'élaborer un socle pour la tâche d'adaptation. Il existe en fait à peu près autant de définitions du contexte qu'il y a d'équipes de recherche. Malgré ça, une définition à la fois générique et pragmatique de la notion de contexte reste encore manquante. Plus précisément, les chercheurs dans le cadre de recherche de l'adaptation au contexte n'ont pas encore abouti à une définition des paramètres constituant le contexte. En fait, toutes les définitions déjà présentées sont soit très abstraites ce qui rend la modélisation du contexte très difficile, soit très relatif à un domaine particulier. Les auteurs dans (Tigli and Lavirotte, 2006) ont déclaré quatre familles de contexte où chacune d'entre elles apporte ses propres définitions du contexte : la notion du contexte généralisé, la notion du contexte géolocalisé, la notion du contexte unifié et la notion du contexte environnemental. Nous nous intéressons aux définitions incluses à la catégorie du contexte environnemental dû au fait qui est celle la plus adoptée dans le domaine de l'informatique sensible au contexte. En 1995, Brown a défini le contexte comme un ensemble d'éléments de l'environnement de l'utilisateur (Brown, 1995). Deux années après, les auteurs (Brown et al., 1997) dans ajoutent à la première définition des entités telles que l'heure, la saison, la température, l'identité et la localisation de l'utilisateur. En 2000, les auteurs dans (Chen and Kotz, 2000) ont déclaré que le contexte détermine soit le comportement d'une application soit les événements issus de l'application et qui intéressent l'utilisateur. Par conséquent, ils définissent le contexte comme étant un ensemble d'états et de configurations observées de l'environnement.

Des travaux très pertinents sont fournis en 1999-2001 par Dey et Abowd (Abowd et al., 1999 ; Dey, 2001) afin d'améliorer la compréhension des notions du *contexte* et *context-aware* tout en faisant une synthèse des travaux existants à l'époque. Grâce à ces travaux, les auteurs ont donné leur propre définition pour qu'elle soit celle qui généralise les précédentes, soit la plus complète et la plus adoptée par les chercheurs dans leurs travaux (ex. nous citons comme exemple (Bettini et al., 2010 ; Emmanouilidis et al., 2013). Donc, ils ont défini le contexte comme suivant :

« Le contexte est toute information qui peut être utilisée pour caractériser la situation d'une telle entité. Une entité est une personne, un lieu ou un objet qui est considéré pertinent pour l'interaction entre l'utilisateur et l'application, y compris l'utilisateur et l'application. »

Contrairement aux travaux précédents (Schilit and Theimer, 1994 ; Brown et al., 1997) qui se contentaient d'énumérer des éléments de contexte tels que (la position, l'identité des personnes et des objets voisins, les changements de voisins, la saison, la température, le temps, etc.), la définition donnée par Dey et Abowd permet de juger si une information donnée fait partie du contexte ou non. De plus, le contexte est considéré par rapport à chaque application (ou activité). Dans le travail de recherche actuel nous appuyons cette définition pour caractériser le contexte de composition de l'application mobile où nous sélectionnons dans un premier temps les entités du contexte (l'appareil mobile cible et les entités logicielles à composer), après nous présentons toutes les informations pertinentes qui peuvent être utilisées pour définir la situation de d'une telle entité.

■ Que veut dire context-aware ?

Le terme anglais "aware" signifie conscient. La notion *context-aware application* qui reflète la notion d'une *application sensible au contexte* en terme français exprime qu'une application peut être consciente de son contexte et réagir en conséquence. En 1992, un système de localisation des employés dans les pièces d'un bâtiment utilisant des badges actifs (Want et

al., 1992) était le premier travail de recherche en informatique qui intègre le *context-aware*. La notion *context-aware computing* qui veut dire en français *l'informatique contextuelle* est abordé pour la première fois dans le travail de (Schilit and Theimer, 1994). Ces auteurs ont défini l'informatique contextuelle comme la capacité d'une application mobile à découvrir et à réagir à des changements dans l'environnement. La définition du contexte donnée par ces auteurs porte sur des variables bien déterminées comme la localisation, l'identité des personnes proches et des objets ainsi que les changements dans ces objets. Cette définition est apparue trop spécifique pour Dey et Abowd dû au fait qu'il est impossible pour un élément pris dans l'environnement de savoir avec certitude qu'il fait partie du contexte ou non. D'après eux, il est impossible d'énumérer de façon exhaustive toutes les variables d'environnement qui peuvent influencer sur le contexte.

Ces auteurs proposent en 1999 (Abowd et *al.*, 1999) qu'un système soit sensible au contexte s'il utilise des informations du contexte pour doter l'utilisateur d'informations et/ou des services pertinents ou la pertinence dépend de la tâche de l'utilisateur. De nouveau, ces définitions ne sont pas encore assez suffisantes pour inclure tous les aspects. La sensibilité au contexte peut être spécialisée en dépendance de son utilisation.

3.2 Vers la notion d'adaptation

Les auteurs dans (Rey and Coutaz, 2002) proposent une définition pour la notion d'interaction du contexte comme étant l'intersection entre les connaissances contextuelles de l'ordinateur et celles de l'utilisateur dont l'idée est que l'utilisateur se fait des connaissances contextuelles du système. Tirant profit de la définition de la sensibilité au contexte présentée en (Abowd et *al.*, 1999), nous allons s'inspirer et rediriger la notion d'interaction du contexte apportée par ces auteurs vers notre axe de recherche qui porte sur l'adaptation d'applications mobiles au contexte de l'appareil mobile qui les supporte, à savoir spécialiser la définition donnée pour le domaine de développement mobile pour qu'elle soit :

« Le contexte d'interaction est l'interaction entre les informations contextuelles de l'application mobile et celles de l'appareil mobile cible où l'idée que l'application doit toujours avoir des connaissances contextuelles sur son environnement d'exécution afin de mettre à disposition des informations et des services à l'utilisateur » (cf. Figure 1.7).

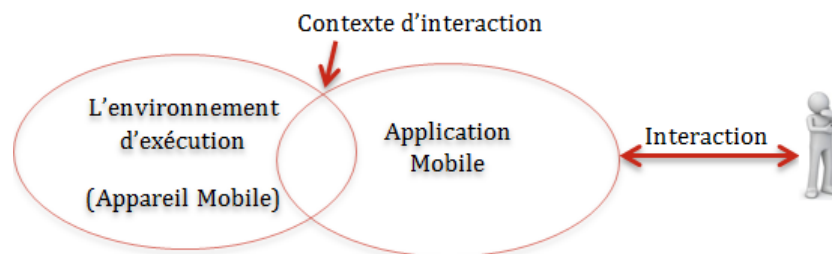


FIGURE 1.7 – Le contexte d'interaction dans un environnement mobile

Par conséquent, le contexte d'interaction dans l'environnement mobile représente la conformité des contraintes d'exécution de l'application avec le contexte d'exécution de l'appareil mobile. En se basant sur les définitions du *contexte* et *context-aware* présentées dans la section précédente, nous devons distinguer entre les applications qui utilisent le contexte comme par exemple un service de météo qui aura besoin d'informations de localisation et de temps pour produire un bulletin et d'autres applications qui adaptent leur comportement

en fonction du contexte. Une autre définition du *context-aware* plus orientée vers l'adaptation au contexte est donné par Brown en (Brown, 1998). Il dit qu'une application sensible au contexte doit automatiquement extraire de l'information ou effectuer des actions en fonction du contexte utilisateur détecté par les capteurs.

D'un côté, une application peut juste se concentrer sur la détection, la perception et l'interprétation des informations contextuelles de l'environnement d'exécution. Ce qui démontre que l'utilisation simple du contexte n'implique pas une modification de son comportement. Par exemple, les applications dans les travaux (Pascoe, 1998; Pascoe et al., 1998) font juste des connaissances sur les éléments de l'environnement de l'utilisateur. D'un autre côté, les applications peuvent changer dynamiquement leur comportement en fonction du contexte de l'environnement d'exécution (Brown et al., 1997; Cheverst et al., 2000). En conséquence, nous avons abouti à une définition beaucoup plus orientée vers notre axe de recherche pour les deux notions d'adaptation et d'auto-adaptation. Afin de pouvoir obtenir des applications adaptatives, le développeur acquiert des connaissances sur l'environnement d'exécution pour garantir si les contraintes d'exécution d'une application mobile sont conformes au contexte d'exécution de l'appareil mobile cible, i.e. l'adaptation du contenu en fonction du terminal (cf. Figure 1.8). Par conséquent, l'adaptation est le fait de s'assurer que l'application souhaitée pourra fonctionner correctement dans un tel environnement mobile.

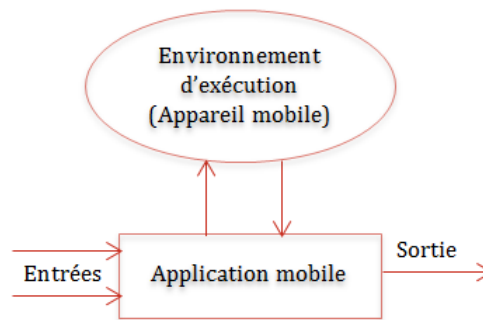


FIGURE 1.8 – Application tenant compte du contexte

Dans le cas du changement des variables du contexte d'exécution, l'application change son comportement de telle sorte où les contraintes d'exécution associées au nouveau comportement doivent être à leur tour conforme à l'environnement d'exécution. Dans ce cas, l'application acquiert des connaissances sur le contexte d'exécution courant, i.e. prévoir les changements du contexte. Ici nous parlons des applications auto-adaptables (cf. Figure 1.9). La boucle de contexte exprime la modification des valeurs des variables du contexte où l'application va subir à un changement de comportement en fonction de ce nouveau contexte.

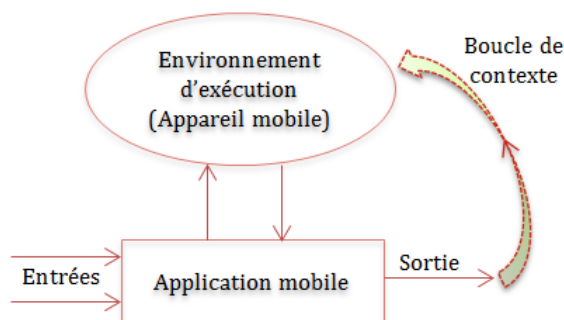


FIGURE 1.9 – Application auto-adaptable

3.3 Modélisation du contexte

Une application sensible au contexte doit fournir des informations et des services pertinents par rapport à la tâche de l'utilisateur en dépendance des informations contextuelles de l'environnement d'exécution de cette application. Une tâche de modélisation du contexte est donc extrêmement importante et indispensable pour atteindre cet objectif. Dans cette section, nous abordons dans un premier temps les modèles dédiés pour la description du contexte. Après, nous présentons quelques travaux qui s'inscrivent dans le domaine de la modélisation du contexte.

■ Modèle du contexte

Afin de pouvoir utiliser les informations contextuelles dans un système sensible au contexte, Dey nous a proposé dans (Dey et al., 2001) de suivre trois étapes nécessaires. En premier lieu, capturer ces informations. La deuxième étape sert à effectuer une interprétation du contexte pour passer à une représentation de haut niveau plus exploitable pour l'application. Finalement, fournir ces informations interprétées à l'application. Pour avoir une représentation fiable et riche des données capturées et aussi garder une trace de l'historique de ses valeurs, plusieurs chercheurs ont proposé de stocker le contexte avant sa diffusion à l'application, i.e. la modélisation de contexte. Dans cette section, nous nous intéressons aux modèles de la littérature pour la prise en compte du contexte.

Les modèles du contexte ont pour objectif de stocker et échanger les valeurs des paramètres contextuels. D'après les travaux de (Hoareau and Satoh, 2009 ; Chen and Kotz, 2000), nous avons trouvé que les structures de données utilisées pour décrire et diffuser le contexte s'organisent selon six catégories : le modèle clés-valeurs, le modèle à balise, le modèle orienté objet, le modèle basé sur la logique, l'ontologie et la logique de situation. Cependant, nous portons une attention particulière aux modèles les plus adaptés pour les environnements mobiles :

- **Le modèle Clés-Valeurs** : ce modèle consiste à stocker le contexte en utilisant un ensemble de couples (attribut, valeur). Par exemple : (location, "hospital"). Dans cet exemple, l'attribut de l'environnement est location. Ce dernier, représente l'information de l'emplacement qui porte la valeur *hospital*.
- **Le modèle à balise** : l'équivalent de ce terme en anglais est *Markup model*. Il contient des données organisées dans les structures hiérarchiques grâce aux balises. Ce modèle sert à représenter le contexte en utilisant le RDF⁶ qui est développé par le W3C.
- **L'ontologie** : les ontologies sont considérées parmi les modèles les plus utiles pour représenter et traiter le contexte. Une ontologie permet de créer des vocabulaires partagés entre les systèmes, i.e. supporter l'interopérabilité, comme elle permet aussi d'ajouter l'aspect sémantique aux données contextuelles.

Chaque modèle est recommandé pour être utilisé pour modéliser le contexte d'un type particulier des applications. Comme par exemple les applications destinées aux ordinateurs de bureau où des serveurs peuvent utiliser des modèles basés sur la logique afin de déduire de nouvelles informations alors que les applications dédiées aux téléphones mobiles nécessitent des modèles plus légers leur permettant de réagir avec le changement du contexte.

■ Approches de modélisation

Dans cette section, nous nous focalisons sur les travaux qui portent sur la phase du stockage du contexte, i.e. description du contexte. La modélisation du contexte est définie par Abi-Char

6. RDF (Resource Description Framework) : est un modèle standard pour l'échange des données sur le web.

et *al.* en 2010 comme une tâche de spécification de toutes les entités et les relations entre ces entités qui sont nécessaires pour décrire le contexte (Abi-Char et *al.*, 2010). Plusieurs représentations contextuelles portant sur les modèles du contexte cités précédemment ont été proposées. Parmi eux nous présentons :

Le contexte *Toolkit* de Dey est l'une des premières architectures qui prend en considération la sensibilité au contexte (Dey et *al.*, 2001). Suivant le principe de cette architecture, une application s'abonne au serveur pour être au courant d'un changement de contexte où les données contextuelles sont capturées par des senseurs et interprétées via des *widgets* avant de les transférer au serveur. Le contexte *Toolkit* de Dey est basé sur le modèle clef-attribut.

La représentation contextuelle la plus connue qui repose sur le modèle à balise c'est CC/PP⁷. Elle est proposée par le W3C (Kiss, 2007) et basée sur le langage général de description des méta-données RDF et encodée en XML. La spécification CC/PP permet de construire un profil, tenant compte des capacités du terminal mais aussi de l'utilisateur. Les auteurs dans (Buchholz et *al.*, 2004) et (Indulska et *al.*, 2003) ont utilisé ce modèle pour la prise en compte du contexte. Mukhtar et *al.* dans leurs travaux effectués respectivement en 2008 et 2009 (Mukhtar et *al.*, 2008 ; Mukhtar et *al.*, 2009) étendent le modèle CC/PP afin de l'adopter pour effectuer la tâche de sélection des composants constituant d'une application multimédia dynamiquement tout en prenant compte leurs ressources nécessaires et/ou les préférences de l'utilisateur.

En 2012, Derdour a proposé un diagramme de classe pour la description du contexte des applications multimédias (Derdour, 2012). Ce contexte est représenté par un profil utilisateur, réseau et plateforme où cette dernière est composée d'un profil matériel, logiciel et environnement. Cette description est basée sur le profil CC/PP pour décrire les caractéristiques physiques de l'utilisateur et les objectifs de transfert des flux multimédias.

De plus, il existe de nombreux travaux décrits dans la littérature qui attaquent l'aspect de modélisation du contexte où chacun fournit une description contextuelle dédiée pour traiter une problématique spécifique dans un système particulier. Dû au fait qu'une personne n'aura pas besoins à tout moment de tous les services auxquels elle pourra accéder, les auteurs dans (Rasch et *al.*, 2011) ont proposé un modèle formel de contexte pour choisir les services les plus pertinents. Ce modèle est basé sur les préférences de l'utilisateur et les descriptions de services pour faire une recherche pro-active de services appropriés, i.e. découverte personnalisée de services.

Toutes les approches de modélisations indiquées en dessus touchent le domaine de l'informatique mobile. Par conséquent, elles sont dédiées aux systèmes sensibles au contexte. Plusieurs d'autres descriptions contextuelles ont été proposées pour la spécification du contexte dans les systèmes pervasives et ubiquitaires. Dans le travail (Strang and Linnhoff-Popien, 2004), les auteurs font une évaluation des modèles du contexte. Ils concluent que les ontologies sont les plus adaptées pour modéliser le contexte dans des environnements ubiquitaires.

Le modèle d'ontologie CoOL (Context Ontology Language) proposé dans (Strang et *al.*, 2003) représente le contexte comme un ensemble d'entités ayant des aspects décrivant leurs caractéristiques. l'auteur dans (Ay, 2007) a proposé une ontologie de contexte commune comme une solution pour l'hétérogénéité des environnements. De plus, nous trouvons le modèle d'ontologie CONON (CONtente ONtologies) (Wang et *al.*, 2004) qui est destiné pour modéliser le contexte dans les systèmes pervasives et l'ontologie de contexte CoDAMos (Context-Driven Adaptation of Mobile Services) qui est proposée en 2005 dans (Preuveneers et *al.*, 2004) et dédiée aux systèmes ambiants intelligents.

En se basant sur les principes de ces différents travaux de modélisation du contexte, nous

7. CC/PP est l'acronyme de « Composite Capabilities/ Preference Profile ».

pensons que le modèle d'ontologies est le plus adéquat même pour définir le contexte de développement des applications mobiles. Dû au fait que ce type d'applications ainsi que leurs environnements d'exécution présentent plusieurs points d'hétérogénéité, les ontologies permettent de fournir un vocabulaire commun partagé entre eux. Donc, la modélisation du contexte en utilisant ce type de modèles du contexte est considérée comme une solution pour l'hétérogénéité présentée par les appareils mobiles et les applications mobiles.

CONCLUSION

Dans ce chapitre nous avons passé en revue quelques informations théoriques sur les notions clés liés à notre travail de recherche incluant des définitions présentées dans la littérature pour chacune des applications mobiles, appareils mobiles et contexte.

Nous avons présenté quelques définitions dédiées aux applications mobiles. Ainsi, nous avons introduit ses plateformes de téléchargements existantes en arrivant par la suite à une étude statistique décrivant les chiffres de téléchargements des applications en générale et en dépendance de ses plateformes d'exécution. Nous avons constaté que contrairement aux dispositifs classiques, i.e. les ordinateurs, les différents appareils mobiles sont caractérisés par des ressources limitées et présentent plusieurs points de différences que ce soit du point de vue matériel ou logiciel.

Cependant, nous avons montré aussi que ces caractéristiques et ressources limitées forment les informations contextuelles de l'environnement d'exécution des applications mobiles et que le contexte de l'environnement d'exécution change souvent. Face à ce constat, nous avons essayé de comprendre ce qui est le contexte pour les applications mobiles et son intérêt pour garantir un meilleur fonctionnement et finalement en présentant quelques approches de modélisation du contexte. Dans le chapitre qui suit, nous aborderons toutes les notions et les différentes techniques ainsi que les travaux existants qui sont liés à nos contributions et qui reflètent un socle théorique pour le processus de composition proposé.

La Composition pour Le Développement d'Applications Mobiles

SOMMAIRE: CHAPITRE 2

Introduction	39
1 L'Ingénierie Dirigée par les Modèles (IDM) pour le développement mobile	40
1.1 Les principes généraux de l'IDM	40
1.2 L'intérêt d'une approche IDM	44
1.3 Pourquoi l'IDM pour le développement mobile ?	45
2 Le développement mobile par la composition de l'existant	47
2.1 Représentation du noyau fonctionnel	47
2.2 Processus de composition	49
2.3 Composition de services/composants	54
2.4 Composition basée composants VS composition orientée services	59
3 Les approches de composition : Synthèse	63
Conclusion	66

INTRODUCTION

L'explosion du nombre d'applications où des services disponibles sur les différentes plateformes des dispositifs mobiles est indéniable. Mis à disposition de l'utilisateur, celui-ci peut alors avoir besoin de plusieurs applications à la fois pour répondre à ses propres exigences. Un besoin naissant est alors de pouvoir combiner différentes applications - entités logicielles - afin de pouvoir tirer le meilleur parti de leurs fonctionnalités pour obtenir une application composite comblant les besoins de l'utilisateur. Dû au fait que les entités logicielles existantes ainsi que ses plateformes d'exécution présentent différents points d'hétérogénéité, l'approche par modélisation permet de faire face à cette problématique et peut conduire à des applications mobiles cohérentes et adaptables à son environnement d'exécution. De nos jours, l'ingénierie dirigée par les modèles prédomine le monde de développement parce qu'il représente la clé d'une porte vers un développement simple et facile pour des plateformes spécifiques en utilisant les modèles. Cependant, plusieurs paradigmes donnant accès à la notion de composition ont été proposés pour la conception des applications dont la modularité est devenue très importante pour principalement favoriser la réutilisation de l'existant. Dans ce chapitre nous mettons l'accent sur le mécanisme de composition d'applications mobiles en relation avec l'approche de modélisation sur lequel s'appuie notre travail de recherche.

1 L'Ingénierie Dirigée par les Modèles (IDM) pour le développement mobile

1.1 Les principes généraux de l'IDM

L'Ingénierie Dirigée par les Modèles (IDM) qui reflète Model Driven Engineering (MDE) en anglais est une des approches modernes du développement des logiciels. Suite à l'approche objet des années 80 et de son principe du « tout est objet », l'ingénierie du logiciel s'oriente aujourd'hui vers l'IDM et le principe du « tout est modèle » se pose alors la question des concepts essentiels. L'approche orientée objets est basée sur deux relations fondamentales : la relation « Instance de » qui permet d'introduire la notion de classe et la relation « Hérite de » qui permet d'introduire la notion de *superclasse*. Il s'agit d'identifier quels sont les relations et les concepts essentiels de l'IDM. Bien qu'aucune réponse à cette question ne puisse être définitive à ce stade de recherches, il apparaît de plus en plus consensuel que deux relations sont fondamentales. La première relation, appelée « Représentation de » est liée à la notion de modèle alors que la relation « Est conforme à » permet de définir la notion de modèle par rapport à celle de méta-modèle. Donc, l'IDM est une ingénierie dédiée à améliorer le développement des systèmes informatiques en se concentrant sur une préoccupation plus abstraite que la programmation classique (Combemale, 2008) où les modèles deviennent les objets centraux à manipuler. Une définition de l'IDM suffisamment large est donnée par Mellor et *al.* dans (Mellor et *al.*, 2003) est : « L'IDM est simplement la notion que nous pouvons construire un modèle d'un système puis le transformer en la chose réelle. ».

Puisque l'IDM met les modèles au premier plan au sein du processus de développement logiciel, les différentes étapes du développement se traduisent par un enchaînement de transformations de modèles. Cette démarche est une forme d'ingénierie générative, i.e. l'automatisation du développement, par laquelle tout ou partie d'une application est engendrée à partir des modèles.

Tout le monde se met d'accord à penser que l'ingénierie logicielle dominée par les modèles est l'avenir des applications et comme dit Bill Gates que « *Modéliser est le futur et je pense que les sociétés qui travaillent dans ce domaine ont raison.* » (Blanc and Salvatori, 2011). Par conséquent, le modèle est devenu le paradigme majeur par lequel l'industrie du logiciel pourra lever le verrou de l'automatisation du développement.

L'OMG a défini MDA (Model Driven Architecture) en 2000 dans cet objectif. L'MDA est une approche principale de développement logiciel qui contribue fortement à l'émergence de l'IDM, i.e. variante particulière de l'ingénierie dirigée par les modèles. Le principe clé et initial de cette approche repose sur l'adoption du standard UML pour définir des modèles séparés décrivant les différentes phases du cycle de développement d'une application. Avec précision, le MDA préconise l'élaboration de modèles : d'exigence (CIM : Computation Independent Model), d'analyse et de conception (PIM : Platform Independent Model), de code (PSM : Platform Specific Model) (Combemale, 2008).

Dans la suite de cette sous-section, nous présentons et définissons les concepts principaux de l'IDM : la modélisation comprenant la notion du modèle et la notion du méta-modèle. Ainsi la transformation de modèles qui définit le passage entre ces différents modèles.

■ La modélisation : Modèle, Méta-modèle

« *Les mauvaises langues prétendent que modéliser est la meilleure façon de perdre du temps puisque, in fine, il faut de toute façon écrire du code.* ». Dans le cadre du développement logiciel, Meyer (Meyer, 1985) rapporte que des spécifications exprimées en langage naturel sont souvent source d'ambiguïtés et qu'il est nécessaire d'exprimer formellement ces spécifications.

Il est indispensable de communiquer dans un langage formel et commun pour partager l'information. De même, au fameux dicton énonçant qu'un bon schéma vaut mieux qu'un long discours. L'utilisation des modèles dans ce cadre est alors un moyen de formalisation.

En informatique « Un modèle a pour objectif de structurer les données, les traitements et les flux d'informations entre entités. » (Minsky, 1965). Nous pouvons définir aussi un modèle comme une abstraction de la réalité, une image simplifiée d'un système donné ou bien une simple représentation d'un système. Les modèles très abstraits sont adoptés pour décrire l'architecture générale d'une application. Un avantage incontestable des modèles est qu'ils peuvent être exprimés sous format graphique ce qui facilite particulièrement la communication entre les acteurs des projets informatiques, i.e. la communication entre les concepteurs/développeurs de l'application. Par conséquent, un modèle définit l'ensemble des fonctionnalités nécessaires pour construire l'application mobile désirée. Ce modèle doit pouvoir être utilisé pour répondre aux besoins des utilisateurs. La représentation graphique de ces modèles offre un gain significatif de productivité.

Dans l'IDM, la notion de modèle fait explicitement référence à la notion de formalisme ou de langage de modélisation bien défini. Un langage de modélisation est défini par une syntaxe abstraite, une syntaxe concrète et une autre sémantique. La syntaxe abstraite se réfère aux concepts de base du langage tandis que la syntaxe concrète définit le type de notation qui sera utilisé pour chaque concept abstrait qui peut être graphique, textuelle ou mixte. Contrairement à la sémantique qui définit comment les concepts du langage doivent être interprétés par les concepteurs mais surtout par les machines.

La modélisation indique la tâche de conception d'un modèle conformément à un langage. Dû au fait que les applications deviennent de plus en plus complexe et leurs maîtrise nécessite de l'abstraction, le recours à cette tâche pour le développement logiciel est devenu indispensable (Roques, 2008). Pour qu'un modèle soit productif, il doit pouvoir être manipulé par une machine. Donc, le langage dans lequel ce modèle est exprimé doit être clairement défini. De manière naturelle, la définition d'un langage de modélisation prends la forme d'un modèle appelé méta-modèle. L'OMG a défini le méta-modèle comme « Un modèle qui définit le langage d'expression d'un modèle. » (Combemale, 2008). Minsky le définit par « Pour un observateur B : un objet M^* est un modèle d'un objet M dans la mesure que B peut utiliser M^* pour répondre aux questions qu'il se pose sur l'objet M. » (Minsky, 1965).

Par conséquent, le méta-modèle modélise les entités d'un langage. Autrement dit, il permet d'identifier les concepts utilisables dans les modèles qui lui sont conformes, i.e. identifier le vocabulaire ou la grammaire. Ainsi, il représente la spécification de la syntaxe du langage en indiquant comment ces concepts sont organisés et quelles sont les informations contenues, i.e. spécifier les relations ainsi que leurs contraintes.

La méta-modélisation est l'activité qui consiste à représenter les langages de modélisation via les modèles. Elle permet de définir des formalismes de modélisation, i.e. un langage qui permet d'exprimer des modèles, grâce à l'identification des concepts à utilisés pour modéliser l'application. Une fois le formalisme défini, tous les modèles conformes à ce formalisme peuvent être traités correctement. Il est improbable d'effectuer cette tâche pour définir un méta-modèle décrivant tous les systèmes informatiques. Un méta-modèle doit être défini pour un objectif particulier afin de créer et d'avoir un langage répondant aux problèmes spécifiques à un domaine particulier (Marvie, 2002).

Le méta-modèle à son tour est exprimé dans un langage de méta-modélisation spécifié par le Méta-méta-modèle. Le langage utilisé au niveau du Méta-méta-modèle doit être suffisamment puissant pour spécifier sa propre syntaxe abstraite et ce niveau d'abstraction demeure largement suffisant (*méta-circulaire* : il peut se définir lui-même). Chaque élément du modèle est une instance d'un élément du méta-modèle. Les modèles peuvent se situer à des niveaux d'abstraction différents en exprimant des relations de raffinement entre eux. Ces

relations expriment des véritables liens de traçabilité qui servent à garantir la cohérence d'un ensemble de modèles représentant la même application.

Un modèle est dit conforme à un méta-modèle et constitue une représentation d'une application souhaitée. La relation entre un Méta-méta-modèle et un méta-modèle est analogue à la relation entre un méta-modèle et un modèle, i.e. les relations existantes entre les deux niveaux M1-M2 et les deux M2-M3 sont équivalentes. Ces relations sont illustrées dans la Figure 2.1.

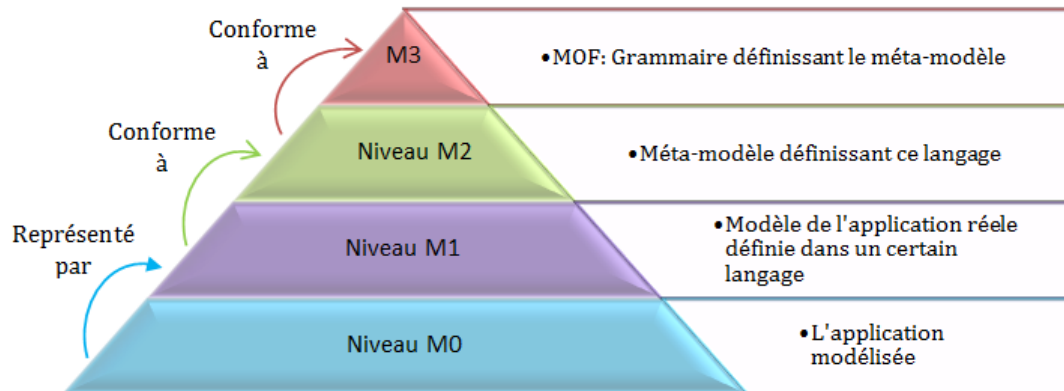


FIGURE 2.1 – Pyramide de modélisation de l'OMG (Minsky, 1965)

Le niveau M0, contient les entités à modéliser qui reflètent dans notre travail de recherche les applications mobiles. Le niveau M1 contient les différents modèles de l'application tandis que le niveau M2 représente les différents méta-modèles qui ont été utilisés. Plus abstraitement, le niveau M3 : Méta-méta-modèle qui a permis de définir uniformément les méta-modèles. M2 définit la structure de M1 tout comme M3 définit la structure de M2. Rappelons que le MOF modèle définit sa propre structure. La modélisation vise à simuler le fonctionnement d'une telle application. Les bienfaits de la modélisation sont résumés par les points suivant :

- **Abstrait** : elle fait ressortir les points importants tout en enlevant les détails non nécessaires.
- **Compréhensible** : elle permet d'exprimer une chose complexe dans une forme plus facilement compréhensible par l'observateur.
- **Précis** : elle représente fidèlement l'application modélisée.

■ Transformation de modèles

La transformation de modèles est un processus fondamental dans le cadre de l'IDM. Cette opération permet de générer des modèles de plus bas niveau (ou code) à partir des modèles de plus haut niveau dont l'objectif est de créer l'application réelle. Le processus de développement basé sur cette approche peut alors être vu comme une séquence de transformations de modèles partiellement ordonnée.

Chaque transformation prend un ou des modèles en entrée et produit un ou des modèles en sortie jusqu'à l'obtention d'artéfacts exécutables (Diaw et al., 2010). Cette transformation se fait par l'intermédiaire des règles de transformation où ces dernières décrivent la façon dont les concepts du langage source peuvent être transformés en d'autres concepts du langage cible. Donc, elle décrit la correspondance entre les entités du modèle source et celles-ci du modèle cible. Une fois spécifiées et exprimées les règles requises, la transformation pourra

être effectuée par un moteur de transformation sur des modèles concrets. Ces derniers représentent les instances du méta-modèle source tandis que les modèles obtenus seront conformes au méta-modèle cible (cf. Figure 2.2).

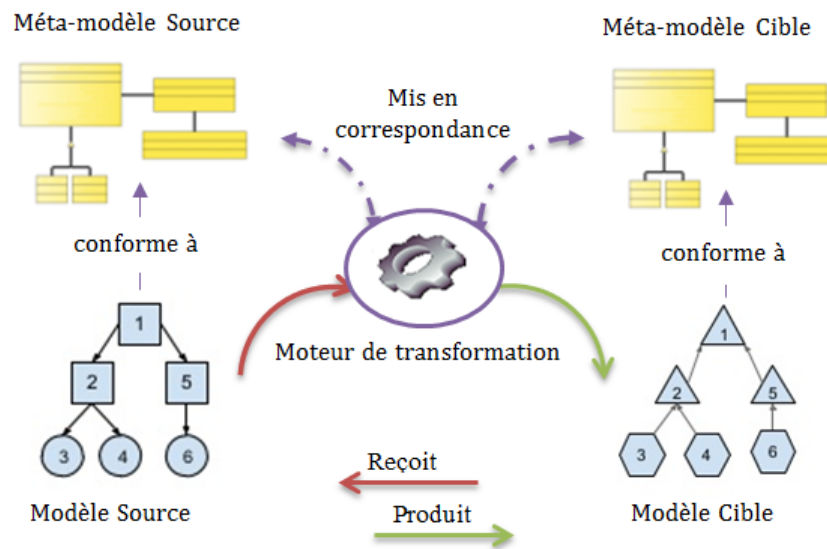


FIGURE 2.2 – Principe de transformation de modèles (De Castro et al., 2011)

Par conséquent, le principe de transformation repose sur deux étapes essentielles où :

- (1) La première consiste à identifier la correspondance entre les modèles sources et cibles au niveau de leurs méta-modèles.
- (2) La deuxième étape vise à appliquer la transformation des modèles sources afin de générer automatiquement le modèle cible via le moteur de transformation en se basant sur les règles de transformation définies.

• *Catégories des transformations*

Dans la littérature (Czarnecki and Helsen, 2003 ; Gerber et al., 2002), il existe deux types principaux de transformations. Le premier type c'est la transformation *Modèle vers Modèle* qui représente le passage d'un formalisme à un autre. Ce type de transformation permet alors de raffiner un modèle abstrait vers un modèle spécifique. Comme propose l'approche MDA, il est possible de passer d'un modèle indépendant des plateformes (PIM) vers un autre dépend d'une plateforme (PSM). On outre, plusieurs travaux sont également élaborés à ce stade. Par exemple, les auteurs dans (Cariou et al., 2009) explique la possibilité de transformer un modèle abstrait vers un modèle d'implémentation plus détaillé.

Le deuxième type de transformations représente la transformation d'un modèle vers un texte *Modèle vers Code*. Ce type de transformation vise généralement à la génération du code où à partir d'un même modèle nous pouvons générer un code dans différents langages en termes des transformations. En vertu de cette transformation, il suffit de définir le méta-modèle du langage de programmation cible pour avoir le code requis qui correspond au modèle source donné tel que la génération du code Java par exemple. En plus de ces deux types principaux de transformations, nous distinguons selon (TOPCASED-WP5, 2008) d'autres types de transformations :

- *Transformation (1 vers 1)* : elle associe au modèle source au plus un seul modèle cible, ex. la transformation d'une classe UML munie de ses attributs en une table d'une base de données relationnelle.

- *Transformation (M vers N)* : elle prend en entrée un ensemble des modèles source et produit un ensemble des modèles cible, ex. les transformations de fusion de modèles (N vers 1) et de décomposition de modèles (1 vers N).
- *Transformation sur place (mise à jour)* : elle consiste à modifier un modèle par l'ajout, la modification ou la suppression d'une partie de ses éléments, ex. la restructuration de modèles (*model refactoring*) qui reflète l'organisation des éléments du modèle source afin d'en améliorer sa structure ou sa lisibilité.

- ***Taxonomie des transformations***

Il est important de souligner que les modèles source et cible peuvent appartenir à des espaces technologiques différents. A titre d'exemple, la transformation d'un modèle de classes persistantes en un schéma de base de données relationnelles qui fait intervenir les espaces technologiques correspondants, respectivement de la modélisation (UML) et celui des bases de données. Partant de la nature des méta-modèles source et cible, nous distinguons selon la classification adoptée dans les transformations dites *endogènes* et *exogènes* combinées à des transformations dites *verticales* et *horizontales*. Une transformation est dite *endogène* si les modèles cible et source sont issus du même méta-modèle et *exogène* dans le cas contraire. La transformation *endogène* représente une tâche de raffinement des modèles. Contrairement à la transformation *exogène* qui consiste à effectuer une tâche de projection.

Une transformation simple ou multiple peut être *exogène* ou *endogène* selon la nature des méta-modèles source et cible impliqués dans la transformation. Par contre une transformation sur place implique un même méta-modèle, donc elle est *endogène*. Une démarche de transformation peut aussi conduire à un changement de niveau d'abstraction. Une transformation est dite *verticale* si elle met en jeu différents niveaux d'abstraction dans la transformation. Le passage de PIM vers PSM ou rétro-conception est une transformation *exogène/verticale* alors que le raffinement est une transformation *endogène/verticale*. Une transformation est dite *horizontale* lorsque les modèles source et cible qui font l'objet de la transformation sont au même niveau d'abstraction. La restructuration, la normalisation et l'intégration des patrons sont des exemples de transformation *endogène/horizontale* tandis que la migration des plateformes et la fusion de modèles sont des exemples de transformation *exogène/horizontale* (cf. Tableau 2.1).

Tableau 2.1 – Taxonomie des transformations

<i>Transformation</i>	<i>Horizontale</i>	<i>Verticale</i>
Endogène	Restructuration/ l'intégration des patrons	Raffinement
Exogène	Migration de logiciels	Génération du code/ rétro-conception

1.2 L'intérêt d'une approche IDM

Afin d'obtenir un produit logiciel qui répond aux attentes des utilisateurs, il est nécessaire de pouvoir transformer des modèles passant d'un niveau d'abstraction à un autre ou d'un espace technologique à un autre. Pour ce faire, des organismes tels que l'OMG et les chercheurs en génie logiciel cherchent à enrichir les méta-modèles qui sont utilisés dans la conception d'applications ou à en définir de nouveaux. L'objectif est de faciliter la création de nouveaux

espaces de modélisation plus adaptés aux exigences des utilisateurs et à automatiser les différentes étapes de modélisation nécessaires à l'élaboration d'un produit fini.

Cependant, l'un parmi les intérêts les plus importants de l'IDM est d'aider les concepteurs/développeurs à gérer les entités logicielles complexes en fournissant des niveaux d'abstraction élevés et d'automatiser leurs implémentations. En outre, cette approche vise à élaborer des modèles pérennes indépendants des détails techniques des plateformes d'exécution afin de générer automatiquement de la totalité du code des applications et d'obtenir un gain significatif de productivité. Cette ingénierie dirigée par les modèles présente plusieurs avantages pour le développement logiciel :

- L'amélioration de la portabilité via la séparation des connaissances métiers de la technologie d'implémentation (ex. la séparation des exigences de l'application de la plateforme d'exécution).
- L'augmentation de la productivité via la transformation automatique.
- L'amélioration de la qualité tout en minimisant le coût et le temps de développement via la réutilisation des patrons bien éprouvés avec les bonnes pratiques de transformation et la génération automatique du code. Par conséquent, elle réduit significativement le temps de développement jusqu'au tiers du temps de développement manuel.
- L'amélioration de la maintenabilité due à une bonne séparation des préoccupations en spécifiant un système indépendamment de la plateforme qui le supporte ; ce qui conduit à élever le niveau d'abstraction et éviter partiellement la programmation à bas niveau.
- Transformer la spécification du système en une autre spécification pour une plateforme particulière afin d'obtenir une implémentation exécutable ce qui conduit à une meilleure cohérence et traçabilité entre les modèles et le code.
- L'utilisation des modèles accélère le développement d'applications dans le contexte des plateformes hétérogènes.

1.3 Pourquoi l'IDM pour le développement mobile ?

Dans un environnement mobile, le marché des applications pour terminaux mobiles connaît une croissance fulgurante ces dernières années s'appuyant sur les plateformes *iOS*, *Android*, *Windows Phone* ou *Tizen*, entre autres. Chaque OS mobile utilise son propre langage de programmation et arbore ses propres APIs. Ainsi, les entités logicielles constituant de l'application mobile peuvent prendre différentes formes d'implémentations. Toutefois, cette hétérogénéité oblige les éditeurs de logiciels à re-développer entièrement leurs produits pour chaque technologie cible ou confronter une difficulté à recomposer des entités logicielles déjà existantes à cause de leurs hétérogénéités ce qui engendre des sur-coûts difficilement supportables en terme de temps et d'argent.

Face à ce constat, les préceptes du MDA (Van Hien, 2007) permettent de capturer les concepts communs aux différentes plateformes en vue de produire des PIM pour représenter une application mobile de manière technologiquement neutre. Puis, nous nous appuyerons sur des transformations *Modèle vers Modèle* pour produire des PSM pour chaque technologie cible. Enfin, ces PSMs produiront le code final exécutable à l'aide de transformation *Modèle vers Code* ce qui donne la possibilité d'attribuer la même application mobile à plusieurs plateformes d'exécution. Par conséquent, l'adoption sur l'IDM dans un environnement mobile notamment l'approche MDA contribue à augmenter le niveau d'abstraction de développement d'applications mobiles. Cette démarche permet une réduction du temps de développement

jusqu'à trois fois grâce à la génération du code pour différentes plateformes mobiles. En outre, elle améliore la qualité du code et augmente l'efficacité des logiciels générés.

Dans ce travail de recherche, nous avons adopté une approche dirigée par les modèles pour atteindre nos buts. Cette démarche est une ingénierie intéressante et très convenable pour faciliter la tâche de développement d'applications mobile par la réutilisation de l'existant tout en faisant face aux problèmes de l'hétérogénéité. Nous nous sommes concentrés beaucoup plus sur la description architecturale de l'application mobile composite et comment la composer à partir des exigences des utilisateurs tout en utilisant des entités logicielles déjà existantes. En outre, nous nous sommes appuyés par la suite sur le mécanisme de transformation pour avoir une architecture détaillée pour l'application mobile souhaitée.

Les raisons qui nous ont poussés à choisir une telle approche et l'adopter pour effectuer la tâche de composition d'applications mobiles sont :

- **La modélisation graphique** : Dans l'IDM, la modélisation graphique est favorisée. La notion de diagramme reflète souvent la syntaxe concrète d'un langage de modélisation. Cette notion est définie par Larkin (Larkin and Simon, 1987) comme suivant : « *is a representation in which information is indexed by 2D locations* ». Une description visuelle permet au développeur de monter en abstraction et d'avoir une vision complète de son application ce qui rend possible et facile la communication entre les développeurs/concepteurs via ces représentations partagées et compréhensibles. Ces représentations sont basées sur des notations visuelles définissant l'ensemble des symboles graphiques que le créateur d'un langage de modélisation affecte aux concepts nécessaires pour définir son application. En utilisant ces symboles graphiques le développeur/concepteur peut créer le diagramme qui correspond à une vue graphique sur le modèle de son application où ce dernier sera conforme au langage de modélisation proposé. La modélisation graphique d'une application mobile permet de détecter la présence d'erreurs, les identifier et donc les corriger tôt dans le processus de développement ce qui diminue sensiblement le coût de développement.
- **La réutilisation du modèle** : La notion de modèle s'avère très utile pour le développement logiciel. Dans les activités humaines, la réutilisation est la stratégie de résolution des problèmes (Lenat et al., 1990). La réutilisation des modèles représente alors une solution aux problèmes récurrents dans le monde de développement logiciel. Elle représente une bonne stratégie pour aider les concepteurs et faire évoluer les modèles (Ghezzi et al., 2002). Avec le mécanisme de la réutilisation de modèles, le concepteur gagnera du temps en modélisant une seule fois les éléments du modèle qui peuvent se répéter afin de le considérer par la suite comme un *canevas* à suivre où un *template* pour modéliser l'application souhaitée. Le modèle à réutiliser permet aux développeurs de l'appliquer sur plusieurs domaines d'application et l'implémenter pour différents environnements d'exécution ce qui facilite la tâche de développement.
- **La génération du code** : La génération du code occupe la dernière étape dans un processus de développement basé IDM, i.e. transformation d'un modèle vers un texte via une succession de transformations de modèles. Cette génération automatique du code vise à orienter les modèles vers une utilisation plus productive et donne la possibilité de basculer entre les plateformes mobiles. Par exemple, les codes des systèmes d'exploitations *Android*, *iOS* ou *Windows Phone* peuvent être générés à partir d'un modèle d'une application mobile indépendante de la plateforme d'exécution afin d'avoir l'application concrète désirée vers une plateforme spécifique. Ce code généré peut ne pas constituer l'ensemble des codes de l'application envisagée et nécessite l'intervention du développeur pour l'amélioration ou bien l'ajout des morceaux du code. Néanmoins, cette génération de code réduit considérablement le coût de développement pour chaque plateformes.

Ces critères ont une importance majeure et une grande influence pour attaquer l'hétérogénéité des entités logicielles constituant de l'application mobile désirée lors de la composition et aussi faire face à l'hétérogénéité des plateformes mobiles afin d'avoir des applications adaptatives.

2 Le développement mobile par la composition de l'existant

Avec l'arrivée des Smartphones, téléphones portables et avec l'évolution du web, l'explosion du nombre d'applications disponibles sur ces différentes plateformes mobiles est devenue indéniable. Mis à disposition de l'utilisateur, celui-ci peut avoir besoin d'utiliser plusieurs applications à la fois selon son contexte d'utilisation afin d'avoir leurs propres applications qui répondent à ces exigences. Face à ce constat et pourtant avec l'apparition quotidienne d'applications, les développeurs ont de plus en plus besoin d'effectuer de telles compositions pour répondre à la demande croissante. Un des besoins naissants est alors de pouvoir combiner différentes applications ou des entités logicielles déjà existantes afin de pouvoir tirer le meilleur parti de leurs fonctionnalités et d'avoir finalement une nouvelle application comblant les besoins des utilisateurs. Par conséquent, les applications mobiles sont presque devenues infinies ce qui explique le besoin de composer de nouvelles applications par la réutilisation de l'existant pour combler s'il y a des exigences sans avoir à reprogrammer.

En faisant allusion au concept de décomposition (Parnas, 1972) suivant le fameux proverbe « *Diviser pour mieux régner* », les unités logiques des architectures des applications sont divisées en entités logicielles dans l'intention de promouvoir leur réutilisation et aussi clôturer l'accroissement de la complexité. Ces briques sont appelées des entités logicielles élémentaires. Par une entité logicielle, nous nous référons à toute entité visant à fournir un tel service ou implémenter une telle fonctionnalité. Face à ce constat, une application peut être construite par composition d'entités logicielles existantes où ces entités peuvent être soit des composants, des services ou des applications elles-mêmes. Les auteurs dans (Nickul et al., 2007) ont déclaré que cette réutilisation nécessite que chaque service utilisé soit au préalable décrit par son fournisseur. L'avantage de la réutilisation est un gain de temps et d'investissement lors de la conception d'une nouvelle application. Par conséquent, un des leviers permettant d'augmenter la rentabilité ainsi que la rapidité de développement est de favoriser la réutilisation de l'existant. Cette réutilisation peut se faire grâce aux mécanismes de composition d'applications.

Afin de combler les besoins des utilisateurs par la réutilisation de l'existant, plusieurs mécanismes de composition sont développés (cf. Section 3). Dans ce travail de recherche, nous considérons que la composition peut être classifiée en *Macro Composition* et *Micro Composition*. La *Macro Composition* concerne la composition des applications classiques et reflète la composition dans un sens large ça veut dire sans limitation tandis que la *Micro Composition* concerne la composition des applications mobiles qui doivent être soumises et adaptables à plusieurs contraintes de l'environnement d'exécution. La notion de *Micro Composition* se réfère à l'obligation de composition en respectant les ressources limitées offertes par les appareils mobiles ainsi la variabilité de ses états d'exécution.

2.1 Représentation du noyau fonctionnel

De nos jours, plusieurs principes, standards et pratiques sont utilisés pour la conception de systèmes informatiques. Cependant, les possibles représentations pour le noyau fonctionnel d'une application sont nombreuses. Nous nous focalisons dans ce travail sur les représentations donnant accès à une composition où la modularité est devenue très importante d'un côté

pour faciliter la tâche de développement et d'un autre côté pour favoriser la réutilisation de l'existant. Deux types de représentations prédominent alors : l'approche utilisant des services (SOSE : Service-Oriented Software Engineering) et l'approche basée sur des composants (CBSE : Component-Based Software Engineering).

■ La représentation à base de composants

Les auteurs dans (Brown and Wallnau, 1998; Cai et *al.*, 2000) s'attaquent à la programmation orientée composants dont l'objectif est de construire des systèmes logiciels en utilisant des composants sur étagère (COTS : Commercial Off-The-Shelf) et ainsi accélérer le processus de création de logiciels. D'après ces auteurs, un composant est une partie indépendante et remplaçable d'un système qui remplit une fonction claire dans une architecture bien définie. Ainsi un composant peut être associé à l'exécution via son instanciation et il est identifiable à travers ses interfaces qui spécifient un contrat. Un composant est *composable* c'est-à-dire qu'il peut prendre place dans un composant dit composite. Ce dernier utilise d'autres composants pour fournir une fonctionnalité de plus haut niveau et il peut-il-même être composable.

Une approche à base de composants (Weinreich and Sametinger, 2001) sert à capitaliser du code dans des entités logicielles dites boîtes noires qui sont alors réutilisables et dont seules les interfaces logicielles d'échange sont connues. Donc, les composants peuvent être inter-connectés via seulement ses interfaces qui peuvent être fournies ou requises. Une interface fournie est connectée à une interface requise. Les interfaces fournies décrivent ce que propose le composant en terme de fonctionnalités tandis que les interfaces requises décrivent ce qu'il est nécessaire de fournir à un autre composant pour son bon fonctionnement, i.e. elles décrivent les dépendances entre composants.

Il y a plusieurs langages de modélisation appelés ADLs (Architecture Description Languages) destinés à modéliser une architecture logicielle à base de composants d'un système. Ces derniers offrent des formalismes permettant à un concepteur de modéliser à un haut niveau d'abstraction les spécifications et le développement des logiciels sans oublier que les ADLs ne dépendent pas des langages de programmation et des environnements d'exécution. Par conséquent, les ADLs sont un support pour la description de la structure de l'application. Ces ADLs offrent des facilités de réutilisation des composants et des moyens de description de la composition par définition des dépendances entre composants et des règles de communication à respecter (Soucé and Duchien, 2002).

Comme exemples des langages de description pour ce type d'architectures nous citons : Fractal (Bruneton et *al.*, 2006), Wright (Allen et *al.*, 1998), Darwin (Luckham and Vera, 1995), C2SADEL (Medvidovic et *al.*, 1999), ACME (Garlan and Perry, 1995), xADL2.0 (Dashofy and Van Der Hoek, 2002).

■ La représentation orientée services

Une deuxième approche pour la construction d'applications et la représentation de ses noyau fonctionnel est basé sur l'Architecture Orientée Services (SOA : Service Oriented Architecture (Erl, 2005)). Ce type d'architectures a pour noyau l'utilisation de parties logicielles mettant à disposition une fonctionnalité ou partie de fonctionnalité permettant de remplir une tâche particulière dont l'objectif est de rendre un service comblant un besoin particulier.

En 2005, Srinivasan et Treadwell (Srinivasan and Treadwell, 2005) attirent l'attention sur la signification du terme SOA qui se réfère à la conception d'un système et non pas à son implémentation. En 2007, les auteurs dans (Papazoglou and Van Den Heuvel, 2007) décrivent l'architecture orientée services comme la manière logique de concevoir des systèmes logiciels qui fournissent des services où un service est mis à disposition par un fournisseur (ou producteur) au bénéfice d'un client (ou consommateur).

Les systèmes d'informations reposant sur une architecture basée services mettent en place un annuaire qui permet d'enregistrer les autres services. Le référencement de services le plus utilisé est UDDI (Universal Description Discovery and Integration). Par conséquent, dans la programmation orientée service il existe trois acteurs principaux : le fournisseur, le consommateur et le courtier (*registry* en anglais). Un fournisseur publie ses services auprès du courtier pendant qu'un consommateur découvre les services connus par le courtier.

Les services implémentent une ou plusieurs interfaces correspondant aux différentes actions spécifiques qu'ils peuvent réaliser où ces services sont inter-connectés entre-eux via des standards qui assurent la réduction des dépendances, i.e. couplage faible. Ces standards sont souvent basés sur des échanges de documents en langage XML (Extensible Markup Language). Donc, l'une des caractéristiques qui font l'intérêt de ce type d'architectures est le couplage lâche où le consommateur n'a pas besoin de connaître l'implémentation du service et que ce couplage peut se faire à l'exécution grâce au courtier.

Les auteurs dans (Srinivasan and Treadwell, 2005 ; Papazoglou et *al.*, 2007) discutent de plusieurs caractéristiques pour ce type d'entités logicielles. Les services peuvent être utilisés séparément ou intégrés dans un service composé et peuvent être autonomes ou dépendants d'autres services ou de ressources pour fournir leur résultat. Ces services communiquent avec leurs clients à travers des messages émis et des réponses fournies et n'ont pas besoin de décrire les détails d'implémentation. Ils doivent uniquement décrire leurs interfaces, politiques et protocoles de communication.

Une déclinaison sur internet des services sont les web Services. Plusieurs langages ont été abordés pour décrire ce type d'entité logicielle. WSDL (Web Services Description Language) (Christensen et *al.*, 2001) est un langage basé sur XML dédié pour la description des services web qui respectent la spécification WS*. En plus de l'utilisation du WSDL par les services web comme langage XML de description, ces services sont aussi basés sur SOAP (Simple Object Access Protocol) (Hadley et *al.*, 2003) comme un protocole de communication utilisant lui-même HTTP comme protocole de transport.

Afin d'automatiser l'utilisation des web services et notamment leur découverte, leur invocation et leur contrôle, une description sémantique pour ces services est donc nécessaire. OWL-S (Ontology Web Language for Services) (Martin et *al.*, 2004) vient en complément à la description WSDL dont l'objectif est d'ajouter l'aspect sémantique des services. Ce langage permet de décrire sémantiquement les services web suivant trois parties : (a) Le profil du service (*service profile*) qui décrit ce que fait le service et est plutôt destiné à être lu par des humains. (b) Le modèle du service (*service model*) qui décrit comment fonctionne le service (entrées, sorties, pré-conditions, résultats, etc.). (c) Les bases du service (*service grounding*) qui détaillent comment interagir avec le service, les protocoles de communication, les formats de message, etc.

2.2 Processus de composition

Nous donnons tout d'abord une définition générale de la composition d'applications présentée par Christian Brel dans son travail de recherche (Brel, 2013) : « La composition d'applications est le moyen de combiner plusieurs morceaux d'applications existantes pour en construire une nouvelle. Cette composition réutilise tout ou une partie des applications existantes. ». Dû au fait qu'une application peut être représentée par des services ou des composants, plusieurs définitions plus précises et spécialisées pour des domaines particuliers ont été proposées pour la notion de composition.

Plusieurs définitions ont été proposées pour la composition qui porte sur les services web. Les auteurs dans (Martin et *al.*, 2005) définissent la composition comme suivant «

La composition est le processus de la sélection, la combinaison et l'exécution des services web pour atteindre l'objectif de l'utilisateur. ». Les auteurs dans (Benatallah et *al.*, 2005; Alonso et *al.*, 2004) ont dit que « Si l'objectif du concepteur d'une application n'est pas atteint par l'invocation et l'exécution d'un simple service web élémentaire, alors le concepteur doit combiner les fonctionnalités d'un ensemble de services. Dans ce cas-là cette tâche est appelée *composition de services web* ». Les auteurs considèrent la composition de services web comme étant un moyen efficace pour créer, exécuter et maintenir des services qui dépendent d'autres services où les services web invoqués lors de la composition sont appelés *services web composants*.

D'autres travaux ont mis l'objet de composition des entités logicielles du type service. Les auteurs dans (Alonso et *al.*, 2004) ont défini le processus de composition de services comme la spécification d'un service composite à partir d'autres services. Une autre définition plus significative présentée par (Abi Lahoud, 2010) est la suivante : « La composition de services désigne une interaction entre deux ou plusieurs services en vue d'accomplir des objectifs déterminés. ». D'après ces travaux (Alonso et *al.*, 2004; Papazoglou and Van Den Heuvel, 2007; Di Nitto et *al.*, 2008; Chang et *al.*, 2008), la composition de services nécessite au préalable d'établir le lien d'exploitation avec le fournisseur de services et après combiner les différents services réutilisés qui sont identifiés sous la responsabilité du consommateur de services afin de réaliser l'application. Par conséquent, ce mécanisme de composition vise dans un premier temps à découvrir les services disponibles qui peuvent répondre aux besoins du consommateur ainsi qu'à sélectionner ceux les plus adaptés aux préférences; après à définir et à gérer les collaborations entre les services effectivement utilisés.

En effet, la composition d'applications est un processus complexe qui fait intervenir un ensemble d'étapes intermédiaires. Ces étapes se répartissent suivant les préoccupations classiques liées à la réutilisation d'entités logicielles existantes commençant par l'identification des entités qui seront composées puis la réalisation effective de cette composition. Par conséquent, le processus de composition d'applications peut être divisé en trois étapes : (1) la découverte, (2) la sélection et (3) la composition d'entités logicielles. Vis à nos objectifs dans ce manuscrit, nous proposons un processus de composition global constitué de cinq étapes comme illustré dans la Figure 2.3 (Djeddar et *al.*, 2015).

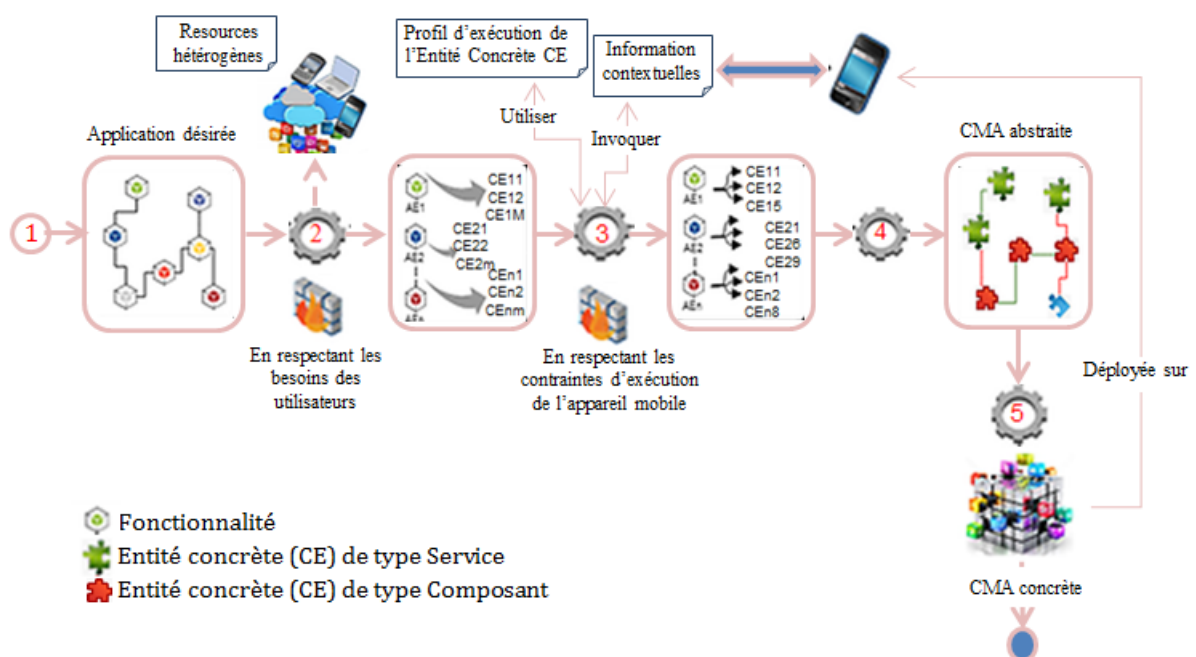


FIGURE 2.3 – Aperçu générale du processus de composition global proposé

Ce processus vise à effectuer la tâche de composition en commençant par l'identification de l'ensemble des fonctionnalités souhaitées tout en ignorant comment elles seront implémentées ainsi que les différentes dépendances entre ces fonctionnalités. Ces dépendances décrivent les ordres d'invocation ainsi que les données échangées entre elles. Après, il associe chaque fonctionnalité identifiée aux entités logicielles concrètes correspondantes qui peuvent être utilisées pour l'implémenter et qui peuvent se trouver dans différents emplacements (ex. internet, emplacement locaux, d'autres appareils, etc.). L'étape suivante permet de sélectionner parmi les entités correspondantes trouvées les plus appropriées qui sont adaptables au contexte courant de l'appareil mobile. La tâche de sélection est basée sur les conditions d'exécution des entités logicielles et la description contextuelle de l'appareil mobile cible. Ensuite, il vise à composer les entités contextuelles sélectionnées tout en incluant les adaptateurs nécessaires dans le cas d'une coordination hétérogène. La dernière étape consiste à générer l'application mobile composite concrète vers une plateforme spécifique sur laquelle elle sera exécutée. Cependant, nous devons mettre l'accent sur l'acronyme CMA que nous proposons pour représenter le terme Composite Mobile Application.

Nous allons maintenant aborder une définition spécifique pour chacune de ces tâches : la découverte, la sélection et la composition ainsi que quelques travaux apportés pour chacune d'entre elles.

■ La tâche de découverte

La découverte est un point clé nécessaire pour le développement d'applications par la réutilisation de l'existant. Elle désigne le processus qui permet d'identifier les entités logicielles qui peuvent répondre aux besoins exprimés par l'architecte. Ce processus de découverte repose, selon le cas, soit sur une recherche dans un *entrepôt* ou un *registre local* ou *public*, i.e. chercher des composants ou des services, soit sur une découverte globale sur le *web*, i.e. chercher des services web. La tâche de découverte sert à comparer les descriptions d'entités logicielles dont il a accès avec les exigences fonctionnelles et non fonctionnelles qui lui sont fournies afin de déterminer les entités candidates. Ces dernières sont les entités existantes d'après leurs descriptions peuvent répondre aux exigences. Afin d'avoir cette liste des candidates, le processus de découverte doit passer par deux étapes essentielles : (1) L'accès aux entités logicielles disponibles ((Verma et al., 2005 ; Kalasapur et al., 2007)). (2) La description de ces entités et les besoins puis la détermination d'une solution comme adéquate ((Li et al., 2009 ; Suraci et al., 2007)).

Donc, la découverte est permise par la description préalable des entités et leur publication au sein d'un *registre* ou d'autres emplacements. Dans le cas où les écarts avec les exigences sont toujours expressifs, les solutions les plus proches possibles sont proposées puis laissées au choix de l'architecte de les utiliser ou non.

Cependant, nous abordons une définition plus orientée vers notre axe de recherche et convenable aux applications mobiles pour la notion de découverte : « Le processus de découverte vise à chercher et choisir les entités logicielles correspondantes pour chaque fonctionnalité désirée en les téléchargeant via internet où certaines d'entre elles sont gratuites tandis que d'autres sont payantes. Ce processus exploite l'ensemble des fonctionnalités identifiées qui représente l'application mobile désirée afin de trouver les différentes entités logicielles qui satisfont les exigences des utilisateurs décrites par ces fonctionnalités. »

Plusieurs travaux ont été effectués sur cet axe de recherche. Les auteurs dans (Toma et al., 2005) ont présenté une approche pour la découverte automatique des services dans les environnements d'exécution distribuée (WSMX : Web Service Modelling eXecution environment). Ce processus de découverte vise à comparer la description de la requête avec la description des services stockés dans un répertoire localisé dans le pair ; ce répertoire est dit le registre local des services. La plateforme WSMX proposée dans ce travail est constituée de deux composants principaux *communication discovery sub-component* et *local discovery*

sub-component. Le premier composant gère la communication avec les autres contrairement au deuxième composant qui est dédié à la découverte local support dirigé par les mots clés. Cette approche supporte la sémantique de similarité entre la requête et le service mais il ne prend pas en considération le processus composition, dans le cas où la requête n'est pas servie par un seul service. Merla dans (Merla, 2010) propose une plateforme pour la découverte sémantique des services web. Cette plateforme effectue la tâche de découverte avec la prise en compte des pré-conditions comme les informations du contexte des services. Cette tâche supporte la priorité inter-contexte entre les services. La plateforme proposée est alors capable de sélectionner le service web qui répond le mieux au contexte indiqué par l'utilisateur sur un appareil mobile à partir de la liste des fonctions des services Web similaires. D'autre part, la déduction de la sémantique de similarité entre les paramètres de la requête et les services fournis est basée sur une ontologie générique d'un domaine particulier spécifié en OWL-S. Les auteurs dans (He et al., 2013) proposent une approche dynamique pour la découverte des services web dans les systèmes pair à pair structurés.

Les différentes approches d'alignement existantes entre les besoins et les entités logicielles qui peuvent répondre à ces besoins peuvent se différencier en terme de deux critères importants : le degré de pertinence de la solution et le type de correspondance. Les algorithmes de découverte peuvent varier selon leurs capacités à gérer la pertinence des candidats déterminés par rapport aux exigences fonctionnelles et non fonctionnelles définies. Afin d'améliorer significativement cette pertinence, la formulation des besoins et des descriptions d'entités logicielles nécessite la prise en compte des sémantiques par l'emploi d'ontologies de domaines d'applications (Rake et al., 2009; Ma et al., 2008). En outre, ces algorithmes de découverte peuvent conduire à l'identification d'une unique entité logicielle existante pour répondre à un besoin spécifique, i.e. l'approche 1-1 (Gu et al., 2004; Verma et al., 2005). Ils peuvent aussi identifier une composition de plusieurs entités pour supporter ce besoin, i.e. l'approche 1-N (Kalasapur et al., 2007; Beauche and Poizat, 2008). Quelques approches de découverte sont conçues pour être capable de construire des solutions qui n'existent pas directement dans le système en composant jusqu'à N entités logicielles disponibles. Ainsi, ces approches supportent la spécification des bonnes collaborations entre ces N entités en étroite liaison avec les exigences du consommateur.

■ La tâche de sélection

Pour une telle composition, le résultat de la tâche de découverte est un ensemble d'entités logicielles correspondantes pour chaque besoin défini ou fonctionnalité désirée. Les entités trouvées mappées à la même fonctionnalité sont équivalentes fonctionnellement mais elles peuvent varier en plusieurs aspects non fonctionnels. La seconde étape de la composition correspond au processus de sélection où ce dernier sert à identifier parmi les entités candidates celles les plus appropriées aux besoins de l'application à construire. En général, ces entités candidates répondent tous aux conditions minimales d'acceptation posées par l'architecte de l'application. La sélection se base donc sur les préférences des utilisateurs ou d'autres critères/contraintes pour identifier celle avec la plus haute qualité et/ou la plus adaptée au contexte d'utilisation. Ces contraintes de sélection diffèrent selon le domaine d'application. Plusieurs travaux ont été effectués à ce stade de recherche. Maintenant, nous présentons parmi eux quelques travaux proposés précisément pour les environnements mobiles.

Dans (Varshavsky et al., 2005), les auteurs ont proposé une approche transversale à la sélection de service. Ils ont focalisé sur la manière dont le client et le serveur sont appairés tenant en considération l'aspect de performance du réseau. Cette approche permet au client de commuter les meilleurs serveurs que les changements de topologie du réseau, i.e. regroupe les clients avec les serveurs à proximité, en se basent sur le mécanisme de routage de réseau *ad-hoc*. Ces auteurs ont montré également que la sélection efficace peut améliorer le débit du réseau jusqu'à 40%. Les auteurs dans (Zhang et al., 2008) ont montré que la sélection

du fournisseur de services avec le plus bas hop-count, i.e. choix des fournisseurs de services les plus proches, ne suffit pas pour parvenir à une meilleure performance du système. A cet effet, ils ont proposé une sélection intégrée qui prend une évaluation plus complète de chaque fournisseur de services y compris la capacité de l'opérateur, le risque de défaillance de service dans le réseau, le temps de réponse, etc. Les auteurs dans (Lenders et al., 2005) ont proposé un protocole de découverte. Ce dernier a adopté deux métriques pour sélectionner une instance de service qui sont : (1) *hop-count* : la distance entre demandeur du service/fournisseur de service et la capacité de service, (2) *CoS (Capacity of Service)* : exprime la capacité nominale d'une instance de service. Les auteurs dans (Engelstad et al., 2006) ont montré que la localisation obtenue des communications est un facteur qui conduit à améliorer la performance du réseau. Ils ont démontré par des simulations que le déclenchement de la re-sélection de serveurs après la détection des changements dans la topologie du réseau est très fiable en diminuant la congestion et les retards.

Les approches précitées considèrent l'impact du mécanisme de sélection du côté client sans prendre en considération les caractéristiques de l'appareil cible qui sont indispensables pour pouvoir garantir un déploiement correct et un meilleur fonctionnement des entités sélectionnées. Pour faire face à cette issue plusieurs travaux ont été développés.

Dû au fait que les dispositifs de réseaux mobiles ad hoc sont caractérisés par des ressources très contraintes et aussi la situation des ressources qui peut changer rapidement, les auteurs (Prochart et al., 2007) discutent la sélection de service pour le service composite. Ils ont effectué la tâche de sélection selon les informations dépendantes du service et les informations dépendantes du dispositif (ex. niveau de la batterie, la force du signal du réseau, etc.). Considérant que, dans (Rosa and Lucena Jr, 2011) une infrastructure de logiciel appelé *AppSpotter* qui permet la composition dynamique et automatisée de composants logiciels d'une application mobile est proposée. Sur cette infrastructure un composant nommé *component selector* sert à extraire à partir des *SPL assets* (software component stored) les composants logiciels qui répondent à un ensemble de critères où ces derniers représentent les caractéristiques de l'appareil mobile : la plateforme, l'écran, le clavier, etc. La tâche de sélection présentée en ces deux travaux (Prochart et al., 2007 ; Rosa and Lucena Jr, 2011) est entraîné par les caractéristiques de l'appareil mobile mais les critères de sélection ne sont pas les mêmes.

Cependant, nous avons abouti à une définition plus adéquate de notre axe de recherche pour le processus de sélection : « Afin d'assurer le déploiement correct et le bon fonctionnement d'une application mobile composite, nous avons besoin dans un premier temps de s'assurer que ses entités logicielles constituantes sont adaptables au contexte courant de l'appareil mobile où l'application composite va être déployée. La phase de sélection correspond à une opération de filtrage qui vise à sélectionner parmi toutes les entités logicielles qui implémentent une même fonctionnalité désirée celles les plus adaptées aux informations contextuelles de l'appareil mobile. Nous disons qu'une entité logicielle est conforme à l'appareil mobile cible si toute les contraintes d'exécution de cette entité sont satisfaites. »

■ La tâche de composition

Dans cette thèse, nous nous focalisons en particulier qu'à l'étape de composition après l'identification des besoins souhaités. Nous abordons dans un premier temps un scénario de composition. Un exemple de composition est : une combinaison de deux services web nommés *GeoIPService* et *GlobalWeather* dont l'objectif est de retourner à partir de la localisation les conditions météorologiques courantes. Le résultat de cette composition est un service composite de météorologie nommé *MyMeteo*. Ce composite agit comme suivant : le service *GeoIPService* obtient l'emplacement courant de l'utilisateur, après le service *GlobalWeather* fournit l'utilisateur par les conditions météorologiques courantes en se basant sur les coordonnées obtenues par le premier service.

Par conséquent, l'étape de composition correspond à l'établissement de la collaboration

entre les entités logicielles qui ont été sélectionnées. Le résultat final est la construction de la composition d'entités encapsulées sous la notion d'une entité composite (Lee et *al.*, 2008). Il s'agit des compositions de services à travers les orchestrations de services pour avoir un service composite, des compositions dans les approches à base de composants à travers les assemblages de composants afin d'arriver finalement à un composant composite ou également la composition des entités hétérogènes comme nous allons traiter dans ce manuscrit.

2.3 Composition de services/composants

■ Collaboration entre services

La collaboration entre services sert à établir des flots de travail et de données entre les services préalablement sélectionnés pour réaliser des fonctionnalités plus complexes où le flot de travail détermine l'ordre des invocations des différents services, les séquences parallèles ou continues, etc. tandis que le flot de données détermine les échanges de données entre ces différents services. L'organisation de ces flots s'effectue suivant deux types de schémas de collaboration, l'orchestration et la chorégraphie (Daniel and Pernici, 2006). Par conséquent, l'orchestration et la chorégraphie correspondent donc à deux types d'approches différentes de spécification de la collaboration qui organisent les communications entre les services (Rao and Su, 2005 ; Dustdar and Schreiner, 2005). De plus, les auteurs dans (Peltz, 2003 ; Benatallah et *al.*, 2005) ont déclaré que l'orchestration et la chorégraphie sont des moyens de concevoir la composition. Le concepteur de systèmes doit prendre en compte différents paramètres pour choisir le type de composition approprié pour construire son système. Les sous-sections suivantes décrivent chacun de ces types en définissant le noyau conceptuel qui spécifie ces opérations de composition. Nous allons aborder aussi quelques définitions présentées dans la littérature données par les travaux portant sur la composition de services.

● *Orchestration*

Le mécanisme d'orchestration nécessite que l'ensemble des services qui seront utilisés dans la composition existent au préalable. Ainsi, l'enchaînement d'exécution de ses services doit être prédéfini afin de réaliser une tâche précise. Le mécanisme opératoire de l'orchestration se déroule suivant deux étapes : Dans un premier temps un client (logiciel ou humain) doit transmettre une requête au moteur d'exécution (Moteur) pour déclencher l'opération d'orchestration. Après, ce moteur appelle les services selon l'ordre d'exécution défini préalablement comme illustré dans la Figure 2.4(a) (Lopez-Velasco, 2009).

Par conséquent, l'orchestration de services exige de définir l'enchaînement des services selon un canevas prédéfini (script d'orchestration) qui décrit les interactions entre ces services en identifiant les messages et en spécifiant les ordres d'invocation. Le module qui exécute le script d'orchestration de services est appelé un moteur d'orchestration où ce dernier représente une entité logicielle qui joue le rôle d'intermédiaire entre les services. Donc, le moteur d'orchestration a pour objectif d'appeler les services identifiés suivant le script d'orchestration défini. Brel dans son travail de recherche (Brel, 2013) a déclaré que l'orchestration peut être vue comme une composition ascendante tandis que Hock-Koon dans son travail de recherche présenté dans (Hock-Koon, 2011) a déclaré que l'orchestration correspond à un schéma de collaboration centralisée où une seule entité réunit l'ensemble des appels aux différents services de la composition comme illustré dans la Figure 2.4(b). Cette entité s'occupe de l'invocation des services dans un ordre prédéfini. Elle a pour objectif de récupérer les différents résultats et de transférer les données pertinentes pour l'exécution correcte des services invoqués. L'entité centralisant est associée à la notion de service composite tandis que les services invoqués représentent les constituants de ce composite. L'orchestration peut

être vue comme une composition verticale où les communications existent uniquement entre composite et constituants.

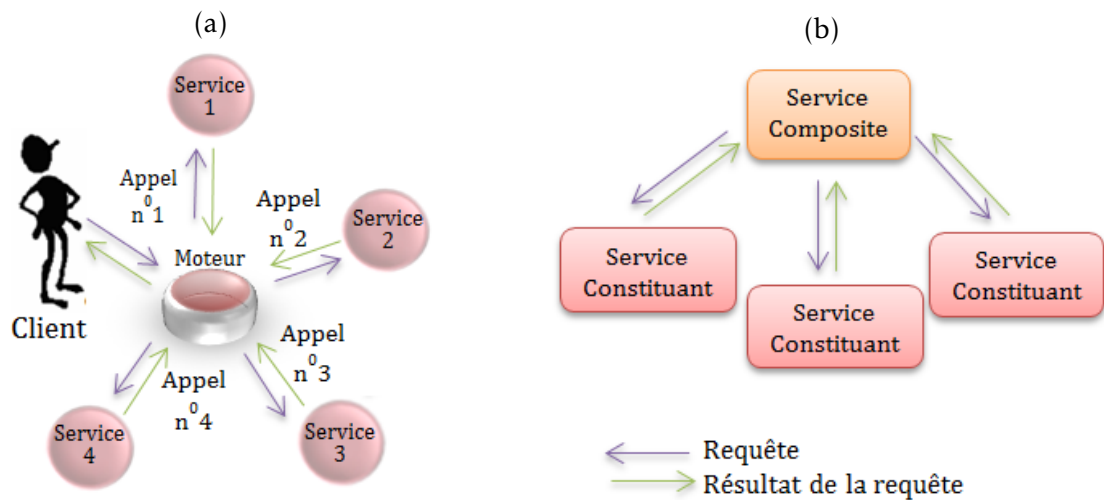


FIGURE 2.4 – Orchestration de services (Lopez-Velasco, 2009 ; Hock-Koon, 2011)

Plusieurs autres définitions ont été présentées dans la littérature pour la notion d'orchestration. Les auteurs dans (Barros et *al.*, 2006) définissent l'orchestration comme un ensemble de processus exécutés dans un ordre prédéfini afin de répondre à un but. Ce type de composition permet de centraliser l'invocation des services web composants. Chaque service est décrit en terme d'actions internes. Les contrats entre deux services sont constitués selon le processus à exécuter. D'après Peltz dans (Peltz, 2003), l'orchestration de services web consiste en la programmation d'un moteur qui appelle un ensemble de services web selon un processus prédéfini. Ce moteur définit le processus dans son ensemble et appelle les services web (tant internes qu'externes à l'organisation) selon l'ordre des tâches d'exécution. L'orchestration permet de mettre en relation plusieurs services en précisant d'une part le flux de contrôle et d'autre part le flux de données. Ces orchestrations sont clairement définies de bout en bout avec une et unique activité d'entrée (point de départ de l'orchestration) et une et unique activité de sortie (point de fin de l'orchestration) (Brel, 2013).

L'orchestration de services (notamment les Web Services WS-*) avec la possibilité d'exécuter les opérations des services en séquences et d'une manière parallèle en utilisant des boucles, des conditions, etc. peut être effectué à l'aide de BPEL4WS (Process Execution Language for web service)(Khalaf et *al.*, 2003). Ce dernier est un langage fondé sur l'XML destiné pour exécuter les procédures de l'entreprise et décrit le processus de composition des services web élémentaires. De plus, la description OWL-S permet de décrire directement les relations entre différents services d'une manière très proche à WS-BPEL.

• Chorégraphie

En effet, l'exécution du processus de chorégraphie n'est pas régie de manière statique comme dans une composition de type orchestration. Dans une chorégraphie, à chaque étape de la composition un service doit choisir le service qui lui succède et qui fait aussi partie de la composition. La composition de type chorégraphie n'est pas connue, ni décrite à l'avance c'est pour cela qu'elle est appelée composition dynamique (Peltz, 2003). La Figure 2.5(a) illustre une vue générale d'une composition de services de type chorégraphie. Dans ce cas-là, un client qui peut être soit un logiciel ou humain établit une requête. Cette dernière sera satisfaite par l'exécution automatique de l'ensemble de services qui représente la composition voulue.

L'exécution automatique se déroule comme suit : la requête de l'utilisateur est transmise au premier service (S1) qui est exécuté. Après, le S1 découvre le service qui lui succède.

Le processus de découverte repose, selon les cas, soit sur une recherche dans un registre local ou public soit sur une découverte globale sur le *Web* à l'aide d'ontologies. Une fois le service découvert, les deux services (S1 et S2) échangent des messages afin de vérifier si leur communication est viable dans le cadre de la requête. Dans ce cas-là, le S1 demande l'exécution d'une méthode du S2 par l'intermédiaire d'un envoi de message (Requête de service). Si cette requête est acceptée par le S2, ce dernier envoie un message d'acceptation au S1 (Acceptation). Le S1 accepte le service proposé par le S2 en lui envoyant un message (Service admis) accordé (Acceptation) par ce second service. Si les échanges de messages sont concluants, le S1 peut invoquer le S2 dans le cadre de la composition et le résultat de l'action du S1 est transmis au S2 qui l'utilise comme paramètre d'entrée. Le processus d'implémentation de la composition est identique pour chaque étape (S2 et S3). Le S4 termine le processus et le résultat de son action est transmis au client (Lopez-Velasco, 2009).

Dans ce type de composition, chaque service web a ses propres opérations à exécuter dans leur ordre. Les services web interagissent entre eux par des messages pour définir la séquence des messages qui peut être utilisée par la suite pour invoquer les services appropriés à une requête client. La chorégraphie décrit un aspect collaboratif entre une collection de services pour atteindre un but commun. En outre, Hock-Koon a déclaré que la chorégraphie correspond à un schéma de collaboration distribué où les échanges de messages se font directement entre les services en coopération afin d'effectuer une tâche particulière vers un but défini (cf. Figure 2.5(b)). Cette coopération correspond à une composition horizontale où les communications sont exclusivement entre services constituants.

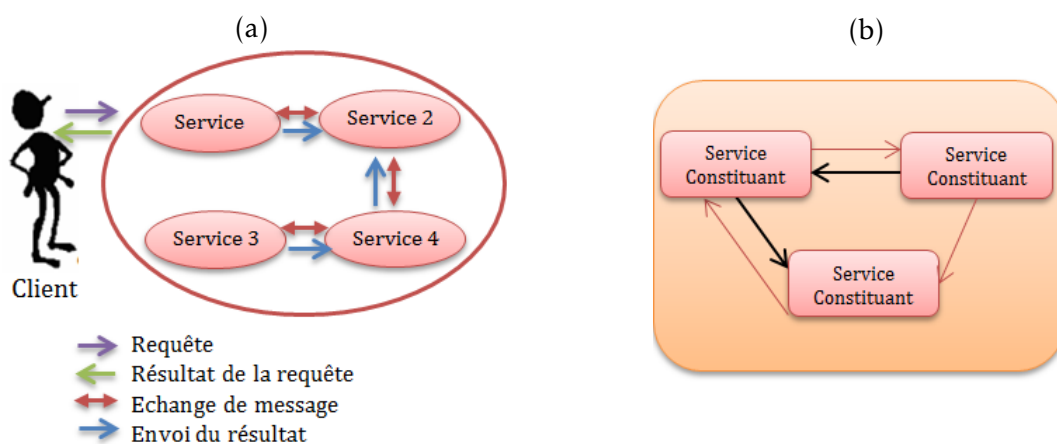


FIGURE 2.5 – Chorégraphie de services (Lopez-Velasco, 2009 ; Hock-Koon, 2011)

Ainsi pour la notion de chorégraphie, plusieurs autres définitions ont été présentées dans la littérature. D'après (Peltz, 2003) la description de chaque service web intervenant dans la chorégraphie inclut la description de sa participation dans le processus. De ce fait, ces services peuvent collaborer à l'aide de messages échangés afin de savoir si tel ou tel service peut aider dans l'exécution de la requête. Chaque service peut communiquer avec un autre service par l'intermédiaire d'échange de message. D'après (Barros et al., 2006), la chorégraphie permet de décrire la composition comme un moyen d'atteindre un but commun en utilisant un ensemble de services web. La collaboration entre chaque service web de la collection (faisant partie de la composition) est décrite par des flots de contrôle. (Benatallah et al., 2005) ont dit que pour concevoir une chorégraphie, les interactions entre les différents services doivent être décrites. La logique de contrôle est supervisée par chacun des services intervenants dans la composition. L'exécution du processus est alors distribuée.

Plusieurs langages ont été proposés pour générer la composition suivant le mécanisme de chorégraphie parmi lesquels nous citons :

- *WSCI (Web Services Choregraphy Interface)* : est un standard développé par l'association entre BEA Systems, Intalio, SAP, Sun Microsystems. WSCI est basé sur le standard XML pour décrire la séquence des messages échangés entre les services web participant dans une interaction chorégraphique et fonctionne en conjonction avec le WSDL ou d'autres langages pour la description des services similaires au WSDL. En outre, il répond à la question : « Comment deux API(s) qui ont été des services web permettent de coopérer entre eux ? », ce langage prend en charge la dépendance entre les messages échangés, les règles de séquence, la corrélation et décrit les collections des messages échangés entre l'interaction des services web.
- *WS-CDL (Web Services Choregraphy Description Language)* : est un langage fondé sur l'XML pour décrire une collaboration pair-à-pair entre deux ou plusieurs parties. Il décrit le comportement observable du service web et ces clients ou le service web qui interagisse avec lui par la description des messages échangés entre eux.

■ Assemblage de composants

Au départ, le modèle composant et l'architecture logicielle étaient peu liés. Depuis les recherches effectuées dans cet axe, Medvidovic et Taylor (Medvidovic and Taylor, 2000) les ont permis de se rapprocher et même de se compléter. Parmi leurs recherches ils ont constaté deux catégories de travaux, la première essaye de fournir les briques de base pour la construction d'entités logicielles réutilisables. La seconde, se concentre sur la description de l'assemblage de ces briques et donc sur leurs interactions afin de construire des applications de qualité. Parmi les objectifs premiers de ces auteurs est d'offrir des langages de haut niveau afin de décrire l'opération d'assemblage de composants. Les recherches effectuées sur l'architecture logicielle ont introduit les notions de connecteurs et de configuration. Malgré la très grande diversité entre les notions, les trois concepts de composant, connecteur et configuration sont généralement acceptés comme essentiels.

Les composants sont une vision qui met en avant l'opération de composition où cette dernière reflète la notion d'assemblage des composants pour réaliser un logiciel plus complexe. La composition dans une approche à base de composant reflète la définition explicite et la modélisation de l'architecture conceptuelle d'un système logiciel, comprenant au minimum : des composants, des interfaces, des connecteurs et des configurations architecturales.

Un composant logiciel représente une unité de calcul ou de stockage de données à laquelle est associée une unité d'implémentation. Il peut être aussi petit qu'une procédure simple ou aussi grand qu'une application. Il est l'élément de base et doit être composable, auto descriptif, réutilisable, configurable et autonome (Barais, 2005). Un serveur, une base de données, une fonction mathématique sont des exemples de composants. Un tel composant logiciel a des interfaces qui représentent les points de communication avec le monde extérieur. Une interface spécifie les services (message, opération ou variable) que peut un composant fournir comme elle peut représenter un service requis par ce composant.

Les auteurs dans (Szyperski et al., 1999) ont défini le composant comme un outil pour faciliter la réutilisation, la composition, le déploiement. Ce composant logiciel est donc une unité de composition qui a, par contrat, spécifié uniquement ses interfaces et ses dépendances explicites de contextes. Il peut être déployé indépendamment et est sujet à composition par des tierces entités.

L'interaction entre composants reflète un prédicat d'interaction nommé connecteur. Ce dernier sert à spécifier qu'un *composant 1* interagit avec *composant 2* où un service fourni par un *composant 1* via son interface fournie sera un service requis par le *composant 2* via son interface requise. Un connecteur est un bloc de construction destiné pour exprimer les

interactions entre composants ainsi que les règles qui gouvernent cette interaction. Autrement dit, les connecteurs sont des entités architecturales qui relient un ensemble de composants et agissent en tant que médiateurs entre eux. Par exemple, les connecteurs peuvent prendre des formes simples d'interaction, comme des appels de procédure ou l'émission d'évènements ou des valeurs/variables. etc. Ces connecteurs peuvent également représenter des interactions complexes ou être associés avec des médiateurs dans le cas d'une coordination hétérogène au niveau des données échangées entre composants (Derdour et al., 2010a ; Derdour et al., 2010b). Les connecteurs peuvent ne pas correspondre à des unités de compilation comme pour les composants.

Cependant, les spécifications des connecteurs peuvent également contenir des règles pour implémenter un type spécifique d'un connecteur. Comme le souligne (Oussalah et al., 2004 ; Mehta et al., 2000), les connecteurs d'assemblage peuvent être classifiés en trois catégories : (1) Les connecteurs implicites comme ceux par exemple de Darwin. (2) Les ensembles énumérés de connecteurs prédéfinis comme ceux par exemple d'UniCon (Shaw et al., 1995). (3) Les connecteurs dont les sémantiques sont définies par les utilisateurs comme ceux par exemple de Wright.

- **Premier groupe** : ce groupe d'ADLs ne réifie pas le concept de connecteur comme entité de première classe. Les interactions entre les composants sont identifiés en terme de liaisons directes (*bindings* en anglais) entre leurs interfaces. Ces liaisons sont modélisées en ligne et ne peuvent en aucun cas être renommées ou réutilisées. Ceci est le cas des interactions de Darwin (Gomaa and Farrukh, 1998) et de Rapide (Luckham et al., 1995).
- **Deuxième groupe** : les ADLs de ce groupe proposent des types de connecteurs énumérés. Généralement ces types de connecteurs ne peuvent pas être instanciés ni être modifiés. Le concepteur ne peut pas utiliser d'autres connecteurs en dehors de ceux définis par l'ADL lui-même. Par exemple, un connecteur dans UniCon est spécifié par un protocole. Un protocole est défini par un type de connecteurs, un ensemble de propriétés et des rôles typés qui servent comme points d'interaction avec les composants. Les types de connecteurs sont : FileIO, Pipe, Procedure Call , RPC, Scheduler, RT, Data Access et PL Bundler.
- **Troisième groupe** : les ADLs de ce groupe considèrent les connecteurs comme entités de première classe au même titre que les composants. Par exemple, dans le langage Wright les connecteurs sont définis comme un ensemble de rôles et une glu. Chaque rôle définit le comportement d'un participant dans l'interaction. La glu définit comment les rôles interagissent entre eux.

Une configuration reflète un ensemble de composants qui peuvent être élémentaires ou composites regroupés dans une seule entité nommée composant composite. Le modèle de composant définit « Qui sont les composants ?, Comment ils peuvent être construits, Comment ils peuvent être composés ou assemblés et Comment ils peuvent être déployés ».

Par conséquent, les compositions de composants logiciels passent par la construction d'assemblages de composants. Ces assemblages décrivent les composants composites qui permettent, comme pour les approches d'orchestration de services, de connecter plusieurs composants entre eux afin d'en former un nouveau. Cette combinaison est effectuée à travers les interfaces logicielles des composants constituants où une interface logicielle fournie étant connecté à une interface logicielle requise. Le composant composite obtenu expose les interfaces logicielles requises nécessaires à son opérationnalisation et les interfaces logicielles fournies relatives aux fonctionnalités de plus haut niveau qu'il propose. Afin d'établir une telle composition, un langage de composition est nécessaire (Fractal (Bruneton et al., 2006), SCA (Beisiegel et al., 2005 ; Mietzner et al., 2008) et SLCA (Hourdin et al., 2008)). Ce langage va

permettre de mettre en place la "colle" (*glue code*) qui est indispensable à la communication entre composants généralement via des composants spéciaux appelés les *Connecteurs* qui peuvent être classés en différentes catégories (Mehta et al., 2000). Ces connecteurs ont pour objectif de transmettre les données d'un composant à un autre, de prendre en charge la coordination entre composants, de convertir des données ou des événements, etc.

2.4 Composition basée composants VS composition orientée services

Avant de s'engager dans la comparaison entre la composition d'applications à base de composant et la composition orientée services plus précisément l'assemblage de composants et la collaboration entre services, il est important d'attirer l'attention que la composition d'applications notamment dans un environnement mobile est influencée par les informations contextuelles de l'environnement d'exécution. Ces informations de contexte concernent les objets de composition qui sont les entités logicielles constituantes et le dispositif mobile qui sera utilisé pour déployer l'application composite.

Par conséquent, en se basant sur les définitions de la composition présentée précédemment ainsi les différentes représentations proposées pour définir une telle application nous proposons notre propre définition pour le mécanisme de composition : « La composition est le processus de construction d'une nouvelle entité logicielle à partir d'autres entités pré-existantes en respectant les ressources limitées des appareils mobiles ainsi la variabilité de ses états d'exécution où l'entité résultat est dite composite. Ce processus permet de spécifier à un haut niveau d'abstraction l'ensemble des entités logicielles à composer afin d'atteindre l'objectif attendu. Cet assemblage comporte une phase d'identification des entités logicielles et de spécification de leurs interactions conformément aux descriptions et aux accords entre entités. »

Plusieurs travaux ont étudié en détail la comparaison entre les composants logiciels, les services et les objets ainsi les paradigmes de représentation d'applications en utilisant ces différentes entités logicielles. Nous nous focalisons dans ce travail de recherche sur deux types d'architecture permettant de définir le noyau fonctionnel d'une application présentée par CBSE et SOSE. Nous avons choisi ces deux types d'architecture qui sont en fait très proches puisque chacun va dans un sens rendre un service pour composer conceptuellement une application mobile, i.e. modéliser l'application composite. Dans notre étude, une application mobile va être construite autour d'un ensemble de services et/ou d'un ensemble de composants. En s'inspirant de ces études de comparaison nous présentons dans les sous sections ci-dessus les informations clés des comparaisons déjà présentées à base desquelles nous avons élaboré notre nouveau paradigme hétérogène.

■ Service VS composant

Un composant est dit off-the-shelf (sur étagère) (Di Nitto et al., 2008) par adoption d'une pièce technologique (le composant) qui est disponible pour les développeurs où ils peuvent récupérer le bloc logiciel de ce composant et s'occuper juste de l'incorporer suivant leur besoin. Contrairement au service qui est centré sur l'utilisation d'une fonction fournie par un tiers. Un tel service est exécuté sur une plateforme à distance sous la responsabilité du fournisseur. En revanche, un consommateur de services exploite le résultat issu de l'invocation du service cible. Donc, il a uniquement besoin de l'interface et de la connexion adéquate pour accéder à cette plateforme et les seules informations qui sont nécessaires pour lui sont juste comment accéder à cette plateforme et comment invoquer ce service.

Une des caractéristiques très importante d'un service est son indépendance avec les technologies d'implémentation où celui-ci sera accessible et utilisable sans aucune hypothèse

sur son implémentation ou sur la façon de son utilisation, de la part des bénéficiaires potentiels. Cela n'est pas le cas pour les composants logiciels. De ce fait, il existe un très grand nombre de modèles de composant (Crnkovic et *al.*, 2007) et pour développer une nouvelle application, l'architecte doit choisir un modèle spécifique et n'utiliser que des composants respectant ce modèle car la collaboration entre modèles différents est très difficile (Crnkovic et *al.*, 2006). Face à ce constat, plusieurs modèles de composant complexes qui offrent des fonctionnalités similaires à celles des services (ex. programmation distribué, découverte dynamique, interopérabilité, etc.) ont été proposés.

La capacité des entités logicielles d'être distribuées/réutilisées ainsi que leur plus grande autonomie nécessite que ces entités utilisent le concept de *black boxe entities*. Cette notion limite l'interaction avec un service ou un composant juste à l'utilisation de ses interfaces fournies et requises tout en interdisant l'accès direct à son implémentation. Dans les modèles hiérarchiques, les composants composites sont le plus souvent *white boxes* en permettant les modifications de ses structures internes mais aussi sans voir et avoir accès à l'intérieur de ces composants de base. Un composant peut être utilisé plus d'une fois c'est-à-dire que : contrairement aux services où l'instance d'un service est unique, il peut y avoir plusieurs instances d'un composant.

■ Assemblage de composants VS Collaboration de services

La composition d'applications dans le contexte de l'ingénierie logicielle basée composants ou orientée services a pour objectif de maximiser la ré-utilisabilité en terme de modularité qui est directement issue de l'objet et d'améliorer le développement d'applications par assemblage de composants logiciels ou l'orchestration/chorégraphie de services.

CBSE et SOSE sont deux paradigmes très similaires qui se basent sur la construction d'applications à partir d'entités logicielles existantes, composants ou services (Amirat et *al.*, 2014). Ils partagent aussi le même processus de développement global qui se compose des activités d'identification des entités logicielles (composant ou service) qui répondent aux besoins, puis de coordination de ces entités pour construire l'application composite finale. Par conséquent, ils reposent sur les mêmes notions de composition pour la construction de nouvelles entités dites composites à partir de l'existant afin de garantir une approche homogène où toute entité peut être vue comme un composant ou un service. Cette approche permet une manipulation des structures de description claires en terme de composition de composites.

Le CBSE repose à la phase de conception sur les notions de type de configuration et type de composant composite et au *runtime* sur leurs instances configurations et composants composites. Néanmoins, une notion de service abstrait existe dans certaines approches (Cavallaro et *al.*, 2009). La plupart des travaux existants se réfèrent à un service comme une entité du *runtime* (Stojanovic and Dahanayake, 2005) où l'extension de service composite via le mécanisme de composition de services est principalement au *runtime*. Dans ce travail, nous référons la notion du service abstrait à l'expression des besoins recherchés par l'architecte pour définir l'application souhaitée mais en fait les services sont concrètement disponibles dans le système pour répondre à ces besoins.

La plupart des travaux existants considèrent le service composite comme l'exécution par un moteur de composition d'un schéma de collaboration entre services ou certaines d'autres (Geebelen et *al.*, 2008) introduisent des notions d'instanciation de schéma de collaboration à partir de *templates* abstrait qui les décrivent. Hock-Koon dans (Hock-Koon, 2011) a choisi de prendre en compte cette représentation similaire à l'OO (Orienté Objets) avec des types de schéma de collaboration comme entités de la phase de conception et des instances de schéma de collaboration comme entités du *runtime*.

Une critique que l'on apporte à l'architecture à base de composants est son caractère statique

après la composition du système. En effet, une fois les composants choisis et couplés ils ne sont que difficilement changés durant l'exécution.

Un point de différence importante entre l'assemblage de composants et l'orchestration de services c'est que dans la plupart des cas les composants cachent le flux de données à l'intérieur d'eux-mêmes. L'interaction entre ces entités logicielles constituant de l'application à composer met en exergue une notion clé qui est le *couplage*. Brel dans (Brel, 2013) a identifié cette notion comme un des points de rupture clé entre les deux paradigmes CBSE et SOSE. Comme ils ont déclaré que réduire le couplage garantit un certain nombre de bénéfices intuitifs en terme d'isolation des erreurs, de facilitation d'ajouts et de retrait d'entités réutilisées, re-configuration, etc.

Les types de composition présentés par ce cadre de travail mettent en place un couplage faible entre les entités constituant qui permettent une grande réutilisation de celles-ci. La différence entre les deux mécanismes de composition c'est que l'orchestration de services sera explicite où les cheminements des données et leurs transformations doivent être définis au préalable. Contrairement à l'opération d'assemblage où ceux-ci peuvent être vus comme des boîtes noires et le cheminement des données n'est pas a priori connu.

■ Composition hétérogène : multi-paradigmes

Contrairement aux composants logiciels qui répondent bien au problème de l'adaptation d'application, les services correspondent plus à la manipulation d'hétérogénéité et la distribution d'entités. De plus, l'interopérabilité dans la communication des protocoles, hardware, et les langages de programmations peuvent être manipulés par des services web. La réactivité a besoin d'un mécanisme pour diffuser des informations aux services de l'environnement quand il sera disponible et en outre favoriser le découplage et une meilleure adaptabilité. Cependant, la création d'applications basées uniquement sur les services peut se révéler complexe et difficilement adaptatives puisque les services découverts ou plus exactement leurs interfaces doivent être connus au moment de la conception. Par ailleurs, une application basée uniquement sur les composants peut difficilement gérer les variations du contexte d'exécution et les communications entre dispositifs. Afin de tirer profit des avantages présentés par les composants logiciels et les entités logicielles service, une combinaison entre l'approche à base de composants et orientée services est apparue nécessaire. Face à ce constat, plusieurs études (ex. équipe INRIA) ont mis l'accent sur l'aspect technologique de la gestion des hétérogénéités où plusieurs systèmes multi-paradigmes ont donc vu le jour (ex. SOA a.k.a. SOA 2.0 qui utilise les services et les événements, SCA (Beisiegel et *al.*, 2005), SLC (Hourdin et *al.*, 2008), FROGi (Desertot et *al.*, 2006)). Les approches qui empruntent et combinent les éléments conceptuels ou techniques issus du CBSE ou du SOSE sont qualifiées d'hybride. Cependant, nous présentons parmi eux ceux qui sont liés à notre travail de recherche et qui reflètent précisément la notion de composition (i.e. dédiés à la composition) et de l'hétérogénéité.

SCA (Service Component Architecture) permet de définir une architecture de composants et de services en utilisant des composants pour manipuler l'orchestration de services et donc créer un service composite. Autrement dit, il correspond à la création d'applications orientées services à base d'assemblages de composants SCA. L'atout majeur de SCA est de supporter les variabilités dans l'implémentation des composants SCA ou de leurs communications qui peuvent être réalisées par de nombreuses technologies différentes (Java, C++, WSDL, RMI, etc.). Ainsi, SCA illustre toute la complexité des entrelacements entre les composants et services. Il est présenté comme une solution simple à l'implémentation de systèmes orientés services et repose sur une architecture à base de composants où l'interface fournie est elle-même qualifiée de service. Cela prouve sa capacité de gérer l'hétérogénéité et la dynamique nécessaire à l'orienté services. En effet, sa brique architecturale de base (composant SCA) tente d'atteindre le même degré d'encapsulation qu'un service dans sa capacité à gérer des ressources hétérogènes.

Enfin, les versions en cours des outils SCA apportent une dynamité accrue dans l'établissement des connexions entre composants SCA, et entre les composants SCA et leur implémentation. Par exemple, l'outil actuel (Eclipse Soa tools 2009) permet de faire communiquer simplement et de façon transparente aux développeurs des composants SCA implémentés dans un certain nombre de technologies différentes telles que Java, EJB, services web, etc.

Un autre modèle classé dans la catégorie des approches hybrides est SLCA (Service lightweight Components Architecture). Ce dernier représente un modèle architectural pour la composition de services en se basant sur l'assemblage de *lightweight Components*. Un modèle SLCA est basé sur les environnements d'exécution matériel et logiciel évolués dynamiquement où ces environnements reflètent un ensemble de ressources qui sont autant des entités logicielles/matérielles pour lesquelles l'application elle s'adapte à leur apparition et à leur disparition.

Son objectif principal est de définir une architecture dynamique pour la composition de services tout en faisant face aux problèmes des systèmes ubiquitaires en tirant profit de plusieurs paradigmes existants. Ce type d'architecture repose sur trois paradigmes principaux : *Web service oriented architecture*, *Lightweight assembly of components* et *Events*. Les services web composites sont créés via un assemblage dynamique des composants à boîte noirs (*black box components*) exécutés dans un conteneur local qui ne fournit pas obligatoirement des services techniques. Par conséquent, la dynamité est donc fournie et la ré-utilisabilité est augmentée. SLCA donc définit un modèle architectural compositionnel basé sur des événements afin de concevoir des services web composites.

- *La gestion d'hétérogénéité des données échangées*

Dans un monde de composition, le défi de l'hétérogénéité ne peut pas porter que sur les objets de composition mais il peut aussi toucher les données échangées entre ces entités à composer. La gestion des hétérogénéités qui portent sur les données échangées représente la capacité à s'abstraire de la nature hétérogène des entités ou comment invoquer et réutiliser une entité qui fournit les bonnes fonctionnalités mais qui ne respecte pas les contraintes de réalisation prévues par l'architecte.

Supposant que nous voulons composer deux entités *Acquire Photo/Read Barcode* où l'entité *Acquire Photo* produit une image de type jpg représentant le produit à acheter tandis que l'entité *Read Barcode* nécessite une image de type WebP pour qu'il puisse effectuer sa tâche en traitant le code-barre indiqué sur cette image. Ce scénario de composition présente alors un défi d'hétérogénéité entre les données échangées. L'incompatibilité de ces données échangées nécessite forcément un mécanisme de médiation pour permettre la compréhension respective de la donnée entre ces entités. Par conséquent, cet aspect se focalise sur la compatibilité des données échangées d'un point de vue sémantique et syntaxique c'est-à-dire la bonne compréhension du sens attaché aux données (ex. un entier qui représente une date peut être au format anglophone (mois, jour puis année) ou francophone (jour, mois puis année), une valeur exprimée en *euros* ou en *dollars*, etc.)

Hock-Koon a traité cette issue pour son service composite proposé, i.e. la gestion des hétérogénéités. Cette gestion représente la capacité du composite à pouvoir communiquer avec les services métiers et de solliciter leurs fonctionnalités de façon correcte. Elle regroupe les préoccupations d'invocation qui reflètent le déclenchement de l'exécution des services constituants et de médiation qui représentent la capacité du composite à assurer la bonne compréhension des données échangées entre ses services constituants.

Dans (Kalasapur et al., 2007), les auteurs proposent une méthode de composition de services qui cible les problèmes d'hétérogénéité des données. Leur approche traite ce problème en

amont au niveau des processus de publication et de découvertes de services en se basant sur une organisation particulière des services disponibles. De fait, les services sont organisés dans un graphe qui représente l'ensemble des compositions possibles. Lors de l'incompatibilité des données entre les services, le système utilise le graphe des compositions pour identifier une succession de services capable d'assurer les modifications nécessaires sur la donnée à priori incompatible. Cette solution repose sur le choix des solutions alternatives évitant les problèmes d'hétérogénéité découverts.

Makhlouf dans son travail de recherche (Derdour, 2010) a attaqué ce problème au niveau des architectures logicielles multimédias. D'après lui, l'architecture logicielle dans une application multimédia est décrite comme un ensemble de composants manipulant plusieurs types de données multimédias avec des contraintes spécifiques qu'elles devraient être prises en considération lors de la conception architecturale. Cependant, il a déclaré que le problème de l'hétérogénéité est basé sur les flux des données multimédias échangées. Face à ce constat, il a proposé un méta-modèle MMSA (Meta-model Multimedia Software Architecture) pour les architectures multimédias où ce méta-modèle permet de décrire les systèmes multimédias comme une collection de composants qui manipulent différents types et formats des données multimédias et interagissent avec eux via des connecteurs d'adaptation. Son modèle proposé vise à résoudre le problème d'hétérogénéité entre les données échangées entre composants en se basant sur quatre types d'interfaces selon les flux des données existantes (image, son, text et vidéo). Ce modèle sert à faciliter l'opération d'adaptation entre médias du même type (ex. *image* vers *image*) ou entre différents types de médias (ex. *texte* vers *son*).

3 Les approches de composition : Synthèse

Comme nous l'avons indiqué précédemment, la contribution principale de ce travail de recherche reflète la composition conceptuelle d'entités logicielles déjà existantes pour construire des applications mobiles adaptatives. Face à ce constat, nous allons sélectionner et présenter quelques approches de composition déjà existantes touchant les notions clés liées à notre travail. L'objectif est de faire apparaître plus précisément la problématique abordée, les motivations derrière ce travail de recherche et d'aborder les objectifs visés.

Les auteurs dans (Chakraborty et al., 2005) ont décrit certaines issues liées à la composition de services dans un environnement mobile. Ils ont présenté un protocole de composition de services distribué pour les environnements mobiles qui prend en considération la mobilité, changement dynamiques dans la topologie de service et les ressources des dispositifs. Ils se sont concentrés principalement sur une architecture distribuée pour faciliter la tâche de composition mais en négligeant ses capacités d'adaptation.

Dans (Wu et al., 2007), les auteurs ont présenté une approche automatique pour la composition de services web tout en attaquant le problème de l'hétérogénéité des processus et les hétérogénéités des données en utilisant un planificateur et un médiateur de données. En revanche, les auteurs dans (Hock-Koon and Oussalah, 2010) étaient censés réifier les notions pertinentes des mécanismes de composition existants dans un méta-modèle de service composite. Ce méta-modèle définit toutes les fonctionnalités entrelacées et fournit une vision globale et explicite de la composition de service. En outre, cette approche permet la spécification du processus d'auto-composition dont la composite a la capacité de modifier dynamiquement son architecture et ses logiques de composition selon le contexte de l'environnement.

Dans (Rosa and Lucena Jr, 2011), les auteurs ont créé une infrastructure de logiciel appelé *App-spotter* qui rend possible la sélection et la composition dynamique et automatisée des composants logiciels pour la création d'applications mobiles. Ils ont proposé un mécanisme

pour la sélection de composants logiciels en se basant sur les caractéristiques de l'appareil mobile. Dans (Furno and Zimeo, 2014), les auteurs ont présenté un modèle pour concevoir des services sensibles au contexte qui peut être exploité comme un domaine flexible pour générer automatiquement des compositions sensibles au contexte au moyen d'un outil spécifique. En outre, l'outil exploite une extension de la définition du problème qui comprend également la représentation du contexte pour étendre les services.

Chacune de ces approches tente de résoudre quelques problèmes particuliers de différents points de vue : la composition en terme de composants, de services ou d'entités logicielles hétérogènes ; l'environnement d'exécution : dispositif mobile ou autre (ex. les ordinateurs) ; les capacités d'adaptation en terme de construire des applications contextuelles ou non adaptatives ; comme illustre le Tableau présenté ci-dessous (cf. Tableau 2.2).

Tableau 2.2 – Un cadre de comparaison d'approches de composition

	Objets de composition			Type de composition		Capacité d'adaptation		Approche définit au	
	Entité Composant	Entité Service	Entités Hétérogènes	Micro Composition	Macro Composition	Contexte Courant	Non adaptative	Niveau Architectural	Niveau Application
Chakraborty et al.(2005)		×		×		×		×	
Zixin et al.(2007)		×			×		×	×	×
Hock-Koon and Oussalah (2010)		×			×	×		×	
Ricardo and Vicente(2011)	×			×		×			×
Furno et al.(2014)		×			×	×		×	×
Notre Processus de Composition	×	×	×	×		×		×	

Ces multitudes d'approches avec ses caractéristiques souvent spécialisées n'ont pas une vision globale de la composition d'applications mobiles. Ces approches limitent la ré-utilisabilité de l'application composite pour un contexte spécifique qui ne pas pourra être celui qui est nécessaire, i.e. le contexte courant. Un besoin naissant est alors de pouvoir obtenir plusieurs versions de la même application composite désirée selon plusieurs informations contextuelles de différents appareils mobiles et donc la nécessité de prendre en compte les configurations hétérogènes offertes par eux ainsi que les ressources limitées et leur état d'exécution.

De plus, mis à la disposition de l'utilisateur, celui-ci peut avoir besoin de plusieurs applications pour combler ses propres exigences. Par conséquent, pour cause du contexte et des exigences des utilisateurs un besoin émergent est alors de combiner des entités logicielles déjà existantes tout en ignorant comment elles sont implémentées, i.e. indépendamment de leurs plateformes d'exécution, afin de tirer profit de leurs fonctionnalités. L'objectif est d'obtenir des applications mobiles comblant les exigences des utilisateurs et avoir des applications composites adaptables à son environnement d'exécution.

Notre travail de recherche est destiné à exprimer clairement les notions pertinentes de ces approches dans un processus de composition pour les applications mobiles suivant une approche de modélisation. Par conséquent, ce modeste travail représente la première tentative pour soutenir le développement d'applications mobiles contextuelles par la composition d'entités logicielles hétérogènes au niveau architectural. L'objectif de ce travail est donc

de proposer un regard conceptuel sur les applications mobiles hétérogènes incluant leur conception ainsi leur adaptation en fonction du contexte. La ligne colorée du Tableau reflète le positionnement de notre processus par rapport aux différentes approches de composition citées précédemment en fonction des différents critères proposés.

CONCLUSION

Dans ce chapitre, nous avons explicité dans un premier temps : le fonctionnement global d'une approche de développement dirigée par les modèles, toutes les notions clés liés à cette approche, l'intérêt de cette démarche de réalisation d'applications et plus précisément son importance pour le développement mobile.

Dans ce chapitre, nous nous sommes particulièrement intéressées au processus de composition. A ce stade, nous avons présenté plusieurs définitions proposées pour cette technique où cette dernière peut faire l'objet sur les composants ou bien les services. D'après ces définitions, nous avons constaté que la composition représente une tâche complexe qui fait intervenir plusieurs étapes intermédiaires. Face à ce constat, nous avons présenté chacune de ces étapes tout en discutant quelques travaux apportés pour chacune d'entre elles. La tâche de composition reflète le noyau principal de ce travail. Pour cela, nous avons expliqué en détail le mécanisme opératoire de cette tâche suivant deux différents types d'objets de composition. Ce chapitre fait intervenir quelques langages dédiés à la collaboration de services et les langages de description d'architectures qui sont destinés à effectuer l'assemblage de composants.

Ce chapitre a permis de présenter un cadre comparatif entre les objets de composition et aussi entre les techniques de composition proposées pour chacun d'entre eux ; pour y arriver à la fin à l'apparition des approches hybrides qui sert à tirer profit de plusieurs paradigmes existants ce qui conduit à l'émergence de la nouvelle notion *multi-paradigmes*. En plus de l'hétérogénéité du type des objets à composer, la tâche de composition présente un autre défi important qui est l'hétérogénéité qui porte sur les données échangées entre ces objets. Dans ce chapitre, nous avons abordé ce point en présentant les travaux qui traitent cette issue. Nous avons conclu ce chapitre par une comparaison de quelques approches de composition liées à notre domaine de recherche suivant un ensemble de critères bien définis.

Dans le chapitre qui suit, nous allons présenter l'aspect conceptuel du processus proposé comme nous allons traiter explicitement les contributions élaborées dans ce manuscrit. Ce chapitre présentera en détail le processus CAMAP que nous avons proposé pour construire des applications mobiles adaptatives.

CAMAP : Un Processus pour la Composition d'Applications Mobiles

SOMMAIRE: CHAPITRE 3

Introduction	68
1 Scénario de composition : ShopReview Mobile App	69
2 CAMAP : Le contexte global	71
3 CAMAP : Étapes et déroulement	72
4 La description du contexte de composition	74
4.1 La nécessité de décrire le contexte	74
4.2 Modélisation du contexte de l'appareil mobile cible	75
4.3 Modélisation du contexte des objets de composition	76
5 Modélisation de l'application mobile composite	78
5.1 La représentation fonctionnelle	78
5.2 La description architecturale	79
6 Mécanisme opératoire du processus : passage entre modèles	83
6.1 Du modèle fonctionnel au modèle fonctionnel raffiné	83
6.2 Du modèle fonctionnel raffiné au modèle architectural	85
6.3 Du modèle architectural au modèle architectural détaillé	86
Conclusion	88

INTRODUCTION

Il est largement reconnu que la plupart des activités humaines actuelles reposent sur l'utilisation des appareils mobiles. L'adoption récente et massive de cette technologie explique la demande croissante d'applications spécifiques qui peuvent accueillir la variabilité de ses environnements d'exécution. Par conséquent, les objectifs prioritaires durant la tâche de composition d'applications précisément pour les environnements mobiles sont la satisfaction des exigences de l'utilisateur et le respect des contraintes d'exécution de l'appareil mobile sur lequel la future application sera exécutée. Pour cause du contexte de l'appareil mobile et les besoins de l'utilisateur le développeur peut se trouver parfois forcé de composer des entités logicielles hétérogènes. Pour cela nous proposons un processus de composition qui consiste à atteindre ces objectifs tout en facilitant la tâche de composition pour le développeur et qui vise à faire face au défi de l'hétérogénéité que ce soit de la part des appareils mobiles cibles ou des différentes entités logicielles à composer. Dans ce chapitre nous allons introduire le socle conceptuel du processus proposé CAMAP tout en présentant les modèles d'ontologies permettant de décrire le contexte de composition, les différentes représentations architecturales dédiées pour définir l'application mobile composite à plusieurs niveaux d'abstraction et aussi les différents moteurs de transformation qui sont nécessaires pour réaliser le passage entre les modèles architecturaux proposés.

1 Scénario de composition : ShopReview Mobile App

Afin d'expliquer les motivations de notre travail de recherche, nous présentons ici les principaux problèmes qui ont été rencontrés et traités. Pour ce faire et pour clarifier les concepts clés du processus de composition que nous proposons dans ce manuscrit, nous introduisons dans cette section deux scénarios de composition pour une application nommée *ShopReview* discutée dans le travail de Cugola et *al.* (Cugola et *al.*, 2014). Cette application reflète la composition d'un ensemble de fonctionnalités à la fois assez simples dans leur fonctionnement et assez convenable pour illustrer nos apports.

Un utilisateur peut utiliser l'application *ShopReview* pour publier le prix du produit qu'il a trouvé dans certain magasin choisi parmi ceux qui sont près de sa situation géographique actuelle. Cependant, l'application mobile *ShopReview* fournit les magasins à proximité où le même produit est vendu mais à un meilleur prix (i.e. prix plus convenable). La seule correspondance entre le prix signalé par l'utilisateur et le produit est obtenue en exploitant son code-barres. Finalement, cette application permet également à l'utilisateur de partager son avis, i.e. diverses informations concernant les produits et les magasins, en utilisant un réseau social comme *Twitter*. Afin de construire cette application par le biais du mécanisme de composition, le développeur se concentre d'abord sur les différentes fonctionnalités souhaitées de l'application. Les fonctionnalités requises sont les suivantes :

- **Fun1** : Acquisition de la photo du produit à partir de la caméra du dispositif mobile.
- **Fun2** : Scanner le code-barres du produit.
- **Fun3** : Traduire le code-barres et obtenir le nom du produit.
- **Fun4** : Fournir le prix du produit via l'utilisateur (entrer le prix manuellement).
- **Fun5** : Récupérer via l'*Internet* les prix les plus courants offerts par les sites du e-commerces.
- **Fun6** : Obtenir l'emplacement courant de l'utilisateur.
- **Fun7** : Récupérer via l'*Internet* la liste des autres magasins à proximité qui offrent le produit à un meilleur prix.
- **Fun8** : Permettre à l'utilisateur de partager le prix du produit trouvé dans un certain magasin, i.e. partager des informations sur *Twitter*.

Comme premier scénario, nous supposons que le développeur a choisi de mettre en œuvre ces fonctionnalités grâce à des entités logicielles concrètes de type composant (i.e. SEs : Software Entities). Cependant, supposons que, selon les entités concrètes sélectionnées par le développeur, l'application mobile *ShopReview* ait besoin d'un GPS pour exécuter la Fun6 (localisation de l'utilisateur via le GPS) et du Wi-Fi pour Fun5, Fun7 et Fun8. En outre, elle a besoin d'un camera avec une fonction d'*autofocus* afin de pouvoir exécuter la Fun2 (la reconnaissance locale du code-barres exige que l'image du produit doit être prise par une caméra avec le mode d'*autofocus*). Par la suite, le développeur a choisi de déployer l'application obtenue sur l'appareil mobile cible dont son contexte d'exécution courant est : Has-GPS is GPS-disabled, Has-Wi-Fi is Wi-Fi-enabled, Has-fixedfocus-camera, BatteryLevel is 50% et Free-StorageCapacity is 500 MB.

Après avoir terminé la tâche de composition, nous avons constaté que l'application *ShopReview* obtenue ne fonctionne pas correctement sur l'appareil mobile cible pour les raisons suivantes : SE_2 ne permet pas la reconnaissance locale correcte du code-barres d'une

image acquise avec une caméra en mode *fixedfocus* et SE_6 ne peut pas être exécutée tant que l'état du GPS est désactivé. Ainsi, le choix de SE_2 et SE_6 pour implémenter les Fun2 et Fun6 sans prendre en considération le contexte courant du dispositif mobile limite l'usage de l'application mobile composée pour un contexte inapproprié (cf. Figure 3.1 - *Scénario de composition 1*). Notre processus de composition vise à faire face à ce problème qui peut survenir lors de la composition d'une application mobile tout en évitant le déploiement de l'application composite dans un environnement inapproprié. Maintenant, supposons que le développeur cette fois-là se concentre sur les contraintes d'exécution des entités concrètes qu'il va choisir pour implémenter les fonctionnalités identifiées où ces contraintes dépendent fortement des différentes informations contextuelles du dispositif mobile sur lequel il veut exécuter sa propre application mobile composite. Dû au fait que plusieurs entités concrètes peuvent être liées à la même fonctionnalité, le développeur peut alors rechercher d'autres entités logicielles concrètes pour implémenter les fonctionnalités requises. Cette fois-ci, il pourrait choisir d'attacher chaque fonctionnalité par l'entité concrète appropriée dont ses contraintes d'exécution sont conformes aux informations contextuelles de l'appareil mobile cible. Par exemple, parce que l'image du produit sera prise par un appareil mobile avec un camera en mode *fixedfocus*, le développeur peut alors choisir une entité logicielle de type service pour acquérir le code-barres du produit où cette entité concrète vise à appeler un service à distance pour traiter l'image acquise avec un algorithme de décodage flou spécial. Afin de construire une application assez puissante qui fonctionne correctement tout en adoptant la réutilisation de l'existant il peut aussi choisir une entité logicielle de type services qui permet de partager des informations sur *Twitter* (cf. Figure 3.1 - *Scénario de composition 2*). Par conséquent, un autre défi majeur que nous devons prendre en considération est la difficulté de permettre la collaboration des entités hétérogènes déjà existantes entre elles du fait qu'il est impossible de combiner des entités logicielles hétérogènes sans adaptation.

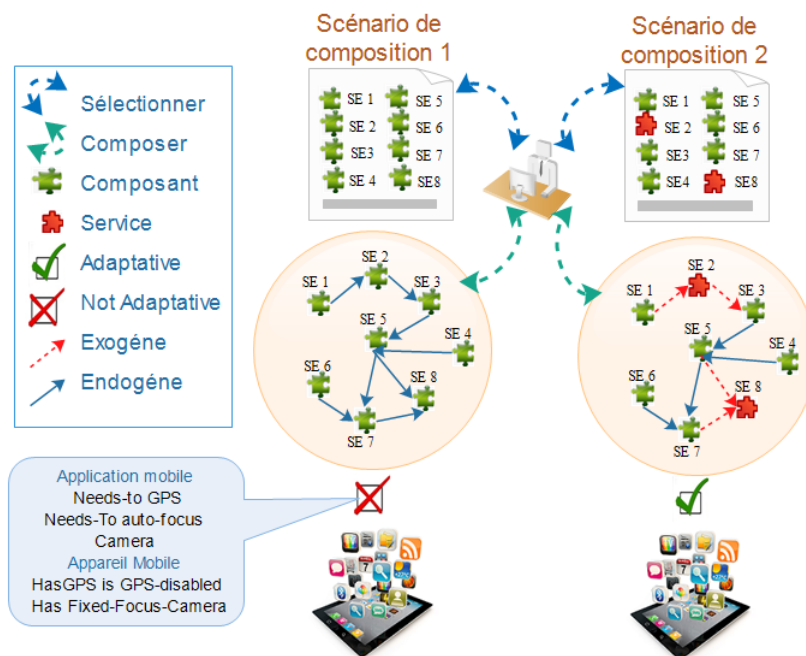


FIGURE 3.1 – Scénarios de composition pour l'application mobile *ShopReview*

Les scénarios de composition proposés pour l'application mobile *ShopReview* nous permet d'illustrer les différents modèles contextuels, les descriptions architecturales et les algorithmes proposés qui reflètent les contributions abordées dans ce manuscrit. Nous allons adopter cet exemple comme une petite étude empirique pour prouver l'efficacité du processus proposé.

2 CAMAP : Le contexte global

Notre travail vise à remédier aux problèmes rencontrés lors de la composition et qui sont illustrés par les scénarios décrits dans la section précédente. Nous rappelons ici que l'objectif principal visé par ce travail est de construire des applications mobiles adaptatives via la composition des entités logicielles pré-existantes selon les préférences des utilisateurs. Pour ce faire, nous commençons par identifier les fonctionnalités nécessaires tout en ignorant comment elles seront implémentées ; autrement dit sans aucune limitation ou contrainte sur l'utilisation de l'application souhaitée. Après, le choix des entités logicielles concrètes qui seront utilisées pour la mise en œuvre des fonctionnalités identifiées est effectué par rapport aux informations contextuelles de l'appareil mobile tel que la disponibilité d'un dispositif GPS, la présence d'une caméra avec une fonction d'*autofocus* ou le niveau actuel de la batterie, etc. A titre d'exemple, la découverte fonctionnelle et la sélection non fonctionnelle sont l'objectif de Surianarayanan et *al.* dans leur travail de recherche présenté dans (Surianarayanan et *al.*, 2015).

Dans notre travail de recherche nous supposons que les entités contextuelles à composer sont déjà sélectionnées. Cependant, le processus proposé vise à composer ces entités contextuelles où l'intégration des médiateurs est indispensable dans le cas d'une coordination hétérogène. Sur cette base, la même application mobile peut être adaptée et déployée sur n'importe quel appareil mobile en recomposant à chaque fois les entités appropriées choisies en fonction des nouvelles informations contextuelles du dispositif visé. De plus, la possibilité de déployer la même application mobile sur un éventail de dispositifs mobiles hétérogènes tels que les Smartphones, les tablettes, etc. La Figure suivante (cf. Figure 3.2) décrit le contexte global du processus proposé. Elle indique et regroupe toutes les notions clés adressées par ce processus.

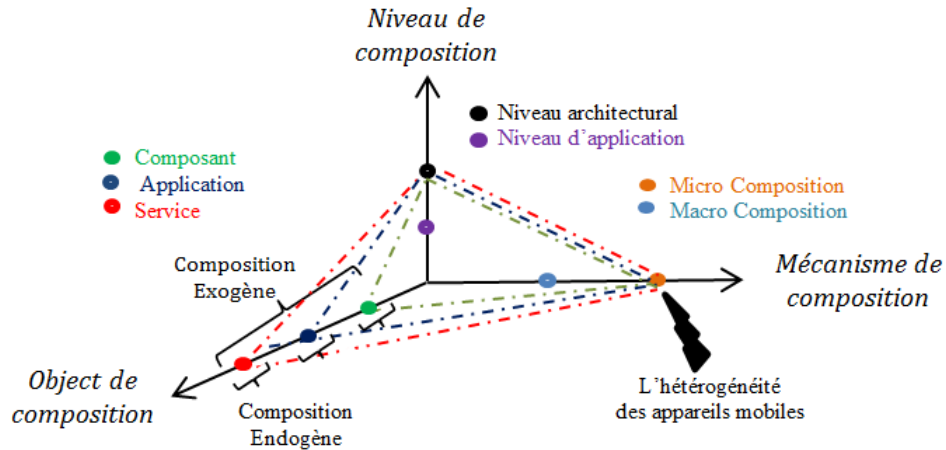


FIGURE 3.2 – Contexte global du processus proposé (CAMAP)

Le mécanisme de composition proposé permet aux architectes/développeurs de construire des applications mobiles hétérogènes ou homogènes. Ces applications désirées peuvent être construites via une *composition endogène* par l'utilisation d'un seul type de l'objet de composition ou une *composition exogène* en permettant la composition de différents types des objets à composer. Les caractéristiques de l'application composite dépendent fortement des entités logicielles qui l'intègrent. Ce mécanisme de composition est dédié aux environnements mobiles (*Micro composition* cf. Chapitre 2 : section 2) et conçu de point de vue architectural. En outre, les appareils mobiles sur lesquels les applications souhaitées seront déployées sont caractérisés par différentes configurations matérielles et logicielles. Par conséquent, les applications mobiles générées doivent être adaptables à leurs environnements d'exécution. Pour cette raison, notre mécanisme de composition prend en considération les informations

contextuelles des dispositifs mobiles ainsi que l'état d'exécution des ressources disponibles sur ces dispositifs lors de la composition conceptuelle de l'application souhaitée.

3 CAMAP : Étapes et déroulement

Le mécanisme de composition proposé est conçu pour permettre la construction des applications mobiles par la réutilisation de toute sorte d'entités logicielles afin de tirer profit de ses services fournis indépendamment de ses détails d'implémentation. Il vise également à respecter les ressources limitées offertes par le dispositif mobile sur lequel l'application mobile composite sera déployée afin d'obtenir une application adaptative.

Afin de pouvoir réaliser ces objectifs nous avons adopté une démarche de méta-modélisation. Aujourd'hui, cette démarche est favorisée et prédomine le monde de développement. A titre d'exemple, elle est adoptée même pour les systèmes avioniques (Balasubramanian et al., 2006) ce qui prouve l'importance et l'efficacité de l'ingénierie dirigée par les modèles. Ce type d'ingénierie reflète un processus de développement qui repose sur un ensemble de transformations qui sont habituellement exécutées séquentiellement à partir d'un niveau abstrait jusqu'à la production du code ou d'une spécification détaillée de l'implémentation (Cariou et al., 2004; Cariou et al., 2010). Donc, il permet un développement simple et facile d'applications mobiles pour des plateformes spécifiques en utilisant les modèles. En conséquence, nous avons proposé un processus de composition afin d'avoir des applications mobiles adaptatives comblant les besoins des utilisateurs. Ce processus proposé se déroule comme suit :

- 1) Élaborer un modèle fonctionnel en déterminant toutes les fonctionnalités désirées et par conséquent les différentes relations de composition entre ces fonctionnalités.
- 2) Raffiner le modèle fonctionnel obtenu en attachant chaque fonctionnalité par les différentes informations contextuelles de la meilleure entité concrète appropriée qui sera utilisée pour l'implémenter (i.e. adaptable au contexte de d'appareil mobile).
- 3) Générer le modèle architectural de l'application mobile composite, en remplaçant chaque fonctionnalité par son entité logicielle concrète sélectionnée (ex. un service, un composant, etc.) y compris les adaptateurs nécessaires afin d'éliminer les problèmes d'hétérogénéité rencontrés.
- 4) Raffiner le modèle architectural généré en spécifiant les tâches des médiateurs intégrés afin d'obtenir une architecture qui décrit la spécification détaillée pour l'implémentation de l'application composite.

Le passage entre ces différents modèles est effectué via des mécanismes de transformation. La Figure suivante (cf. Figure 3.3) décrit le fonctionnement global du processus proposé en commençant par la spécification des exigences dans un modèle fonctionnel jusqu'à l'obtention d'une architecture qui décrit la composition détaillée de l'application mobile désirée qui peut être hétérogène et adaptable à son environnement d'exécution.

Un mécanisme de transformation représente le passage d'un formalisme à un autre plus spécifique. Notre processus de composition manipule quatre formalismes : le *modèle fonctionnel*, le *modèle fonctionnel raffiné*, le *modèle architectural* et le *modèle architectural détaillé*. Il traite la transformation de type *Modèle vers Modèle* où : (a) le passage du modèle fonctionnel à un autre raffiné ; le passage du modèle architectural au modèle architectural détaillé consiste à effectuer une opération de raffinement. Cette dernière correspond à une transformation endogène qui porte sur le même méta-modèle, i.e. modèles source et cible conformes au

même méta-modèle. (b) le passage du modèle fonctionnel raffiné au modèle architectural consiste à une opération de projection correspond à une transformation exogène qui porte sur des méta-modèles source et cible différents, i.e. modèles source et cible conformes aux méta-modèles différents.

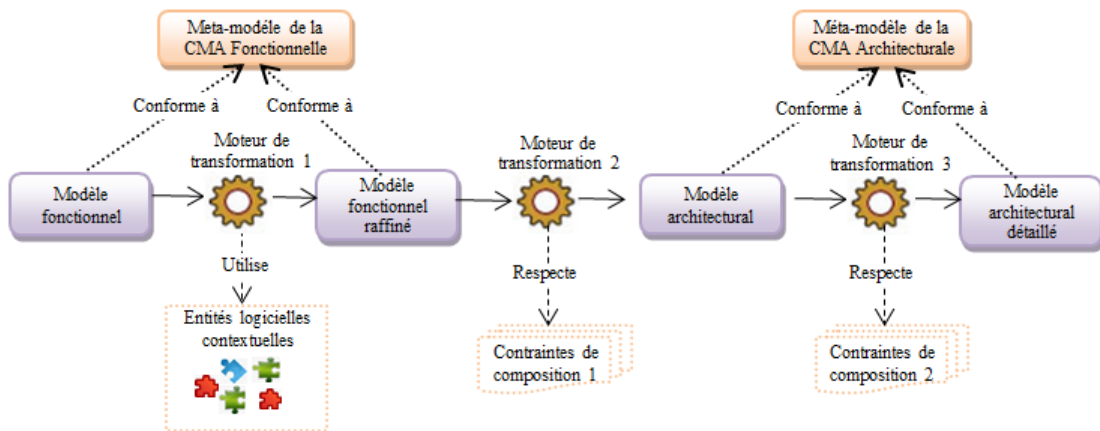


FIGURE 3.3 – Le fonctionnement global du processus proposé (CAMAP)

Le modèle fonctionnel reflète l'identification des besoins et la spécification des dépendances entre ces besoins pour construire l'application mobile désirée indépendamment de toute plateforme d'exécution. Il sert donc à représenter l'application mobile fonctionnelle qui est indépendante de tout environnement d'exécution, i.e. le contexte de l'appareil mobile cible. En revanche, le modèle architectural vise à représenter et à attacher l'application composite à un contexte d'exécution spécifique.

Les deux *moteurs de transformation 1* et *3* (cf. Figure 3.3) reposent sur le même principe qui est la restructuration des modèles. La tâche de restructuration consiste à réorganiser les éléments architecturaux d'un tel modèle et donc effectuer des modifications au niveau de leurs modèles sources. Cette modification reflète un ensemble d'opérations à effectuer (ex. ajout, modification ou suppression d'une partie de ses éléments) afin d'en améliorer la structure du modèle source et de le raffiner ce qui représente une transformation sur place, i.e. mise à jour. Le *moteur de transformation 2* sert à associer au modèle source un et un seul modèle cible. Ce type de transformation sert à remplacer chaque élément architectural représenté dans le modèle fonctionnel raffiné par son équivalence dans le modèle cible qui est le modèle architectural de l'application composite (cf. Figure 3.4).

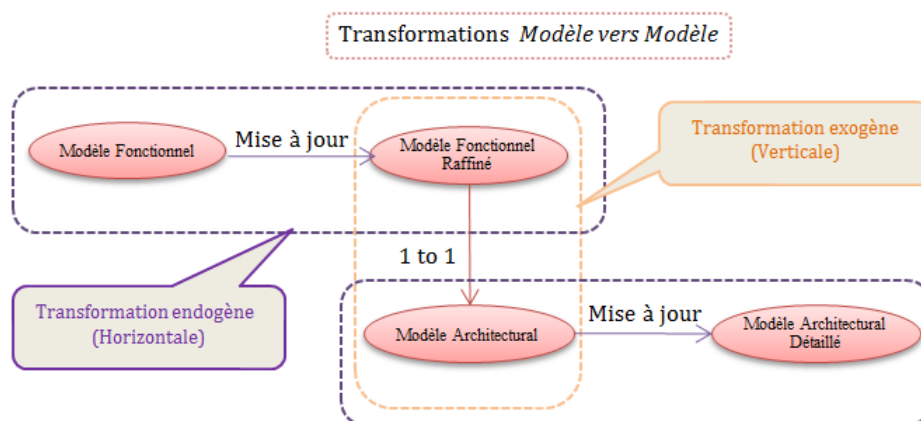


FIGURE 3.4 – Positionnement de notre processus de composition dans le monde de transformations

Les sections qui suivent soulignent et décrivent les points de notre contribution ainsi que le mécanisme opératoire du processus de composition dédié pour aider à développer des applications mobiles adaptatives.

4 La description du contexte de composition

4.1 La nécessité de décrire le contexte

Le développement d'applications pour des environnements mobiles est soumis à plusieurs conditions d'exécution. Dans ce contexte, les applications mobiles sont restreintes par les ressources limitées offertes par les appareils mobiles sur lesquels elles seront déployées comme : une mémoire limitée, un environnement mobile alimenté par batterie a une capacité limitée et de la disponibilité de certains services (ex. un accès Wi-Fi, un service GPS, un appareil photo avec une fonction d'*autofocus*, etc.) (Zhang et al., 2011).

Les ressources limitées et l'hétérogénéité des appareils mobiles montrent que la portabilité des exigences joue un rôle important dans le domaine de développement des applications mobiles. Autrement dit, les applications mobiles dépendent fortement des caractéristiques de l'environnement d'exécution. De ce fait, pour assurer le déploiement correct et le bon fonctionnement de l'application mobile composite, il est nécessaire de faire en sorte que leurs entités constitutives soient adaptables au contexte actuel de l'appareil mobile cible.

Cependant, une telle fonctionnalité souhaitée définie à un niveau conceptuel peut être implémentée avec plusieurs entités logicielles équivalentes mais capable de fonctionner dans des conditions différentes. Pour cela, afin de composer une application mobile adaptative nous devons toujours prendre précisément en considération les différentes caractéristiques des appareils mobiles lors du choix des entités logicielles. De ce fait, pour l'adaptation des applications mobiles au contexte il est important de la prendre en considération au niveau conceptuel, i.e. le contexte d'exécution doit être fourni lors de la conception de l'application mobile. Donc, l'adaptation contextuelle d'une application repose sur le fait d'observer le contexte de composition. Par conséquent, le processus proposé a la possibilité d'observer l'environnement de composition, i.e. d'identifier et de récupérer les informations pertinentes sur les objets de composition et l'environnement d'exécution. L'originalité de notre processus repose entre autres sur le fait que nous proposons un moyen compatible à nos objectifs pour décrire le contexte de composition. Cependant, nous avons proposé une définition du contexte cohérente par rapport à nos objectifs et une formalisation de la définition du contexte de composition. Ce dernier reflète :

- La spécification des informations contextuelles des entités logicielles constituantes de l'application, i.e. les entités logicielles correspondantes qui peuvent être utilisées pour implémenter les différentes fonctionnalités requises, y compris leurs rôles et toutes les conditions nécessaires pour leur exécution.
- Toutes les caractéristiques associées à l'appareil mobile sur lequel l'application mobile souhaitée sera déployée ainsi que l'état d'exécution des ressources disponibles.

Nous avons choisi d'utiliser les *ontologies* comme langage de description afin de modéliser le contexte de composition. Ceci fournit une description sémantique qui permet de déterminer le comportement des entités logicielles et les caractéristiques de l'appareil mobile. En outre, dû au fait qu'une entité logicielle peut être implémentée soit avec un service soit un composant, etc. et les appareils mobiles présentent des configurations hétérogènes. L'ontologie fournit une description standard de ces entités et ces appareils mobiles hétérogènes. Par conséquent, elle

prend en charge l'interopérabilité entre eux comme un vocabulaire commun. Les sous-sections qui suivent ont pour objectif de modéliser le contexte de composition d'une application mobile.

4.2 Modélisation du contexte de l'appareil mobile cible

Dans le domaine des applications mobiles, une composition réussie dépend fortement de son environnement d'application. De ce fait, pour assurer le déploiement correct et le bon fonctionnement des applications mobiles composites il faut prendre en compte les différentes informations contextuelles de l'appareil mobile lors de la réalisation de la tâche de composition. Donc, il est nécessaire d'identifier les différentes caractéristiques de l'environnement de déploiement. A cet effet, nous proposons une description basée sur les ontologies pour modéliser le contexte actuel de l'environnement du déploiement tel qu'illustrée sur la Figure 3.5.

Sachant que nous pouvons trouver différents dispositifs avec diverses capacités et des caractéristiques hétérogènes. En outre, l'état d'exécution des ressources disponibles sur ces appareils mobiles est temporel et peut être modifié à tout moment. Par exemple, le niveau de la batterie du dispositif mobile devient faible, l'accès Wi-Fi est désactivé, etc. Par conséquent, le contexte courant du dispositif mobile est représenté avec ses caractéristiques matérielles et logicielles ainsi que l'état actuel des ressources disponibles. Dans notre travail de recherche, nous proposons de modéliser les informations contextuelles d'un appareil mobile selon trois catégories :

- **Le contexte de déploiement (A1)** : représente les caractéristiques matérielles de l'appareil mobile.
- **La plateforme d'exécution (A2)** : représente le système d'exploitation installé sur l'appareil mobile cible.
- **Le contexte d'exécution (A3)** : représente l'état actuel des ressources disponibles sur l'appareil mobile de l'utilisateur.

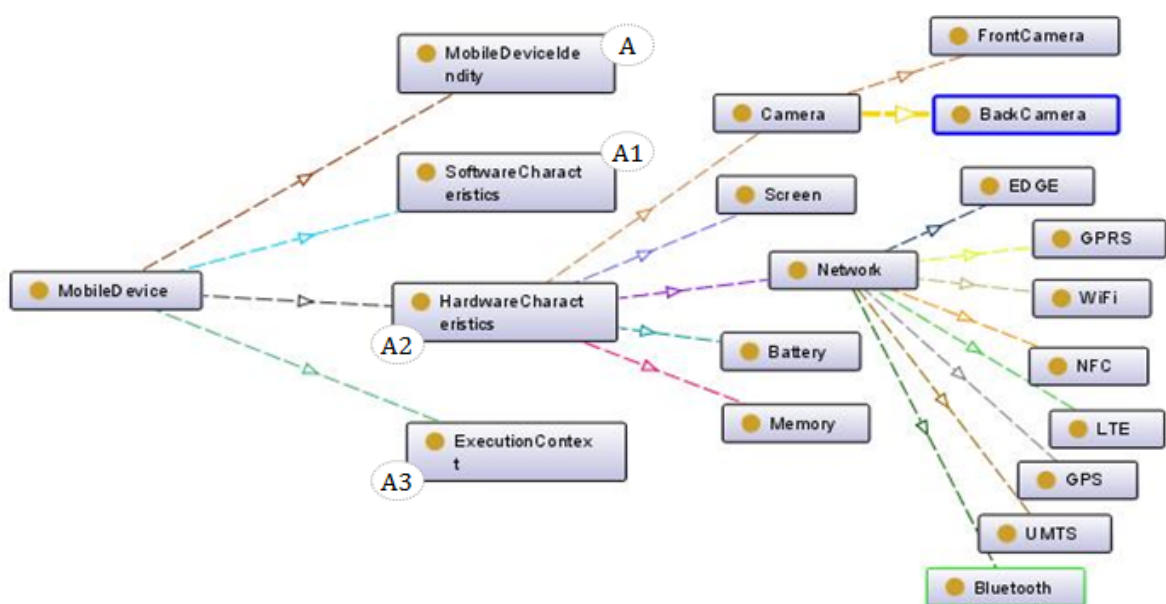


FIGURE 3.5 – L'ontologie du contexte de l'appareil mobile

Nous avons associé chaque appareil mobile par une identité ((A) : *MobileDevice-Identity*) qui reflète le nom du dispositif et son numéro de référence. La disponibilité de certains équipements (ex. une connexion Wi-Fi, GPS et d'une caméra) et également les valeurs de certaines fonctions de l'appareil (ex. la taille de l'écran, le type de la plateforme installée, la capacité de stockage restante, etc.) sont les critères sur lesquels nous sommes basés pour assurer le bon déploiement de l'application mobile désirée ((A1) : *SoftwareCharacteritics*, (A2) : *HardwareCharacteristics*). Ainsi, la modélisation de l'état actuel des ressources disponibles (i.e. l'état de Wi-Fi, l'état de GPS et le niveau actuel de la batterie) est un aspect crucial pour atteindre notre objectif qui est d'assurer le bon fonctionnement de l'application mobile composite désirée ((A3) : *ExecutionContext*).

Supposons que nous ayons une entité logicielle (constituante d'une telle application mobile) qui a besoin d'un accès Wi-Fi pour qu'elle puisse effectuer sa tâche et nous supposons qu'elle consomme 3% d'énergie pendant sa durée d'exécution moyenne, i.e. l'exécution de cette entité logicielle implique que la batterie de l'appareil mobile perdra 3% de sa capacité. De plus, nous supposons que l'appareil mobile cible possède le service Wi-Fi. Néanmoins, son état actuel est désactivé et le niveau actuel de la batterie est de 2%. Dans ce cas, cette entité logicielle ne fonctionnera pas correctement tant que l'accès Wi-Fi est désactivé et le niveau de la batterie est faible. Un autre exemple, si on trouve une entité logicielle constituante qui a besoin de 120Mb de mémoire et la capacité de stockage disponible actuellement sur l'appareil mobile est plus faible que celle qui est nécessaire, cette entité logicielle ne sera pas déployée avec succès.

Cependant, nous avons traité le contexte de l'appareil mobile à travers deux dimensions : un contexte statique qui est représenté par les caractéristiques matérielles et logicielles du dispositif mobile et un autre dynamique qui est spécifié via l'état courant des ressources existantes sur ce dispositif. Par conséquent, le processus de composition proposé a la capacité de connaître le contexte d'usage pour une application donnée afin de garantir une composition réussie dédiée à un contexte spécifique.

4.3 Modélisation du contexte des objets de composition

Plusieurs entités logicielles peuvent être utilisées pour implémenter une telle fonctionnalité désirée. Par conséquent, ces entités logicielles sont fonctionnellement équivalentes ((B) : *FunctionalAspects*) mais elles peuvent varier dans plusieurs aspects non fonctionnels ((C) : *NonFunctionalAspects*), i.e. des conditions d'exécution différentes. Les caractéristiques fondamentales liées à l'entité logicielle sont représentées par : son type d'implémentation (service, composant, etc.) et son rôle (description de la tâche à effectuer).

Notre processus de composition repose sur l'utilisation des entités logicielles qui sont adaptables au contexte courant de l'appareil mobile cible pour composer et obtenir une application adaptative. Afin de choisir les entités logicielles contextuelles nous proposons d'associer chaque entité avec un profil d'exécution spécifique. Ce profil d'exécution contient tous les aspects non-fonctionnels qui représentent les conditions nécessaires pour son exécution (ex. la plateforme nécessaire (C1), les connectivités réseaux requis avec leurs caractéristiques (C2), les caractéristiques requises pour un appareil photo (C3), consommation d'énergie (C4), la taille de l'écran nécessaire (C5), la capacité nécessaire pour la déployer (C6), etc.). De cette façon, les profils d'exécution contiennent divers paramètres associés par des valeurs et qui reflètent les différentes caractéristiques requises pour déployer et exécuter cette entité logicielle. Ces métriques sont les critères que nous proposons pour assurer le déploiement approprié et le bon fonctionnement de l'entité logicielle.

Par conséquent, la description de l'entité logicielle concerne non seulement les caractéristiques fonctionnelles mais elle doit aussi être assez riche pour fournir son contexte d'exécution dont l'objectif est d'effectuer une composition efficace. Le modèle d'ontologies

présenté dans la Figure 3.6 décrit la représentation fonctionnelle ainsi toutes les conditions nécessaires pour l'exécution d'une entité logicielle.

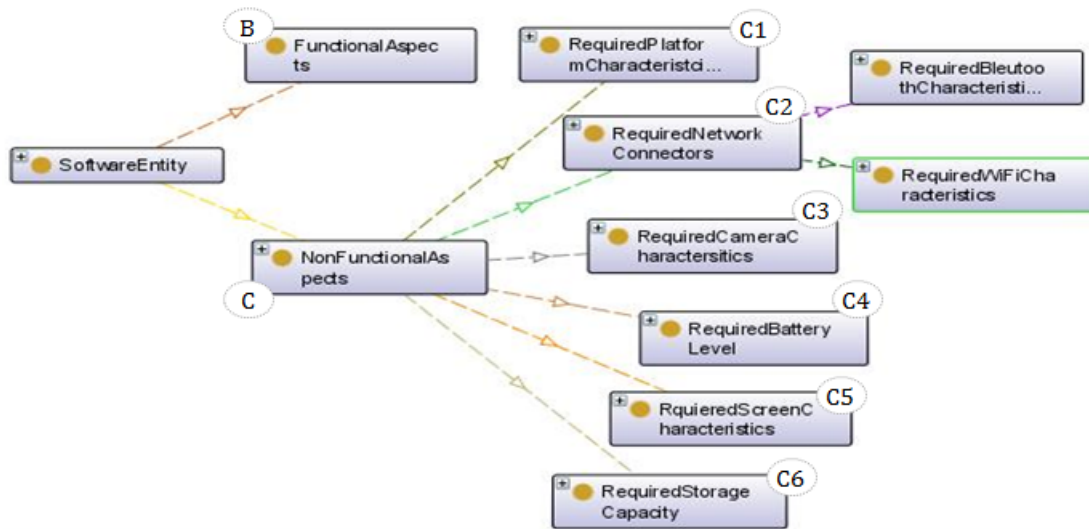


FIGURE 3.6 – L'ontologie du contexte de l'entité logicielle

A titre d'exemple, nous supposons que nous avons une fonctionnalité requises *Fun2 : Read Barcode* qui vise à lire le code-barres d'un produit. Cette fonctionnalité peut être réalisée avec trois entités logicielles différentes (cf. Figure 3.7) :

- SE_1 : Service (reconnaissance du code-barres à distance).
- SE_2 : Composant (reconnaissance locale du code-barres).
- SE_3 : Service (<http://searchupc.com/>).

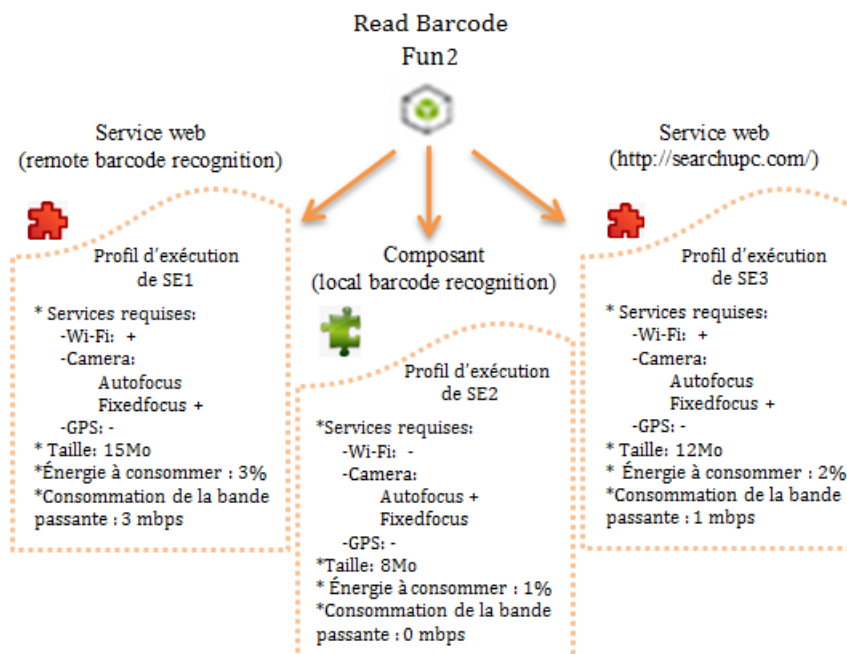


FIGURE 3.7 – Profil d'exécution des entités logicielles

Chacune de ces entités logicielles possède un ensemble de contraintes d'exécution, i.e. liste des conditions requises pour exécuter une entité logicielle. Par exemple, l'entité logicielle SE_1 a besoin d'une caméra en mode *fixedfocus*, un accès Wi-Fi et 3% d'énergie à consommer pour qu'elle puisse effectuer sa tâche et donc fonctionner correctement. L'image du produit sera prise par un appareil équipé d'une caméra avec la fonction *fixedfocus* ce qui empêchera la reconnaissance locale du code-barres ce qui nécessite l'invocation d'un service à distance pour traiter l'image acquise. En outre, il a besoin de 15Mo d'espace mémoire pour peut être déployée correctement sur l'appareil mobile, etc. En revanche, l'entité logicielle SE_2 a besoin d'une caméra avec une fonction d'*autofocus* et 1% de l'énergie de la batterie pour pouvoir fonctionner correctement. En outre, il a besoin de 8Mo de la capacité du stockage pour être déployée correctement sur l'appareil mobile, etc. Ici, nous ne présentons pas les valeurs réelles mais seulement les différences relatives entre les aspects non fonctionnels. Toutes les conditions nécessaires sont exprimées dans un profil d'exécution comme illustré dans la Figure 3.7 où les métriques qui portent le signe + ou - reflètent le besoin ou le non besoin du service/ressource pour l'exécution de l'entité et les autres métriques portent des valeurs changeables.

5 Modélisation de l'application mobile composite

Le principe du processus de composition proposé dans ce manuscrit repose sur la représentation de l'application mobile composite suivant plusieurs niveaux d'abstraction en commençant par la modélisation des besoins jusqu'à l'obtention d'une description architecturale détaillée. Dans les sous-sections suivantes nous présentons les méta-modèles proposés pour décrire l'application mobile composite dans ces différents niveaux d'abstraction.

5.1 La représentation fonctionnelle

Le premier formalisme proposé sert à définir l'application mobile désirée abstraitement et indépendamment de n'importe quel environnement d'exécution. Cependant, il est nécessaire de se focaliser dans un premier temps sur l'identification des différentes fonctionnalités qui sont requises pour accomplir l'objectif de la future application mobile et fournir les dépendances entre les fonctionnalités identifiées. De plus, ce formalisme permet d'effectuer la tâche de collaboration tout en spécifiant les différents ordres d'invocations ainsi les données échangées.

Le méta-modèle présenté dans la Figure 3.8 permet au concepteur de l'application mobile d'élaborer le modèle fonctionnel (les classes en blanc). Ce méta-modèle proposé est nommé CMA-FD qui est l'acronyme de Composite Mobile Application Functional Description ; celui-ci représente une description fonctionnelle de haut niveau de l'application mobile désirée (A). Il permet la définition de fonctionnalités primitives (A1) ou composites (A2) tout en identifiant les différentes relations de composition nécessaires (A3). Une fonctionnalité primitive se réfère à une action abstraite ; elle a un identifiant (A4), un nom (A5), une description (A6). Une fonctionnalité composite est un ensemble de fonctionnalités primitives (A7). Les relations de composition indiquent le flux de travail (i.e. *Workflow*) et le flux de données (i.e. *Dataflow*) entre les fonctionnalités identifiées. Le flux de travail représentent les ordres d'invocation des fonctionnalités en établissant des liens de précédence (A8) alors que le flux des données exprime les données échangées, i.e. les entrées et sorties, entre elles en établissant des liens d'utilisation (A9). Le modèle fonctionnel de la CMA vise à représenter l'application mobile souhaitée indépendamment de toute technologie ou du domaine d'application.

Un objectif important de notre processus de composition est de composer des applications mobiles qui sont adaptables à son environnement d'exécution. Afin d'élaborer l'application

mobile souhaitée vers un appareil mobile spécifique nous avons besoin d'attacher chaque fonctionnalité identifiée par sa propre entité logicielle concrète qui sera utilisée pour l'implémenter et un ensemble des données d'entrées et de sorties (B1). De ce fait, le formalisme proposé est conçu pour être raffinée afin d'associer les fonctionnalités déterminées par les informations contextuelles des entités concrètes qui sont choisies pour les implémenter. Ces informations contextuelles reflètent les contraintes d'exécution (B2) définis dans le profil d'exécution comme nous avons indiqué précédemment. Par conséquent, nous aurons un modèle fonctionnel raffiné qui est conforme à son tour au méta-modèle CMA-FD (cf. Figure 3.8, les classes grisées). Nous associons la capacité à raffiner le modèle fonctionnel et donc obtenir un modèle fonctionnel raffiné au *moteur de transformation 1* (cf. Figure 3.3).

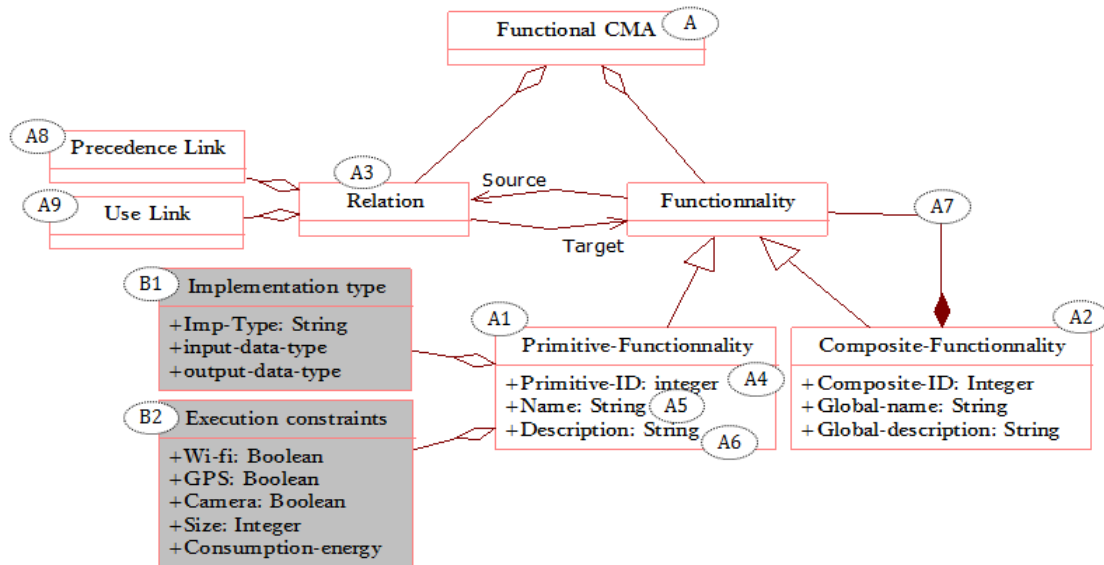


FIGURE 3.8 – Le méta-modèle CMA-FD

5.2 La description architecturale

La tâche de composition présentée par le processus proposé est dirigée par les exigences des utilisateurs et le contexte de l'appareil mobile cible. Les entités logicielles concrètes choisies en fonction des besoins des utilisateurs et du contexte pour implémenter les fonctionnalités désirées peuvent être hétérogènes que ce soit au niveau de leur type ou de leur nature. A cet effet, nous avons proposé un autre méta-modèle nommé H^2 CMA-AD (Heterogeneous or Homogeneous Composite Mobile Application - Architectural Description) pour décrire des applications mobiles composites hétérogènes aussi bien qu'homogènes au niveau architectural (cf. Figure 3.9, les classes en blanc). L'objectif de ce méta-modèle est de pouvoir représenter n'importe quelle application mobile quelque soient les détails d'implémentation de ses entités logicielles constitutives. Donc, ce méta-modèle décrit un formalisme qui nous permet d'élaborer une composition hétérogène. Ce formalisme sert à représenter l'application composite via les entités logicielles concrètes choisies pour implémenter les fonctionnalités requises. Un grand avantage de notre processus de composition est de ne pas limiter le choix technologique de ces entités concrètes. Cependant, le formalisme proposé vise à spécifier les points d'hétérogénéité dans le cas de coordinations hétérogènes. On outre, il nous donne la possibilité d'associer ces relations de composition avec des médiateurs afin de remédier les issues d'hétérogénéité.

Une entité logicielle est une fonction qui prend en entrée un ensemble de paramètres nécessaires pour qu'elle fonctionne et renvoie en sortie le résultat souhaité. Cette fonction

doit être exécutée selon un ensemble de conditions que nous avons regroupées dans un profil d'exécution. Par conséquent, nous définissons une entité logicielle comme un quadruplé : La fonction à exécuter, les données d'entrée et de sortie, le profil d'exécution. Suivant le méta-modèle illustré dans la Figure 3.9 une application mobile composite (C) est constituée d'un ensemble d'entités logicielles (C1) liées entre-elles via des connecteurs (C2) où :

- Chaque entité logicielle peut être un composant (C3), un service (C4) ou une entité logicielle composite (C5) (service composite, composant composite, etc.)
- Chaque entité logicielle possède des données d'entrée (C6) et de sortie (C7) et également un profil d'exécution (C8) contenant toutes les conditions nécessaires pour son exécution.
- L'échange des données entre les entités logicielles reliées est effectué via des ports fournis (C9) et des ports requis (C10) pour les entités de type composant et via des services fournis (C11) et des services requis (C12) pour les entités de type service.
- Les différentes relations entre les entités constitutives sont des liens de précedence (C13) et des liens d'utilisation (C14) où ces connecteurs seront attachés avec des médiateurs endogènes (C15) ou exogènes (C16) dans le cas d'une coordination hétérogène.
- L'application mobile globale est dédiée à être exécutée dans un environnement mobile spécifique (C17) avec un contexte d'exécution (C18) bien déterminé.

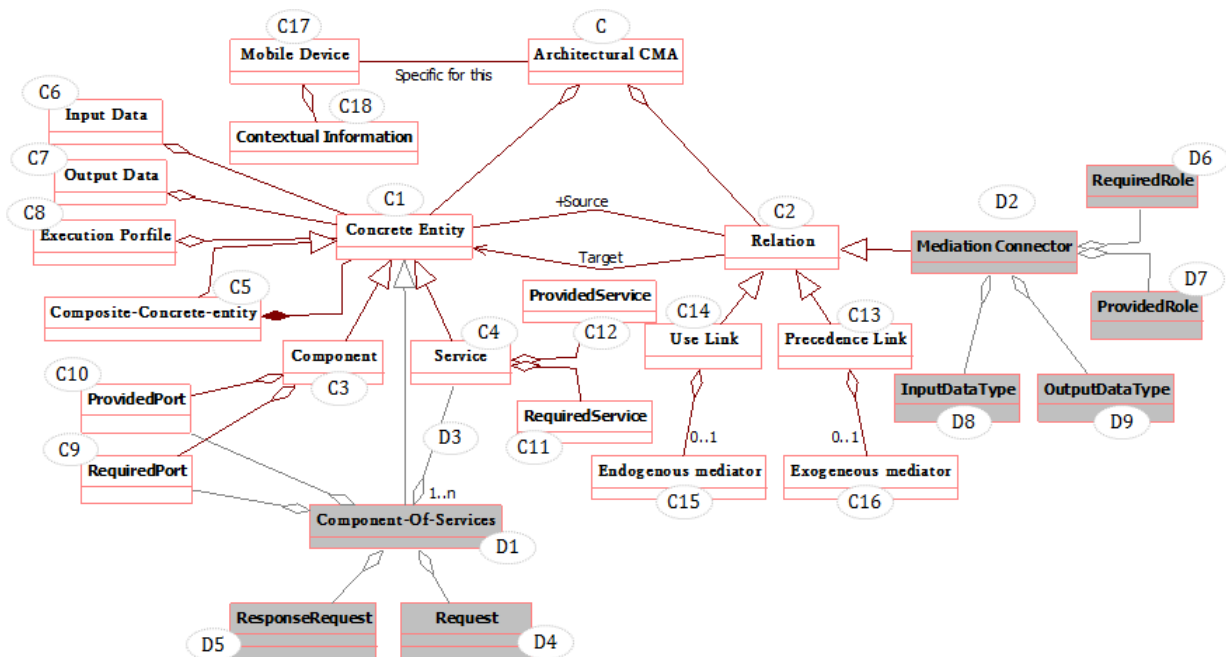


FIGURE 3.9 – Le méta-modèle CMA-FD

Cette description architecturale vise à représenter la CMA souhaitée avec des services ou des composants séparément ou avec des entités logicielles hétérogènes pour cause du contexte de l'appareil mobile et des exigences souhaités. Par conséquent, elle permet la création d'applications mobiles adaptatives comme une *composition endogène* ou *exogène*. Via ces deux types de composition nous pouvons avoir deux types d'applications : (1) *Une application homogène* constituée d'un ensemble d'entités du même type manipulant des données du type homogène. (2) *Une application hétérogène* reflétant un ensemble d'entités de différents types reliées entre-elles via des médiateurs et/ou les données échangées entre-elles nécessitent des

transformations pour qu'elles soient compréhensibles. Le défi de l'hétérogénéité présente deux types de problèmes lors de la tâche de composition :

- (1) **L'hétérogénéité de la nature des entités** : si les entités reliées ne peuvent pas communiquer directement dû au fait que les données échangées entre elles ne sont pas compréhensibles, nous disons que ces entités reliées n'ont pas la même nature. A titre d'exemple, la fonctionnalité *Aquire Photo* fournit une image de type jpg tandis que la fonctionnalité *Read Barcode* a besoin d'une image de type WebP pour qu'elle puisse fonctionner.
- (2) **L'hétérogénéité du type des entités** : ce type d'hétérogénéité représente la coordination de deux types différents des entités logicielles. Dans ce cas nous disons que les entités reliées ne sont pas du même type. A titre d'exemple, une entité de type composant reliée avec une autre de type service.

Le processus de composition CAMAP vise à adresser et traiter ces problèmes d'hétérogénéité en proposant deux types de médiateurs :

- (1) **Les médiateurs endogènes** : ce type de médiateurs est destiné à éliminer l'hétérogénéité entre deux entités logicielles de différentes natures qui ne peuvent pas communiquer directement.
- (2) **Les médiateurs exogènes** : ce type de médiateurs vise à surmonter l'hétérogénéité entre deux entités logicielles qui n'ont pas le même type d'implémentation (Djeddar et al., 2014).

De ce fait, le formalisme proposé pour définir le modèle architectural de la CMA est dédié pour être raffiné afin d'intégrer le fonctionnement des médiateurs proposés et donc obtenir un modèle architectural détaillé. Ce dernier est conforme à son tour au méta-modèle H^2 CMA-AD proposé (cf. Figure 3.9, les classes grisées) où : (a) Le *médiateur exogène* indiqué dans le modèle architectural sera remplacé par une entité de type composant nommé *Component-of-services* (D1) qui encapsule les entités de type services afin d'éliminer l'hétérogénéité entre les entités de différents types en construisant des interfaces de communications communes ; (b) Le *médiateur endogène* devient un connecteur de médiation (D2) qui est dédiés pour assurer la compatibilité des données échangées.

Par conséquent, le mécanisme de composition proposé n'est pas seulement destiné à la détection des points d'hétérogénéité mais aussi d'intégrer des médiateurs et de décrire leur fonctionnement pour remédier aux problèmes d'hétérogénéité posés lors de la composition.

■ Spécification des médiateurs proposés

Après avoir défini le modèle architectural pour l'application mobile composite souhaitée, notre processus vise à spécifier plus précisément les rôles des médiateurs intégrés.

D'une part, les *médiateurs exogènes* visent à assurer la communication entre deux entités logicielles de différents types. Vu que les entités logicielles ne peuvent pas communiquer en raison de leurs types d'implémentation hétérogènes, les *médiateurs exogènes* visent à encapsuler ces entités reliées de telle sorte qu'elles peuvent interagir entre elles. Ils construisent des interfaces bien-formées communes pour ces entités afin de tirer profit de ses services mais juste en manipulant les entrées et les sorties nécessaires indépendamment de leurs détails d'implémentation. Plus précisément, dans le cas d'une coordination exogène entre un composant et un service, le service sera encapsulé (D3) dans la nouvelle entité *Component-of-services*. Ce dernier peut encapsuler un ou plusieurs services collaborés (cf.

Figure 3.10) où il joue le rôle du moteur défini dans l'orchestration de services. Il vise à déclencher l'exécution de l'entité de type service qu'il englobe en fournissant dans un premier temps les données requises obtenues via son port requis par le biais d'une requête déclenchée (D4) et de diffuser le résultat obtenu via son port fournis en déclenchant une autre requête de réponse (D5). Cela assure une communication compatible entre les entités sources et cibles via des interfaces communes.

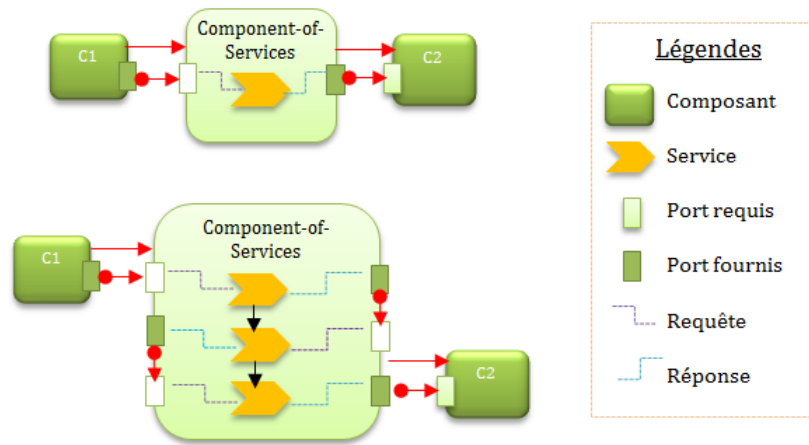


FIGURE 3.10 – Spécification d'un médiateur exogène

D'autre part, les médiateurs endogènes représentent les services de médiation qui seront sélectionnés pour assurer la transformation des données échangées qui sont hétérogènes. Dans ce cas-là, les liens qui sont attachés par ce type de médiateur seront des connecteurs présentant des interactions complexes. Ces connecteurs ont pour objectif de convertir les données échangées en appelant les services de transformation appropriés tout en utilisant ses rôles (D6)(D7) pour recevoir et diffuser ces données.

Le formalisme proposé représente ce connecteur comme une glu qui définit une fonction qui sert à chercher le service de médiation approprié (*search MD* : chercher un service de médiation) en se basant sur les types des données échangées (D8)(D9) dans la bibliothèque de services de médiation ; et une autre qui sert à appeler le service de transformation trouvé (*call MD* : appeler un service de médiation) afin d'assurer la compatibilité des données échangées (cf. Figure 3.11).

Dû au fait que les médiateurs exogènes vont éliminer l'hétérogénéité entre deux types différents des entités logicielles en encapsulant les services dans un composant spécifique, les liens d'utilisation vont alors ne relier que (*component/component*) ou (*component/Component-of-services*).

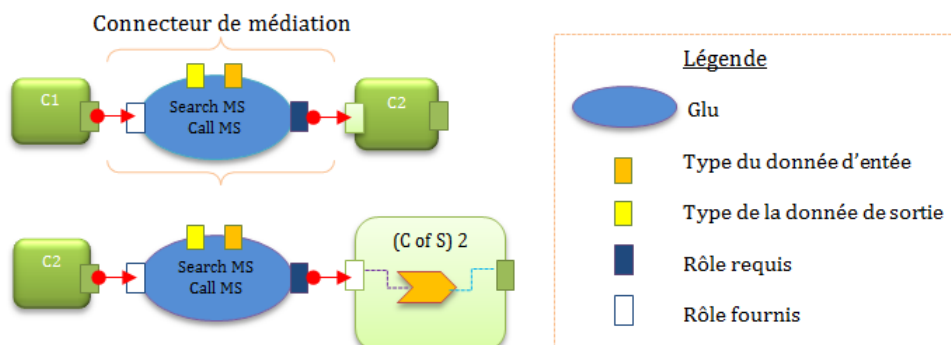


FIGURE 3.11 – Spécification d'un médiateur endogène

■ Les types de composition traités

Le processus de composition proposée dans ce manuscrit traite quatre types de composition (cf. Tableau 3.1) :

- (1) **Composition hétérogène exogène** : composition d'entités logicielles de différents types qui n'ont pas la même nature en attachant des *médiateurs exogènes* et *endogènes*.
- (2) **Composition homogène exogène** : composition d'entités logicielles de différents types qui ont la même nature en attachant des *médiateurs exogènes*.
- (3) **Composition hétérogène endogène** : composition d'entités logicielles du même type qui n'ont pas la même nature en attachant des *médiateurs endogènes*.
- (4) **Composition homogène endogène** : composition d'entités logicielles du même type qui ont la même nature par une relation simple et donc sans adaptation.

Tableau 3.1 – Les types de composition traités par le processus de composition CAMAP

Type de composition	Problèmes d'hétérogénéité	Médiateurs proposés
Composition hétérogène exogène	Entités de différentes natures/Entités de différents types	Médiateurs endogènes/Médiateur exogènes
Composition homogène exogène	Entités de différents types	Médiateur exogènes
Composition hétérogène endogène	Entités de différentes natures	Médiateurs endogènes
Composition homogène endogène	Aucun problème d'hétérogénéité	Aucun médiateur

6 Mécanisme opératoire du processus : passage entre modèles

Après avoir défini les différents formalismes que nous avons adoptés pour élaborer notre processus de composition, nous nous focalisons dans cette section sur les différents *moteurs de transformation* dédiés pour générer l'architecture détaillée de la CMA souhaitée à partir d'une description fonctionnelle donnée. Donc, nous allons présenter le mécanisme opératoire du processus de composition proposé.

6.1 Du modèle fonctionnel au modèle fonctionnel raffiné

Notre mécanisme de composition vise, comme une première étape, à définir le modèle fonctionnel de l'application mobile désirée suivant le méta-modèle CMA-FD proposé. La Figure 3.12 - Modèle (a) - représente le modèle fonctionnel de l'application mobile *ShopReview* présenté précédemment. Cette application a besoin d'exécuter un ensemble de fonctionnalités primitives qui nécessite l'exécution d'une tâche unique dans un ordre spécifique pour atteindre un objectif déterminé. Tout d'abord, nous prenons une image pour le produit que nous voulons acheter. En se basant sur l'image prise, l'application vise à extraire le code-barres du produit, i.e. la reconnaissance du code-barres indiqué sur la photo prise. Après avoir eu le code-barres du produit nous aurons besoin de le traduire en une chaîne de caractères afin d'avoir le nom

du produit. La prochaine fonctionnalité vise à recueillir le prix du produit de l'utilisateur. En utilisant le nom du produit et en se basant sur le prix donné par l'utilisateur, une autre fonctionnalité est nécessaire pour récupérer via l'Internet les prix les plus pratiques proposés sur les sites d'e-commerce. Ensuite, l'application sert à récupérer l'emplacement actuel de l'utilisateur afin d'utiliser la position obtenue pour récupérer par le biais d'Internet d'autres marchés à proximité qui offrent le produit à un prix concurrent. Enfin, l'utilisateur peut partager le prix du produit trouvé dans un magasin donné et donner également son avis sur *Twitter*. Le modèle fonctionnel montré sur la Figure 3.12 - Modèle (a) - représente l'ensemble de ces fonctionnalités requises et leurs dépendances y compris ses ordres d'exécution et tous les flux de données échangées entre elles.

Le *moteur de transformation 1* a besoin comme des informations sources d'un ensemble d'entités logicielles qui soient à la fois appropriées pour les besoins de l'utilisateur décrits dans le modèle fonctionnel et compatibles avec les informations contextuelles de l'appareil mobile cible. Toutes les informations contextuelles des entités logicielles concrètes correspondantes aux besoins sont décrit et stocker via le modèle d'ontologies proposé (cf. Sous-section 4.3). Les entités logicielles adaptatives sont le résultat de la tâche de sélection du processus de composition global proposé dans (Djeddar et al., 2015). Si nous prenons par exemple la fonctionnalité *Read Barcode*, nous pouvons trouver plusieurs entités logicielles concrètes qui peuvent être utilisées pour l'implémenter. Chacune d'elles a ses propres conditions d'exécution et une implémentation spécifique. La fonctionnalité *Read Barcode* reflète une entité logicielle abstraite qui vise à lire le code-barres écrit sur l'image du produit où celle-ci représente une donnée d'entrée de type d'image issu de la fonctionnalité *Acquire Photo*. Le code-barres obtenu représente une donnée de sortie de type entier. Cette fonctionnalité peut être implémentée soit par une entité logicielle de type service (SE_1 : reconnaissance de code-barres à distance) ou une entité de type composant (SE_2 : reconnaissance locale du code-barres). Ce dernier ne peut être exécuté correctement que sur des appareils mobiles équipés par une caméra avec le mode d'*autofocus* et ne fonctionne pas correctement sur d'autres appareils. Selon l'exemple de motivation présenté, SE_2 n'est pas adaptable au contexte du dispositif mobile car nous avons supposé que l'appareil mobile cible dispose une caméra avec le mode *fixedfocus*. Donc, dans ce cas-là SE_1 est l'entité logicielle qui sera choisie pour implémenter la fonctionnalité *Read Barcode*. Cette entité vise à appeler un service externe pour traiter l'image acquise afin d'extraire le code-barres du produit à partir d'une photo prise avec un appareil photo à *fixedfocus* mode (voir le modèle fonctionnel raffiné de l'application mobile *ShopReview* illustré dans la Figure 3.12 - Modèle (b)).

Par conséquent, le *moteur de transformation 1* correspond à une opération de raffinement fournissant un modèle fonctionnel d'une CMA adaptative qui a pour objectif d'être exécutée sur un dispositif mobile avec un contexte spécifique. Il sert à attacher chaque fonctionnalité représentée dans le modèle fonctionnel avec son entité logicielle appropriée contextuelle qui sera utilisée pour l'implémenter, i.e. attacher toute les informations contextuelles de l'entité concrète choisie. Chaque SE attachée possède un profil d'exécution qui contient toutes les conditions nécessaires pour son exécution. Comme illustré dans la Figure 3.12 - Modèle (b) -, la fonctionnalité *Read Barcode* est associée avec une entité logicielle de type service où ses caractéristiques requises pour être exécutée sont : caméra avec la fonction de *fixedfocus*, 16Mo de la capacité de stockage de l'appareil mobile et 5% d'énergie de la capacité de la batterie. Le *moteur de transformation 1* vise également à associer tous les liens d'utilisation avec les données d'entrées et de sorties des entités logicielles attachées. Par exemple, le lien d'utilisation entre la fonctionnalité *Acquire Photo* et la fonctionnalité *Read Barcode* est associé avec les données de l'image du produit tandis que l'autre qui est entre *Read Barcode* et *Get-Product Name* est associé par le résultat fourni qui est le code-barres du produit (i.e. *Product's Barcode*).

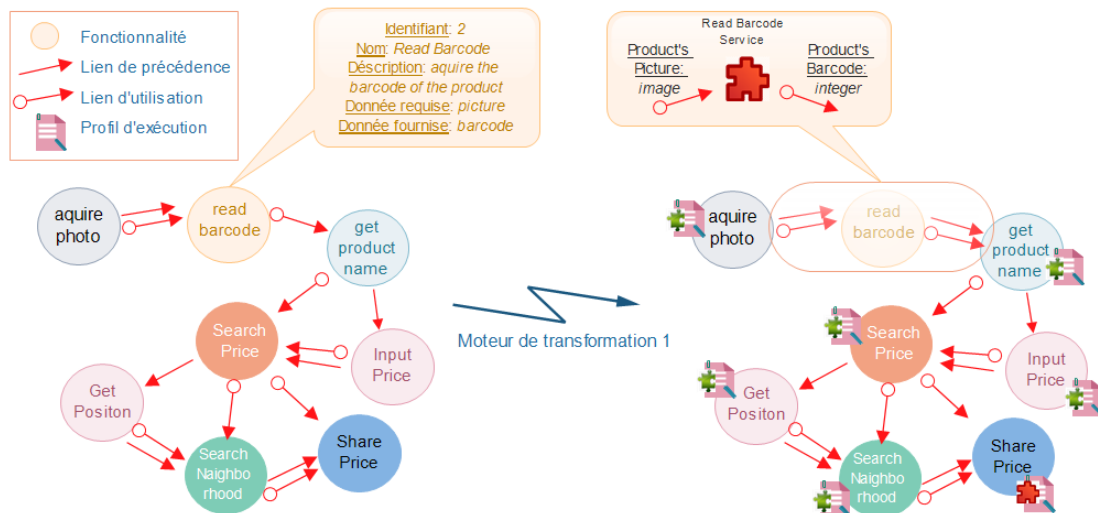


FIGURE 3.12 – Le moteur de transformation 1

6.2 Du modèle fonctionnel raffiné au modèle architectural

Via notre processus de composition, chaque CMA sera développée pour un environnement mobile spécifique. Donc, après avoir eu le modèle fonctionnel raffiné de la CMA nous nous concentrons sur la description architecturale de l'application mobile désirée. A ce stade, nous nous basons sur le choix technologique c'est-à-dire le type d'implémentation de toutes les fonctionnalités identifiées pour construire ce modèle architectural. Le rôle le plus important du *moteur de transformation 2* est de détecter et de préciser les différents points d'hétérogénéité qui peuvent survenir lors de la composition des entités contextuelles sélectionnées. Ce *moteur de transformation* vise à effectuer une opération de projection où chaque fonctionnalité définie dans le modèle fonctionnel raffiné de la CMA sera représenté avec son entité logicielle adaptative appropriée. En conséquence, nous aurons un modèle architectural de l'application mobile désirée qui est conforme au méta-modèle architectural proposé H^2 CMA-AD. Ce dernier représente une description de haut niveau des différentes entités logicielles concrètes adaptatives ainsi que les médiateurs nécessaires qui peuvent être attachés sur les interdépendances entre ces entités concrètes réutilisées afin d'éliminer l'hétérogénéité entre elles.

Cependant, le *moteur de transformation 2* doit respecter plusieurs contraintes de composition pour atteindre son objectif. Le respect des contraintes de composition correspond à une tâche de médiation qui a pour objectif de détecter les points d'hétérogénéité et à associer les relations de composition avec les différents médiateurs nécessaires.

Par conséquent, le *moteur de transformation 2* a le potentiel d'associer au cours de la tâche de composition les différentes relations de composition avec des *médiateurs exogènes* et/ou *endogènes* dans le cas de la présence d'une *combinaison hétérogène* et/ou *exogène*. En revanche, la composition *endogène/homogène* permet de composer des entités logicielles du même type et de la même nature par une relation simple et par conséquent sans aucune adaptation. Ce *moteur de transformation* est destiné à détecter et éliminer les problèmes d'hétérogénéité du point de vue architectural.

La Figure 3.13 illustre le modèle architectural de l'application mobile *ShopReview* y compris les adaptateurs nécessaires. Dans notre exemple, la fonctionnalité *Read Barcode* sera remplacée par son entité concrète choisie SE_1 qui est de type composant. Après avoir remplacé chaque fonctionnalité définie dans le modèle fonctionnel raffiné avec son entité concrète appropriée, le

moteur de transformation 2 gère les différents liens d'utilisation ainsi que les liens de précedence définis entre ces fonctionnalités afin d'identifier s'il est nécessaire d'associer ces relations de composition avec des médiateurs en terme de type d'implémentation de l'entité source et l'entité cible et également de la nature des données échangées entre elles, i.e. la détection et la gestion des points d'hétérogénéité. Les liens d'utilisation peuvent être associés avec des *médiateurs endogènes* car ils portent sur les données échangées tandis que les liens de précedence peuvent être attachés avec des *médiateurs exogènes* car ils portent sur les entités de logiciels elles-mêmes.

Si nous prenons par exemple le SE_2 et SE_3 qui sont de type hétérogène, dans ce cas, le lien de précedence qui relie ces deux entités doit être attaché avec un *médiateur exogène*. Ainsi, le SE_1 fournit un code-barres de type entier tandis que SE_3 a besoin d'une donnée d'entrée de type entier afin qu'il puisse mener à bien sa tâche. Par conséquent, les données échangées ne sont pas de nature hétérogène et ces deux entités liées ne nécessitent aucune transformation de données pour qu'elles puissent communiquer, i.e. communication directe sans aucune nécessité à la transformation des données échangées. Par conséquent, le lien d'utilisation n'a pas besoin d'être associé à un médiateur.

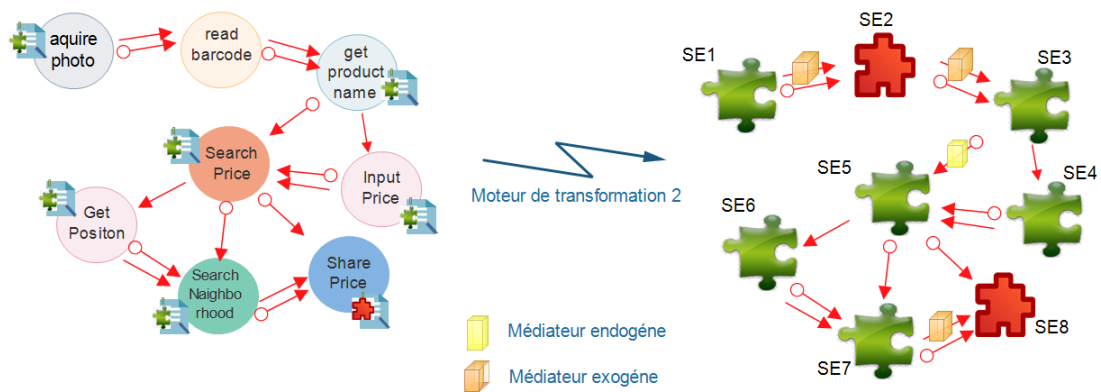


FIGURE 3.13 – Le moteur de transformation 2

6.3 Du modèle architectural au modèle architectural détaillé

Notre processus de composition vise comme une dernière étape à spécifier le rôle des médiateurs attachés. Le *moteur de transformation 3* utilise le modèle architectural obtenu pour générer une spécification détaillée de la CMA désirée. Ce *moteur de transformation* vise dans un premier temps à gérer les liens de précedences pour encapsuler à chaque fois où il détecte un *médiateur exogène* l'entité de type service dans une nouvelle entité nommée *Component-of-services* où :

- Le service requis par l'entité service sera une requête déclenchée par la nouvelle entité c'est-à-dire l'entité *Component-of-services* fait un appel à l'entité service tout en fournissant les données nécessaires pour son exécution qui sont obtenues via son port requis.
- Le lien d'utilisation qui relie l'entité source et l'entité service, i.e. entre un port fourni et un service requis, sera un lien d'utilisation entre l'entité source et l'entité encapsulant, i.e. entre un port fourni et un port requis.
- Le service fourni par l'entité service sera transmis vers le port fournis du composant encapsulant en déclenchant une requête de réponse.

- Le lien d'utilisation qui relie l'entité service et l'entité cible, i.e. entre un service fournis et un port requis, sera un lien d'utilisation entre l'entité encapsulant et l'entité cible, i.e. entre un port requis et un port fournis.

L'encapsulation peut dépasser un seul service dans le cas où le service encapsulé soit relié par d'autres services du côté cible. Dans ce cas-là, tous les services reliés seront encapsulés dans le même composant encapsulant jusqu'à ce que nous arrivions à une relation entre un service et un composant. Donc, l'encapsulation est le terme proposé pour refléter le regroupement d'un ou plusieurs entités services afin d'unifier les interfaces de communication entre les entités hétérogènes reliées. Les relations de composition entre les services encapsulés seront telles qu'illustrées sur la Figure 3.10. Le composant encapsulant fait à chaque fois un appel à un service selon l'ordre d'invocation défini via une requête en fournissant les données nécessaires obtenues via son port requis et reçoit le résultat obtenu via son port fournis. Ce *moteur de transformation* vise aussi à gérer les liens d'utilisation attachés par des *médiateurs endogènes* afin de les remplacer par des connecteurs complexes qui sont les connecteurs de médiation où :

- Un connecteur de médiation sera un lien de composition entre (*component/component*) ou (*component/Component-of-services*).
- Un port fournis de l'entité source sera relié avec le rôle requis du connecteur de médiation avec un lien d'utilisation.
- Un rôle fournis du connecteur de médiation sera relié avec le port requis de l'entité cible. Le modèle architectural détaillé obtenu représente une description détaillée de l'application mobile composite incluant tous les adaptateurs nécessaires pour assurer le bon fonctionnement pour un contexte mobile spécifique où cette architecture sera associée par un profil d'exécution contenant toutes les informations contextuelles de l'appareil mobile cible.

CONCLUSION

Dans ce chapitre, nous avons présenté le cadre conceptuel du processus de composition proposé. Nous avons abordé dans un premier temps un scénario de composition que nous avons adopté comme une référence tout au long de ce chapitre pour montrer l'efficacité du processus de composition proposé. Après, nous avons présenté le contexte global de notre processus de composition afin de faire expliciter toutes les notions clés manipulées ainsi nous avons défini le fonctionnement global en expliquant le mécanisme opératoire du processus proposé. Pour avoir des applications qui réagissent de façon correcte il est crucial d'intégrer le contexte dès la conception de l'application. Pour cet objectif, nous avons proposé une description contextuelle pour les entités logicielles constitutives de l'application mobile désirée ainsi pour l'environnement d'exécution auquel l'application composite devra être adaptée. Ces descriptions reflètent alors le contexte de composition de l'application mobile.

En utilisant le processus proposé, les développeurs doivent implémenter la structure de leurs applications de façon native sur chacune de leurs plateformes. A cet effet, nous avons proposé un langage de modélisation qui leur permet de définir le modèle fonctionnel de ces applications mobiles. En outre, notre processus de composition aide les développeurs à construire la même application avec des environnements différents, autrement dit avoir plusieurs versions de telle sorte que chaque version sera spécifique à un contexte bien déterminé. Pour cela, nous avons proposé de raffiner le modèle fonctionnel obtenu en attachant toutes les informations contextuelles nécessaires pour la composition. Vu que la composition dirigée par le contexte d'exécution et les besoins des utilisateurs peuvent soulever plusieurs problèmes d'hétérogénéité, nous avons proposé un langage de description architectural dédié pour définir une application mobile composite hétérogène. Le processus proposé repose sur plusieurs moteurs de transformation pour assurer le passage entre les différents modèles de l'application composite afin de générer son architecture détaillée.

Dans le chapitre qui suit, nous allons réaliser le processus de composition proposé en abordant l'implémentation des points contributifs présentés dans ce chapitre. Enfin, nous allons présenter une évaluation basée sur la simulation afin de prouver l'efficacité de fonctionnement de ce processus de composition.

Implémentation et Expérimentation

SOMMAIRE: CHAPITRE 4

Introduction	90
1 Implémentation des méta-modèles proposés	91
1.1 Méta-modèles pour la représentation fonctionnelle	91
1.2 Méta-modèles pour la description architecturale	94
2 Implémentation du processus de composition	101
2.1 Extraction des informations contextuelles	103
2.2 Paramétrage des règles ATL	104
2.3 Ordonnancement automatique des règles ATL	105
3 Résultats et Évaluation	106
3.1 Les résultats du processus de composition CAMAP	106
3.2 L'évaluation du processus de composition CAMAP	111
Conclusion	114
1 Contribution	116
2 Perspectives	117
Production Scientifique et Projets de Recherche	119
Bibliographie	121

INTRODUCTION

Le chapitre précédent a défini le socle conceptuel de notre processus de composition. Cependant, nous visons dans ce chapitre à réaliser un prototype de notre processus de composition en implémentant dans un premier temps les méta-modèles proposés, puis les moteurs de transformation qui assurent le passage entre les modèles représentant l'application mobile composite. Vu que le passage entre une représentation mobile à une autre plus détaillée est implémenté via le mécanisme de transformation *Modèle vers Modèle*, nous choisissons ATL comme un langage pour exprimer les différentes règles de passage et la technologie Java pour réaliser ces transformations. Ce chapitre vise ainsi à évaluer le processus proposée via la simulation du temps de passage entre les modèles proposés pour représenter la CMA désirée en se basant sur deux scénarios différents pour la composition de l'application mobile *ShopReview*. L'objectif est d'illustrer une application des méta-modèles proposés et les moteurs de passage entre les modèles instanciés et de prouver la faisabilité de notre processus de composition à détecter et à gérer les points d'hétérogénéité.

1 Implémentation des méta-modèles proposés

1.1 Méta-modèles pour la représentation fonctionnelle

■ Le méta-modèle fonctionnel

La construction d'une telle application mobile nécessite au préalable l'identification de différentes fonctionnalités comblant les besoins souhaités. L'un des principaux objectifs de notre processus est de présenter un langage spécifique pour décrire l'architecture fonctionnelle de l'application mobile composite désirée. Pour arriver à un modèle fonctionnel décrivant la CMA souhaitée qui est l'objet de la tâche de composition, nous définissons un langage spécifique pour représenter l'application dans un niveau abstrait indépendamment de n'importe quel environnement d'exécution. Le méta-modèle qui définit ce langage est le CMA-FD illustré dans le chapitre précédent (cf. Figure 3.8). Ce méta-modèle est dédié pour définir l'aspect fonctionnel de la CMA requise et dans notre travail nous avons adopté les outils de modélisation d'*Eclipse*, en anglais Eclipse Modeling Tools, pour éditer et modéliser ce méta-modèle afin de pouvoir représenter la CMA fonctionnelle.

La plateforme d'*Eclipse* fournit des outils graphiques pour faciliter l'édition de modèles EMF qui est l'acronyme de Eclipse Modeling Framework (Steinberg et al., 2008). Ce dernier s'inscrit dans le *Framework* du projet des outils pour *Eclipse* basé sur Java et peut être utilisé pour modéliser notre modèle de domaine (le méta-modèle) sous l'extension *.ecore. Par conséquent, nous nous sommes appuyés sur cette technologie pour implémenter le méta-modèle proposé CMA-FD comme illustré dans la Figure 4.1 (les classes en blanc).

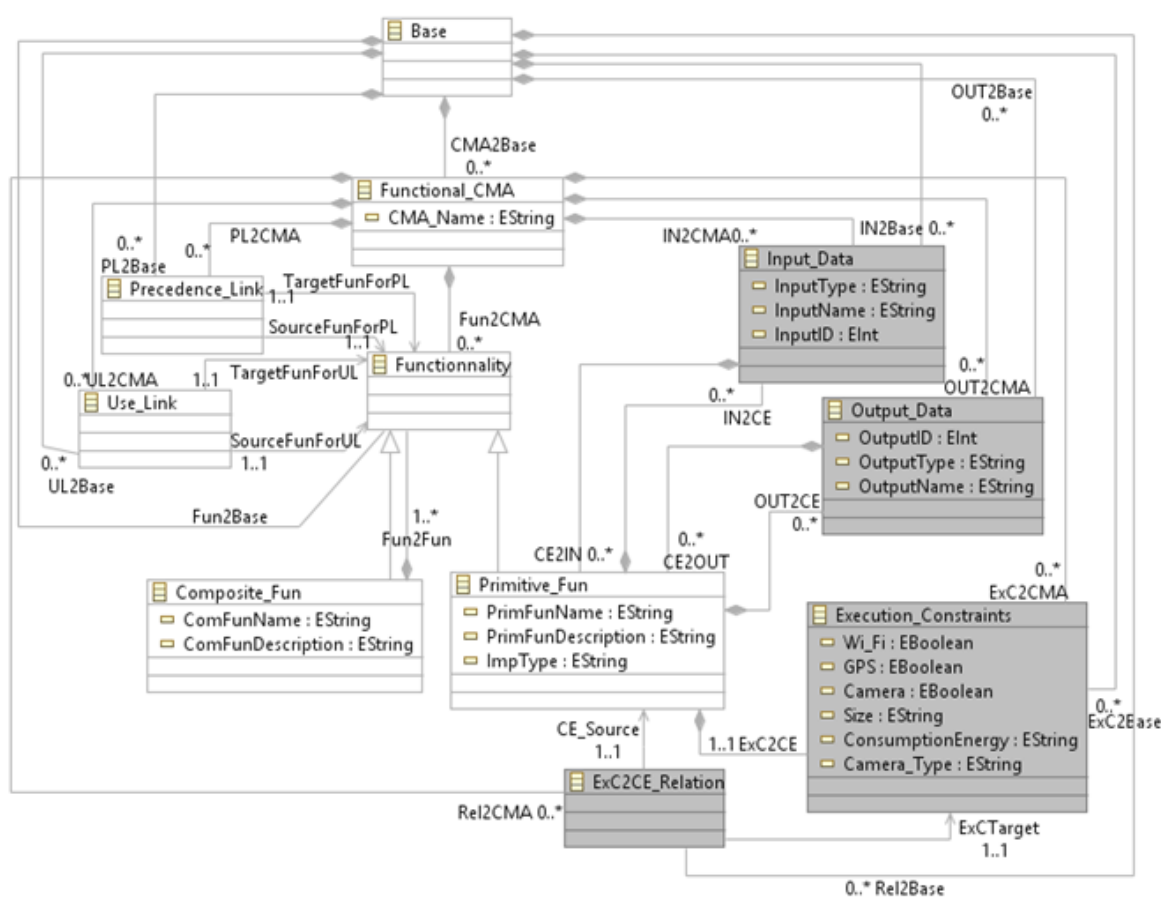


FIGURE 4.1 – Le méta-modèle CMA-FD exprimé en EMF (.ecore)

Le méta-modèle proposé dans le chapitre précédent (cf. Figure 3.8) reflète juste une vision générale sur le langage de description proposé pour décrire le modèle fonctionnel de la CMA souhaitée. Par conséquent, nous avons utilisé la technologie EMF pour fournir la définition détaillée de la grammaire de ce langage incluant toutes les notions nécessaires exprimées par des classes ainsi que toutes les relations de composition entre ces classes suivant les notations présentées par UML.

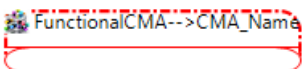
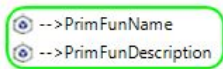
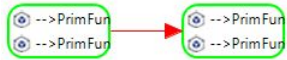

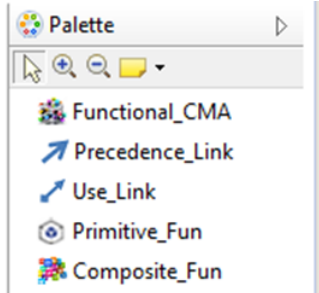
Après avoir défini le langage de description d'un modèle fonctionnel, nous visons maintenant à fournir une représentation graphique aux instances de ce langage. L'instanciation d'un modèle fonctionnel reflète l'opération de création d'un modèle conforme au méta-modèle défini en EMF qui peut se faire par deux façons :

- (1) Instancier le méta-modèle et avoir un modèle sous forme XML.
- (2) Générer un éditeur graphique à partir du méta-modèle par l'utilisation du GMF afin de permettre la création graphique des modèles conformes à ce méta-modèle.

Dans notre travail, nous souhaitons développer un éditeur graphique qui nous permettra de créer des modèles fonctionnels conformes au méta-modèle CMA-FD.ecore. Donc, nous adoptons la technologie GMF qui est l'acronyme de Graphical Modeling Language (Biermann et al., 2006) pour générer une palette de description spécifique qui permet de dessiner graphiquement n'importe quel modèle fonctionnel.

Le Tableau 4.1 montre la palette graphique obtenue ainsi que la syntaxe graphique des différents éléments architecturaux introduits via le méta-modèle CMA-FD.ecore. Par conséquent, chaque notion définie au niveau du CMA-FD.ecore a son équivalence en éléments architecturaux graphiques qui sont dédiés pour définir graphiquement le modèle fonctionnel.

Tableau 4.1 – Syntaxe graphique d'éléments architecturaux du modèle fonctionnel

<i>Nom de l'élément architectural</i>	<i>Description de l'élément architectural</i>	<i>Forme graphique de l'élément architectural</i>
CMA Fonctionnelle	<ul style="list-style-type: none"> • L'application mobile composite souhaitée au niveau fonctionnel : elle possède un nom. 	
Fonctionnalité primitive	<ul style="list-style-type: none"> • La fonctionnalité souhaitée ayant un nom, une description et un type d'implémentation de son entité concrète correspondante. 	
Lien de precedence	<ul style="list-style-type: none"> • Il relie les fonctionnalités requises tout en définissant leur ordre d'invocation. 	
Lien d'utilisation	<ul style="list-style-type: none"> • Il relie les fonctionnalités requises tout en définissant les données échangées entre elles. 	
Palette du modèle fonctionnel	<ul style="list-style-type: none"> • La palette générée : est dédié à la conception des modèles fonctionnels qui sont instanciées du méta-modèle proposé CMA-FD. Cette palette permet de dessiner tous les éléments architecturaux présentés ci-dessous 	

■ *Le méta-modèle fonctionnel raffiné*

Nous avons indiqué précédemment que la tâche de composition d’applications mobiles dépend fortement du contexte de ses environnements d’exécution et que ces environnements ont des caractéristiques hétérogènes et sont soumis aux évolutions du contexte ce qui influe sur le comportement de l’application mobile. Pour cela, les applications mobiles doivent toujours être conformes à leur environnement d’exécution pour garantir l’exécution correcte de ces fonctionnalités. Notre processus traite cette problématique et donne la possibilité aux développeurs de créer plusieurs versions de l’application mobile désirée où chacune d’entre elles sera dédiée pour un appareil mobile donné avec un contexte spécifique. Pour ce faire, nous visons à raffiner le modèle fonctionnel indépendant de n’importe quel environnement d’exécution afin d’avoir un autre raffiné dédié à un contexte spécifique.

Après avoir élaboré la représentation fonctionnelle de la CMA souhaitée en utilisant la palette de description proposée (cf. Tableau 4.1), nous avons besoin de raffiner le modèle fonctionnel obtenu en attachant pour chaque fonctionnalité identifiée les différentes informations contextuelles de l’entité concrète adaptative qui est choisie pour l’implémenter. Les classes en gris présentées dans le méta-modèle CMA-FD.ecore illustré dans la Figure 4.1 représentent les différents éléments architecturaux nécessaires pour obtenir le modèle fonctionnel raffiné. Le Tableau 4.2 montre la syntaxe graphique proposée pour représenter ces éléments architecturaux afin de pouvoir élaborer graphiquement le modèle fonctionnel.

Tableau 4.2 – Syntaxe graphique des éléments architecturaux du modèle fonctionnel raffiné

<i>Nom de l’élément architectural</i>	<i>Description de l’élément architectural</i>	<i>Forme graphique de l’élément architectural</i>
Type d’implémentation	<ul style="list-style-type: none"> Le type de l’entité logicielle concrète choisie pour implémenter la fonctionnalité 	
Donnée d’entrée Donnée de sortie	<ul style="list-style-type: none"> Les données requises par l’entité concrète Les données fournies par l’entité concrète 	
Profil d’exécution	<ul style="list-style-type: none"> Représente les conditions d’exécution qui sont nécessaires pour qu’une entité concrète puisse être déployée correctement 	

Le *moteur de transformation 1* prend le rôle de générer ce modèle fonctionnel raffiné. Ce passage reflète l’exécution d’une succession de règles de transformation définies dans un ordre strict.

L’algorithme suivant (cf. Tableau 4.3) définit une stratégie de propagation qui regroupe l’ensemble des règles correspondantes pour implémenter le passage du modèle fonctionnel vers le modèle fonctionnel raffiné. En outre, il est basé sur l’utilisation des informations contextuelles définies par le modèle d’ontologie des entités logicielles. Cet algorithme sert à extraire les valeurs des paramètres contextuelles et les utiliser pour exécuter les règles de transformation définies dans un ordre spécifique selon l’application traitée.

Il gère les fonctionnalités définies dans le modèle fonctionnel où pour chacune d’entre elles il sert à exécuter à chaque fois une méthode de lecture (*ReadFromOntology-ExProfile*) à partir de l’ontologie des entités logicielles adaptatives pour extraire les informations contextuelles nécessaires. Cette méthode sert à exécuter un ensemble des règles d’inférences pour extraire

les valeurs des paramètres contextuels. Après, il vise à déclencher l'exécution de la règle appropriée en utilisant les valeurs contextuelles obtenues.

Tableau 4.3 – Pseudo code : Algorithme de la génération du modèle fonctionnel raffiné

	Lignes de Code	Commentaires
1	For each Fun i \in IdentifiedFunList Do {	
2	ReadFromOntology-ExProfile (Fun i) {	Lire les informations contextuelles du profil d'exécution de la Fun num i à partir de l'ontologie des entités logicielles
3	WiFi-Value := RequestWiFi(Fun i);	Exécuter une règle d'inférence pour extraire la valeur du paramètre contextuel Wi-Fi
4	GPS-Value := RequestGPS (Fun i);	
5	ImpTyp-Value := RequestTmpTyp(Fun i); ... }	Continuer à extraire toutes les informations contextuelles
6	Execute Add-Ex-Profile Rule;	Exécuter la règle de transformation qui sert à attacher la Fun i par les contraintes d'exécution de l'entité logicielle choisi pour l'implémenter
7	For each InputData j \in InputData(Fun i) {	Pour chaque donnée d'entrée appartient à l'ensemble des données d'entrées de la Fun num i.
8	ReadFromOntology-ExProfile (InputData j, Fun i) {	Lire les informations contextuelles de la donnée d'entrée num j de la Fun num i
9	InputType-Value := Request-InputType(InputData j, Fun i);	Exécuter une règle d'inférence pour extraire la valeur du paramètre contextuel InputType
10	InputName-Value := Request-InputName (InputData j, Fun i);	
11	InputID-Value := Request-InputID (InputData j, Fun i); }	
12	Execute AddInputData Rule }	Exécuter la règle de transformation qui sert à attacher la Fun i par la donnée d'entrée num j de l'entité logicielle choisi pour l'implémenter
13	For each OutputData j \in OutputData(Fun i) {	
14	ReadFromOntology-InputData (OutputData j, Fun i){	Lire les informations contextuelles de la donnée de sortie num j de la Fun num i
15	OutputType-Value := Request-OutputType(OutputData j, Fun i);	Exécuter une règle d'inférence pour extraire la valeur du paramètre contextuel OutputType
16	OutputName-Value := Request-OutputName (OutputData j, Fun i);	
17	OutputID-Value := Request-OutputID (OutputData j, Fun i); }	
18	Execute AddOutputData Rule }	Exécuter la règle de transformation qui sert à attacher la Fun i par la donnée de sortie num j de l'entité logicielle choisie pour l'implémenter

1.2 Méta-modèles pour la description architecturale

■ *Le méta-modèle architectural*

Après avoir obtenu le modèle fonctionnel raffiné, il est nécessaire de remplacer les différentes fonctionnalités par les entités logicielles concrètes correspondantes et assurer une composition consistante entre elles. Pour cela nous avons proposé un autre langage de description pour définir le modèle architectural de la CMA désirée qui est la cible de la tâche de composition. Ce langage est dédié pour représenter la CMA via ces entités logicielles concrètes adaptatives ainsi que les différents points hétérogénéités posés par les coordinations des entités de différents types et/ou qui manipulent des données non compatibles. La grammaire de ce

langage est définie via le méta-modèle H^2CMA -AD présenté précédemment. Ce méta-modèle est aussi exprimé via la technologie EMF comme illustré dans la Figure 4.2 où plusieurs détails doivent être spécifiés pour pouvoir représenter le modèle architectural de la CMA : les informations contextuelles des entités concrètes adaptatives, les points d'hétérogénéité dans les cas des combinaisons hétérogènes.

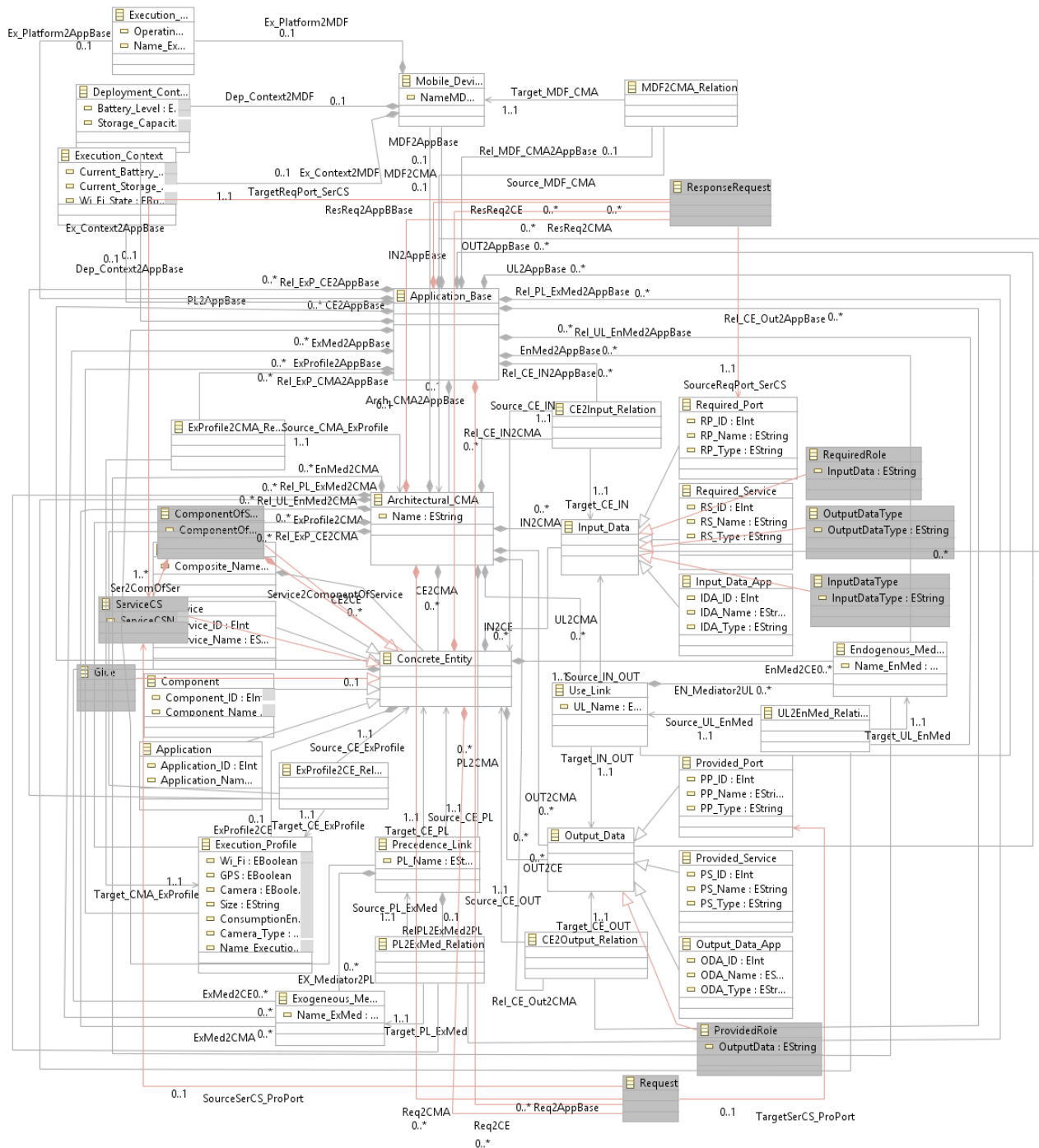


FIGURE 4.2 – Le méta-modèle H^2CMA -AD.ecore exprimé en EMF (.ecore)



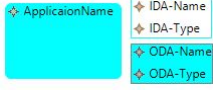

L'architecture de la CMA (la classe *Architecture-CMA*) regroupe un ensemble d'entités logicielles concrètes définies au niveau abstrait (la classe *Concrete-Entity*). Une entité logicielle peut être soit un composant (la classe *Component*) soit un service (la classe *Service*) ou autre (la classe *Application*) tandis que la classe *Composite-CE* reflète une fonctionnalité

composite. Chaque entité logicielle concrète doit être attachée par ses données requises (la classe *Input-Data*) via la relation représentée par la classe *CE2Input-Relation* et par ses données fournies (la classe *Output-Data*) via la relation représentée par la classe *CE2Output-Relation*. Les données requises d’une entité concrète de type composant seront représentées par des ports requis (la classe *Required-Port*) tandis que les données fournies seront représentées par des ports fournis (la classe *Provided-Port*). Ainsi, les données fournies et requises pour une entité de type service ou une application seront représentées respectivement via *Provided-Service*, *Required-Service*, *OutputData-App*, *InputData-App*.

Les liens de précédences (la classe *Precedence-Link*) qui relie deux entités de différents types seront attachés par des médiateurs exogènes (la classe *Exogenous-Mediator*) via la relation exprimée par la classe *PL2ExMed-Relation*. Les liens d’utilisation (la classe *Use-Link*) qui relie deux entités qui n’ont pas la même nature seront attachés par des médiateurs endogènes (la classe *Endogeneous-Mediator*) via la relation exprimée par la classe *UL2EnMed-Relation*. Chaque entité est associée par son profil d’exécution (la classe *Execution-Profile*) via la relation *ExProfile2CE-Relation* ainsi que l’application mobile composite souhaitée doit être attachée à son propre profil d’exécution via la relation *ExProfile2CMA-Relation*.

Après avoir défini tous les éléments architecturaux ainsi que les différentes relations utilisées pour représenter le modèle architectural dans le méta-modèle *H²CMA-AD.ecore*, nous avons implémenté ce méta-modèle suivant toutes les étapes définies par la technologie GMF afin d’obtenir une représentation graphique pour le modèle architectural du CMA. Le Tableau 4.4 montre la syntaxe graphique dédiée pour définir ces éléments architecturaux.

Tableau 4.4 – Syntaxe graphique des éléments architecturaux du modèle architectural

Nom de l'élément architectural	Description de l'élément architectural	Forme graphique de l'élément architectural
Composant Port requis Port fournis	<ul style="list-style-type: none"> Entité logicielle de type composant Supporte les données d’entrée du composant Supporte les données de sortie du composant 	
Service Service requis Service fournis	<ul style="list-style-type: none"> Entité logicielle de type service Supporte les données d’entrée du service Supporte les données de sortie du service 	
Application Donnée d’entrée de l’app Donnée de sortie de l’app	<ul style="list-style-type: none"> Entité logicielle de type application Supporte les données d’entrée de l’application Supporte les données de sortie de l’application 	
Médiateur exogène Médiateur endogène	<ul style="list-style-type: none"> Sert à indiquer et éliminer l’hétérogénéité entre les deux entités de différents types Sert à assurer la compatibilité des données échangées qui sont hétérogènes 	

Le *moteur de transformation 2* prend la responsabilité de générer le modèle architectural du CMA. Ce passage reflète l’exécution d’une succession de règles de transformation spécifiques définies dans un ordre bien défini. Il vise à manipuler la représentation fonctionnelle raffinée obtenue afin de dériver le modèle architectural en remplaçant chaque fonctionnalité par son entité concrète appropriée qui est contextuelle. Le passage du modèle fonctionnel raffiné à

la représentation architecturale implique le respect d'un certain nombre de contraintes que nous avons posées pour constituer l'application composite, via les entités logicielles concrètes choisies, à un niveau abstrait.

Un exemple d'une contrainte de composition est la suivante : « Composer une fonctionnalité Fun1 implémentée par un composant avec une fonctionnalité Fun2 implémentée par un service nécessite d'associer le lien de précédente avec un médiateur exogène afin d'éliminer le problème d'hétérogénéité de la communication entre ces entités concrètes reliées, i.e. proposer de fournir des interfaces de communications compatibles ». L'algorithme suivant (cf. Tableau 4.5) représente le fonctionnement du *moteur de transformation 2* en respectant toutes les contraintes de composition nécessaires pour avoir un modèle architectural consistant.

Tableau 4.5 – Pseudo code : Algorithme de la génération du modèle Architectural

	Lignes de Code	Commentaires
1	Execute RefinedFun2ArchCMA rule	Remplacer chaque Fun par son SE appropriée et aussi toutes les données d'entrées et de sorties par les éléments architecturaux proposés pour les représenter (ProvidedPort, Required Port, Provided Service, etc)
2	For each Precedence-Link {	Pour chaque lien de précédence
3	If (SourceFun.ImpType compared with CibleFun.ImpType) then	Vérifier si la Fun source et la Fun cible ont le même type d'implémentation des SE contextuelles appropriées
4	Execute AddPrecedenceLink Rule ;	Ajouter un lien de précédence si elles ont le même type
5	Else Execute AddPrecedenceLink-Ex-Med Rule ;	Ajouter un lien de précédence attaché par un médiateur exogène dans le cas contraire
6	For each SourceFun.InputData do {	Pour chaque donnée d'entrée de la Fun source
7	For each CibleFun.OutputData do {	Pour chaque donnée de sortie de la Fun cible
8	if (SourceFun.Inputdata.ID compared with CibleFun.OutputData.ID) then {	Gérer les liens d'utilisation
9	if (SourceFun.Inputdata.Type compared with CibleFun.OutputData.type) then	Vérifier la compatibilité des données échangées
10	Execute AddUseLink Rule ;	Types des données échangées compatibles : ajouter un lien d'utilisation entre la donnée d'entrée de la Fun source avec la donnée de sortie de la Fun cible.
11	else Execute AddUseLink-En-Med Rule ; }	Types des données échangées non compatibles : ajouter un lien d'utilisation attaché par un médiateur endogène
12	} }	
13	Derive Contextual Parameter's values of the CMA execution profile ;	Déduire les contraintes d'exécution de la CMA à partir des informations contextuelles de ces SEs constitutives
14	Execute Add-Ex-Profile Rule ;	Attacher la CMA par son propre profil d'exécution

Le *moteur de transformation 2* vise dans un premier temps à remplacer les fonctionnalités identifiées par ces entités concrètes en appliquant une règle nommée *RefinedFun2ArchCMA*. Après, il gère les liens d'utilisation et les liens de précédence pour relier à nouveau les entités remplacées et décider s'il est nécessaire d'attacher ces relations de composition par des médiateurs. Donc il sert à comparer les types d'implémentation des entités logicielles à relier afin d'associer les liens de précédence par des médiateurs exogènes si les types d'implémentation ne sont pas compatibles, i.e. appliquer la règle *AddPrecedenceLink-Ex-Med* et la règle *AddPrecedenceLink* dans le cas contraire. Ainsi, il vise à comparer les types des données échangées afin d'associer les liens d'utilisation par des médiateurs endogènes si les données échangées ne sont pas compréhensibles, i.e. appliquer la règle *AddUseLink-Ex-Med* et la règle *AddUseLink* dans le cas contraire. Ensuite, le *moteur de transformation 2* est conçu pour générer pour chaque CMA son propre profil d'exécution tout en déclenchant la règle *Add-Execution-Profile*. Il a la possibilité de remplir le profil d'exécution de la CMA obtenue en

se basant sur les différentes contraintes d'exécution de leurs constituantes et les informations contextuelles de l'appareil mobile visé via les modèles d'ontologies proposées.

■ *Le méta-modèle architectural détaillée*

Il est nécessaire maintenant de générer l'architecture détaillée de notre CMA souhaitée. Après avoir eu le modèle architectural qui représente l'application mobile composite via les entités concrètes choisies pour implémenter les exigences identifiées incluant tous les adaptateurs nécessaires, nous visons à raffiner ce modèle pour spécifier les rôles des médiateurs intégrés. L'objectif est de fournir une description plus détaillée de la CMA désirée facilitant par la suite la génération de la CMA concrète.

Les classes grisées illustrées dans le méta-modèle H^2 CMA-AD.ecore (cf. Figure 4.2) indiquent les éléments architecturaux qui seront utilisés pour remplacer les médiateurs endogènes et les médiateurs exogènes qui sont indiqués dans le modèle architectural.

Un médiateur endogène qui vise à assurer la compatibilité des données échangées sera remplacé par un connecteur de médiation (la classe *MediationConnector*). Ce connecteur possède un rôle requis (la classe *RequiredRole*) et un autre fournis (la classe *ProvidedRole*). Le rôle requis est dédié pour supporter la donnée à transformer qui est fournie par l'entité source tandis que le rôle fournis vise à extraire la donnée transformée via ce connecteur de médiation dont l'objectif est de transférer le résultat obtenu à l'entité cible.

Comme nous avons précisé précédemment, ce connecteur doit connaître le type de donnée à transformer ainsi que le type requis pour pouvoir appeler le service de médiation approprié. Ces informations seront représentées via les éléments architecturaux *InputDataType* et *OutputDataType*. Un connecteur de médiation sert à comparer ces types des données afin d'exécuter dans le cas de la non compatibilité deux méthodes nommée respectivement *Search MD* et *Call MD*. La première méthode sert à chercher le service de médiation nécessaire pour assurer la compatibilité des données et la deuxième à l'exécuter.

Un médiateur exogène qui vise à fournir des interfaces bien formées et compatibles pour pouvoir assurer la communication entre les entités exogènes sera remplacé par une nouvelle entité qui est représentée par la classe *Component-of-services*. Cet élément architectural regroupe (encapsule) les entités de type service qui sont reliées avec des composants (voir la relation de composition entre la classe *Service* et la classe *Component-of-services*). Par conséquent, cette nouvelle entité est considérée elle-même comme une entité concrète de type composant (l'entité mère) qui regroupe un ensemble de services et possède des ports fournis (la classe *ProvidedPort*) et des ports requis (la classe *RequiredPort*).

Les services requis et fournis (*RequiredService* et *ProvidedService*) de l'entité service encapsulé seront transformés respectivement en un port requis et un autre fournis pour l'entité mère. La donnée d'entrée qui est représentée par le port requis sera transférée via une requête (la classe *Request*) pour déclencher le service approprié. En conséquence, une réponse pour cette requête sera récupérée et transmise (la classe *ResponseRequest*) à un port fournis de l'entité mère. Le Tableau 4.6 montre la syntaxe graphique proposée pour représenter ces éléments architecturaux afin de pouvoir représenter graphiquement le modèle architectural détaillé de la CMA souhaitée.

Nous avons associé la génération du modèle architectural détaillé au *moteur de transformation 3*. Ce passage indique une opération de remplacement qui est effectuée via l'exécution d'une succession de règles de transformation qui sont définies dans un ordre spécifique suivant un ensemble de contraintes de composition. Ces dernières reflètent les spécifications que nous avons proposées pour représenter les médiateurs endogènes et exogènes. L'algorithme suivant (cf. Tableau 4.7) représente le fonctionnement du *moteur de transformation 3* en respectant toutes les contraintes de composition nécessaires pour avoir le modèle architectural détaillé, i.e. une stratégie d'exécution des règles de passage.

Tableau 4.6 – Les éléments architecturaux de l’architecture détaillée de la CMA


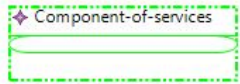






Nom de l'élément architectural	Description de l'élément architectural	Forme graphique de l'élément architectural
Entité service	<ul style="list-style-type: none"> Entité logicielle de type service 	
Component-of-services	<ul style="list-style-type: none"> Une nouvelle entité qui vise à encapsuler les entités de type service 	
Requête Réponse	<ul style="list-style-type: none"> Sert à déclencher l'exécution d'un service tout en fournissant les données nécessaires Diffuser la donnée obtenue au port fourni 	
Glu	<ul style="list-style-type: none"> Le noyau du connecteur. Il a pour but d'exécuter deux fonctions (Search mediation service et Call mediation service) afin d'éliminer l'hétérogénéité des données échangées. 	
Type de la donnée d'entrée Type de la donnée de sortie	<ul style="list-style-type: none"> Représente le type de la donnée à transformer Représente le type vers lequel la donnée sera transformée 	 
Rôle requis Rôle fournis	<ul style="list-style-type: none"> Une interface qui supporte la donnée à transformer Une interface pour recevoir et garder la donnée transformée 	 

Tableau 4.7 – Pseudo code : Algorithme de la génération du modèle architectural détaillé

	Lignes de Code	Commentaires
1	For each PL do {	Gérer les liens de précédences
2	If (PLi attachedwith ExMed) then {	Si on trouve un médiateur exogène
3	If (cibleSE is Service) then {	Si l'entité cible de ce lien de précédence est de type service
4	Execute Delete ExMed Rule ;	Supprimer le médiateur exogène
5	Execute Delete PL(SourceSE, CibleSE) Rule ;	Supprimer ce lien de précédence
6	Execute Add-ComOfSer Rule ;	Ajouter une nouvelle entité mère (ComponentOfService)
7	Execute EncapsuleSer (CibleSE) Rule ;	Encapsuler l'entité service dans la nouvelle entité mère ajoutée
8	Execute AddPL-Link (SourceSE, ComOfSer) ;	Restaurer le lien de précédence entre l'entité source et la nouvelle entité
9	For each UL (SourceSE, CibleSE) {	Récupérer les ULs qui ont été entre l'entité composant et l'entité service pour qu'ils seront des ULs entre l'entité composant et la nouvelle entité.
10	ComOfSer-ReqPort :=ReqSer (CibleSE) ;	Transformer le service requis de l'entité service à un port requis pour l'entité mère
11	Execute AddReqPort (ComOfSer-ReqPort, ComOfSer) Rule ;	Exécuter une règle de transformation qui sert à ajouter le port requis à la nouvelle entité
12	Execute AddRequest (ComOfSer-ReqPort, CibleSE) Rule ;	Ajouter une requête entre le port requis ajouté et l'entité service pour pouvoir déclencher le service et transférer les données requises pour son exécution

13	Execute Add-UL (Source-ProPort, ComOfSer-ReqPort) Rule;	Ajouter un lien d'utilisation entre le port fournis de l'entité source (composant) et le nouveau port requis de l'entité mère
14	Delete-UL;	Supprimer l'ancien lien d'utilisation
15	Execute Delete- ReqSer (CibleSE); }	Supprimer le service requis de l'entité cible service
146	While (CibleSE.Cible is Service) do {	Vérifier si l'entité cible qui relie l'entité service est aussi de type service
17	SourceSE := CibleSE; CibleSE := CibleSE.Cible;	
18	Encapsule (CibleSE);	Encapsuler l'entité cible de l'entité encapsulée service parce qu'elle est aussi de type service
19	Execute AddPL-Link (SourceSE, CibleSE);	Ajouter un lien de précedence entre l'entité service source et l'entité service cible
20	For each UL (SourceSE, CibleSE){	Gérer les liens d'utilisation entre ces deux entités
21	ComOfSer-ProPort :=ProSer (SourceSE);	Le service fournis de l'entité service source devient un port fournis pour l'entité mère
22	Execute AddProPort (ComOfSer-ProPort, ComOfSer) Rule;	Exécuter la règle qui ajoute ce port fournis
23	Execute AddResponseRequest (SourceSE,ComOfSer-ProPort) Rule;	Ajouter une requête de réponse entre le port fournis ajouté de l'entité mère et l'entité service source
24	ComOfSer-ReqPort :=ReqSer (CibleSE);	Le service requis de l'entité service cible devient un port requis pour l'entité mère
25	Execute AddReqPort (ComOfSer-ReqPort, ComOfSer) Rule;	
26	Execute AddRequest (ComOfSer-ReqPort, CibleSE) Rule;	
27	Execute AddUL (ComOfSer.ProPort, ComOfSer-ReqPort) Rule;	Ajouter un UL entre le port requis (qui était un service requis de l'entité service cible)de l'entité mère et le port fournis (qui était un service fournis de l'entité service source)
28	Execute Delete-UL Rule;	Supprimer les anciens ULs et PLs et les rajouter suivant la nouvelle représentation
29	Execute AddUL (ComOfSer.ProPort, ComOfSer-ReqPort) Rule;	...
30	Execute Delete-UL Rule;	...
31	Execute Delete-ProSer (SourceSE) Rule; Execute Delete-ReqSer (CibleSE) Rule;}	...
32	Execute Add-PL (ComOfSer, CibleSE);	...
33	For each UL (SourceSE, CibleSE) {	...
34	ComOfSer-ProPort :=ProSer (CibleSE);	...
35	Execute AddProPort (ComOfSer-ProPort, ComOfSer) Rule;	...
36	Execute AddUL (ComOfSer.ProPort, CibleSE.ReqPort) Rule;	...
37	Delete UL ; Delete ProSer (SourceSE); }}	...
38	For each UL do {	Gérer les liens d'utilisation
39	If (ULi attachedwith EnMed) then {	Vérifier si UL est attaché avec un médiateur endogène
40	ComOfSer-ProPort :=ProSer (CibleSE);	Changer le médiateur par un connecteur de médiation
41	Execute Delete-UL (SourceSE.ProPort, CibleSE.ReqPort) Rule;	...
42	Execute Delete-EnMed Rule;	...
43	Execute Add-MedConnector Rule;	...
44	Execute Add-UL (SourceSE.ProPort, MedConnector.ReqRole) Rule;	Relier le rôle requis du connecteur de médiation avec le port fournis de l'entité service source
45	Execute Add-UL (MedConnector.ProRole, CibleSE.ReqPort) Rule; }	Relier le rôle fournis du connecteur de médiation avec le port requis de l'entité service cible

Le *moteur de transformation 3* gère les liens de précédence et les liens d'utilisation dont l'objectif est de détecter ceux qui sont associés par des médiateurs où :

- A chaque fois où un médiateur endogène est trouvé, il déclenche un ensemble de règles pour le remplacer par un connecteur de médiation ; et
- A chaque fois où un médiateur exogène est trouvé, il déclenche un ensemble de règles pour le remplacer par une entité *Component-of-services*. Après avoir encapsulé l'entité service, le *moteur de transformation 3* doit effectuer un test pour vérifier si l'entité service est liée avec une autre du même type. Donc, il vise à encapsuler toutes les entités liées qui sont du type service jusqu'à ce qu'il arrive à une composition de type service/composant.

2 Implémentation du processus de composition

Tout d'abord, nous rappelons ici que la tâche de composition d'applications mobiles adaptatives suivant le processus proposé reflète une succession de transformations de modèles. Nous implémentons le passage entre les différents modèles proposés en utilisant les mécanismes de transformation de modèles (Czarnecki and Helsen, 2003). Le langage ATL (Atlas Transformation Language) représente notre choix technologique pour définir les différentes règles de passage nécessaires et la technologie Java pour les invoquer et les exécuter. Les *moteurs de transformation* sont basés sur les structures XML des modèles sources et les informations contextuelles définies par les modèles d'ontologies proposés. En se basant sur ces informations, les *moteurs de transformation* peuvent choisir les règles ATL convenables et les déclencher dans un ordre spécifique afin de générer les modèles cibles appropriés. Dû au fait que le langage de transformation de modèles ATL ne supporte pas la notion de paramétrage de ses règles de transformation, nous avons proposé un modèle de paramètres dédié spécialement pour stocker et identifier à chaque fois les valeurs nécessaires pour exécuter les règles ATL.

La Figure suivante (cf. Figure 4.3) illustre une vision générale sur le fonctionnement d'un moteur de passage au niveau d'implémentation, à savoir elle résume les différentes tâches effectuées par un moteur de transformation.

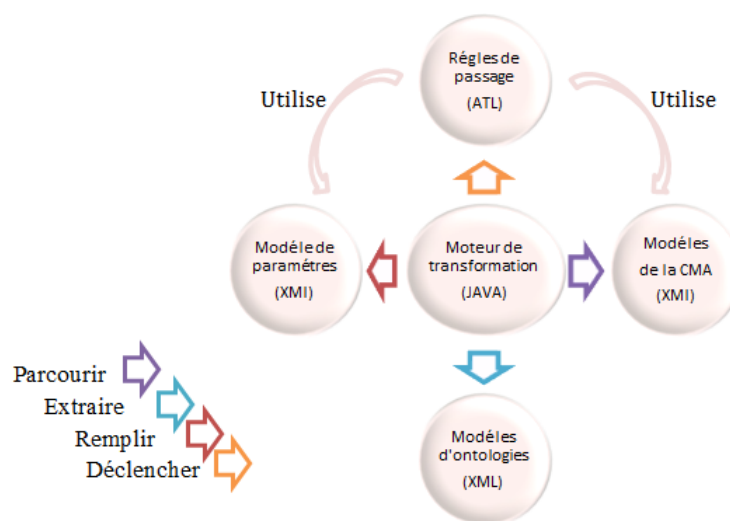


FIGURE 4.3 – Implémentation d'un moteur de passage (les choix technologiques)

Tous les *moteurs de transformation* possèdent le même principe de fonctionnement mais chacun d'entre eux :

- Fait le parcours d'un modèle spécifique de la CMA (ex. le *moteur de transformation 1* utilise le modèle fonctionnel et génère le modèle fonctionnel raffiné et le *moteur de transformation 2* utilise le modèle fonctionnel raffiné et génère le modèle architectural);
- Exécute un ensemble des règles spécifiques à son rôle (ex. la règle *Add-Ex-Profile* pour le *moteur de transformation 1* et la règle *RefinedFun2ArchCMA* pour le *moteur de transformation 2*).

L'interaction entre les différents modèles proposés ainsi que les règles définies est effectuée au sein du moteur de passage via la technologie Java. Par conséquent, un moteur de passage vise dans un premier temps à extraire les informations contextuelles nécessaires des entités concrètes et de l'appareil mobile cible à partir des modèles d'ontologies. Après, il vise à sauvegarder les informations obtenues dans le modèle de paramètres. Cependant, les règles ATL à exécuter reposent sur l'utilisation des valeurs définies au niveau de ce modèle de paramètres pour qu'elles puissent fonctionner. Comme il vise aussi à extraire des informations à partir des modèles XML des représentations fournies afin d'effectuer des comparaisons nécessaires pour pouvoir identifier le type de la règle à déclencher.

Concernant les *moteur de transformation 1* et 3, le processus de composition utilise le méta-modèle CMA-FD exprimé via la technologie EMF à la fois comme un méta-modèle source et cible. Ces passages correspondent à effectuer une transformation endogène comme illustrés dans la Figure suivant :

```

-- @path MM=/Functional_M2RefinedFunctional_M/MetaModels/
CMA-FD-Metamodel.ecore
-- @path
CEinformation=/Functional_M2RefinedFunctional_M/MetaModels/
ParametreMetaModel.ecore
module AddExConstraints;
create RefinedFunctionalModel: CMA-FD-Metamodel refining
FunctionalModel : CMA-FD-Metamodel,
Parameter : CEinformation;
    
```

FIGURE 4.4 – Transformation endogène via le langage ATL

Contrairement au *moteur de transformation 2* qui est basé sur CMA-FD comme un méta-modèle source et sur H^2 CMA-AD comme un méta-modèle cible. Ce passage correspond à une transformation exogène comme illustré dans la Figure suivante :

```

-- @path
ArchCMAmetamodel=/Refined_Fun_Model2Architectural_CMA/MetaModels/CMA-AD-Metamodel.ecore
-- @path
RefinedFunMetamodel=/Refined_Fun_Model2Architectural_CMA/MetaModels/CMA-FD-Metamodel.ecore
module RefinedFun2ArchCMA;
create ArchCMAModel : CMA-AD-Metamodel from RefinedFunModel
: CMA-FD-Metamodel;
    
```

FIGURE 4.5 – Transformation exogène via le langage ATL

Dans les sous-sections suivantes nous allons expliquer les différentes tâches effectuées par le *moteur de transformation* pour générer les modèles souhaités tout en montrant les différents outils et les langages adoptés.

2.1 Extraction des informations contextuelles

Dans ce travail de recherche nous portons une attention particulière au modèle d'ontologie qui est considéré parmi les modèles les plus utiles pour représenter et traiter le contexte et les plus adoptés pour les environnements mobiles. L'ontologie permet d'ajouter l'aspect sémantique aux données contextuelles comme elle permet principalement de créer des vocabulaires partagés entre les systèmes. Pour cette raison, nous pensons que ce modèle de description du contexte est le plus adéquat pour couvrir l'objectif de faire face à l'hétérogénéité des dispositifs mobiles et de composer des applications mobiles adaptables à ces environnements d'exécution.

En outre, le modèle d'ontologie couvre les inconvénients d'autres modèles de description du contexte en permettant la modélisation des données dynamiques ainsi de saisir une variété de types de contexte. Pour ce faire, nous avons adopté la notation d'ontologies pour décrire le contexte de composition d'une application mobile suivant les descriptions présentées précédemment (cf. Figure 3.5, Figure 3.6). Un *moteur de transformation* a besoin de manipuler certaines informations pour pouvoir identifier la règle à exécuter, i.e. il décide d'exécuter une telle règle suivant une comparaison effectuée au niveau des informations contextuelles. A titre d'exemple, le *moteur de transformation 1* a besoin comme source des informations contextuelles des entités concrètes qui sont compatibles avec les caractéristiques d'appareils mobiles afin de lui permettre d'accomplir sa tâche. Dans notre processus de composition, nous supposons que les entités concrètes contextuelles qui sont nécessaires pour mettre en œuvre les fonctionnalités identifiées sont déjà choisies et toutes les informations contextuelles de ces entités sont exprimées suivant le modèle d'ontologies proposé par le concepteur de l'application.

Les ontologies sont basées sur le standard XML/RDF pour représenter ses modèles instanciés ce qui donne la possibilité d'interagir facilement avec ces modèles en utilisant la technologie Java. L'objectif est de sauvegarder et d'extraire les données contextuelles requises. Tirant profit de cet avantage et afin de pouvoir manipuler et extraire ces informations contextuelles stockées nous fournissons une gamme de règles d'inférence organisées suivant les différentes catégories proposées. Nous avons choisi le langage d'interrogation SPARQL pour exprimer ces règles et la technologie Java pour déclencher l'exécution de ces règles. Nous adoptons la technologie Java pour attacher la notion de généricité aux règles proposées et donc les exprimer une fois pour toutes, ce qui donne la possibilité d'exécuter les mêmes règles mais avec des paramètres changeables. La Figure suivante (cf. Figure 4.6) montre un exemple d'une règle SPARQL exprimée en code java.

```
1: public String prefix = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"+
3:     "PREFIX owl: <http://www.w3.org/2002/07/owl#>"+
4:     "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>"+
5:     "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>"+
6:     "PREFIX se: <http://softwareentity.ontology.fr.org/MobileDevice/>";
7: String FunctionalityName = TextField2.getText();
8: String RequestGetPlat-Type = this.prefix+
9: " SELECT DISTINCT ?SE ?FunName ?ReqPlatFormeType"+
10: " WHERE { ?SE se:hasFunctionalAspects ?FA. "+
11: " ? SE se:hasNonFunctionalAspects ?NFA. "+
12: " ? NFA se:hasReqPlatformChar ?ReqPlatChar. "+
13: " ? FA se:hasFuncName ?FunName. "+
14: " ReqPlatChar se:hasReqPlatformType ?ReqPlatFormeType"+
15: " FILTER (CONTAINS (?FunName, '"+ FunctionalityName + "' + "));
```

FIGURE 4.6 – Règle SPARQL avec des paramètres génériques (code java)

Elle sert à extraire le type de la plateforme requise pour l'exécution de l'entité logicielle qui implémente la fonctionnalité *Read Barcode*. A titre d'exemple, pour obtenir les contraintes d'exécution de cette entité il faut juste indiquer le nom de la fonctionnalité qu'elle sert à l'implémenter. Dans ce cas-là des règles SPARQL seront déclenchées automatiquement et les informations contextuelles demandées seront obtenues.

2.2 Paramétrage des règles ATL

La nécessité derrière la définition du modèle de paramètres est de répondre à la limite du langage de transformation ATL, i.e. manque de la notion de généricité lors l'exécution des règles ATL, pour pouvoir réaliser une transformation paramétrée. A cet effet, nous avons proposé un méta-modèle de paramètres (E) dont l'objectif est de supporter des attributs avec des valeurs changeables qui reflètent les paramètres définis dans les règles ATL (cf. Figure 4.7). Par conséquent, le modèle de paramètres qui est instancié de ce méta-modèle contient les valeurs des paramètres définis au niveau des règles ATL pour effectuer une transformation spécialisée.

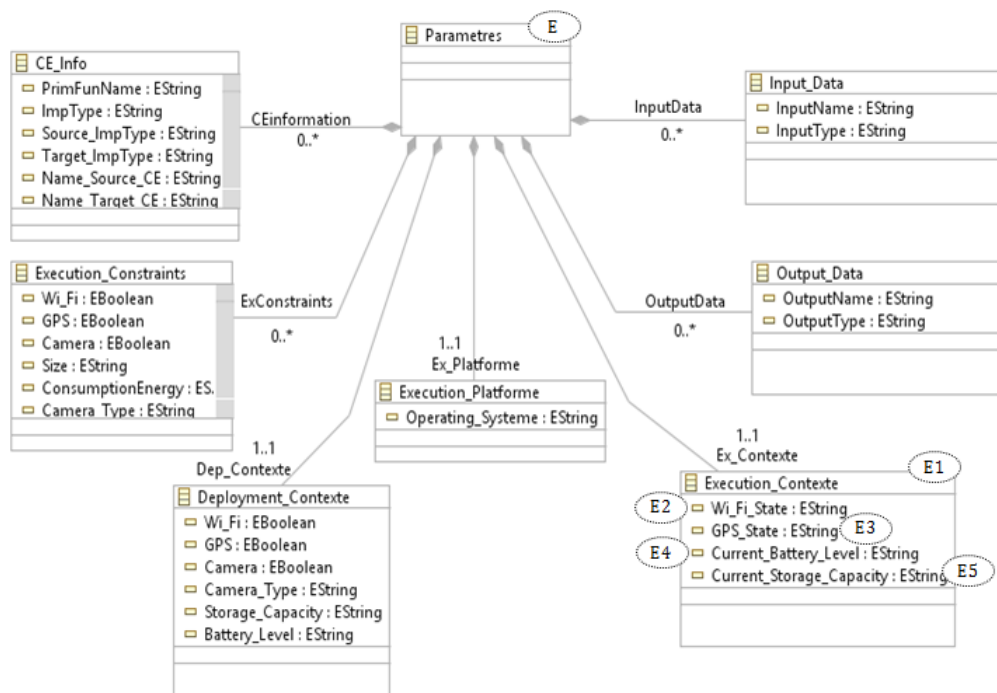


FIGURE 4.7 – Méta-modèle de paramètres

Chaque élément architectural défini au sein de ce méta-modèle regroupe un ensemble d'attributs organisés selon la structuration définie par les ontologies proposées. A titre d'exemple, nous avons la classe *Execution-Contexte* (E1) qui supporte les informations contextuelles de l'appareil mobile cible. Ces informations sont classifiées dans la catégorie Contexte d'exécution dans l'ontologie proposée pour l'appareil mobile. Donc, cette classe supporte les états actuels des différentes ressources existantes sur un tel appareil mobile (E2)(E3) (E4)(E5). Ces attributs portent les valeurs des paramètres contextuels selon l'étude du cas traité, i.e. l'application mobile souhaitée et le contexte d'exécution choisi.

L'instanciation de ce méta-modèle engendra un modèle de paramètres dans un format XMI (cf. Figure 4.8) où la manipulation des données introduites par ce modèle se fait à l'aide de la technologie Java.

```

<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="Parametres">
  <Parametres>
    <InputData InputName="ProductName" InputType="String" />
    <OutputData OutputName="Barcode" OutputType="Integer"/>
    <ExConstraints Wi_Fi="false" GPS="true" Camera="true" Size="16"
      ConsumptionEnergy="45" Camera_Type="AutoFocus"/>
    <CEinformation PrimFunName="Fun1" ImpType="Service" Source_ImpType="" Target_ImpType=""
      Name_Source_CE="" Name_Target_CE="" />
    <Dep_Contexte Wi_Fi="false" GPS="true" Camera="true" Camera_Type="AutoFocus"
      Battery_Level="12" Storage_Capacity="12"/>
    <Ex_Platforme Operating_Systeme="45"/>
    <Ex_Contexte Wi_Fi_State="false" GPS_State="false" Current_Storage_Capacity="45"
      Current_Battery_Level="45"/>
  </Parametres>
</xmi:XMI>
    
```

FIGURE 4.8 – Modèle de paramètres

Cependant, les différentes règles de passage sont écrites une fois pour toutes. Dans chacune d'entre elles, nous avons défini un ensemble de *helpers* d'ATL qui servent soit à extraire les informations nécessaires à partir des attributs présentés dans le modèle de paramètres soit d'effectuer différents tests (ex. vérifier si la fonctionnalité à laquelle nous allons attacher le type d'implémentation si existe dans le modèle fonctionnel). La Figure 4.9 montre une règle de passage générique écrite en ATL qui vise à associer la fonctionnalité *Read Barcode* avec le type d'implémentation de l'entité concrète contextuelle qui sera utilisée pour l'implémenter. Cette règle de passage contient deux *helpers* tels qu'illustrés dans la Figure 4.9.

Le premier vise à extraire le nom de la fonctionnalité pour laquelle nous avons besoin d'ajouter le type d'implémentation à partir du modèle de paramètres afin de vérifier son existence et de la sélectionner dans le modèle fonctionnel (*TestPrimitiveFun*). Le second *helper* permet d'extraire le type d'implémentation choisi pour cette fonctionnalité à partir du modèle de paramètres et le mettre dans une variable (*ImpType*) qui sera utilisée par la suite pour attacher l'information contextuelle obtenue à la fonctionnalité sélectionnée.

```

-- @path MM=/Functional-M2RefinedFunctional-M/MetaModels/Functional-RefinedFunctional-CMA-
  Metamodel.ecore
-- @path CEinformation=/Functional-RefinedFunctional_M/MetaModelsParametreMetaModel.ecore
module AddImpType;
create OUT: MM refining IN: MM, Parameter: CEinformation;
-- Select the Fun in which we will add the Implementation Type --
helper context MM!Primitive_Fun def : TestPrimFun : Boolean = let x : String =
  CEinformation!CE_Info.allInstances()->collect(p| p.PrimFunName)->first() in
  if(self.PrimFunName=x)then true else false endif;
-- Add Implementation Type for this Fonctionnalité --
helper def: ImpType: String =CEinformation!CE_Info.allInstances()->collect(p| p.ImpType);
rule AddImpType{
from e: MM!Primitive_Fun(e.TestPrimFun)
  to c:MM!Primitive_Fun
  (ImpType<-thisModule.ImpType)}
    
```

 FIGURE 4.9 – La règle ATL *AddImpType*

2.3 Ordonnancement automatique des règles ATL

Notre mécanisme de composition nécessite un déclenchement implicite des règles de passage dans un ordre spécifique. Les règles à exécuter et leur ordre d'invocation doivent

être identifiés en fonction des différentes contraintes de composition tout en effectuant des comparaisons au niveau des informations liées aux modèles descriptifs de la CMA. En plus du fait que l'ATL ne supporte pas la notion de généricité, ce langage de transformation ne permet pas l'identification automatique des règles à exécuter ainsi que ses ordonnancements d'appel. Pour cette raison, une fois que les règles de passage sont écrites en ATL nous les convertissons à un code Java afin de les exploiter pour effectuer la tâche de composition via les moteurs de passage, i.e. les déclencher implicitement. A partir du code Java généré pour chaque règle de passage, nous avons seulement besoin de manipuler la partie du code indiquée dans la Figure 4.10 afin d'invoquer la règle de passage correspondante.

```

PassageRuleName Runner = new PassageRuleName ();
Runner.loadModels(Refined-Functional-Model, Parameter-Model);
Runner.doAddImpType(new NullProgressMonitor());
Runner.saveModels(Architectural-Model);

```

FIGURE 4.10 – La règle *AddImpType* en code Java

La Figure 4.10 montre les méthodes transformation : (1) *loadModel* qui sert à importer les deux modèles de description (ex. le modèle fonctionnel) et de paramètre (*Source1*, *Source2*). (2) *doAdd-Imp-Type* qui sert à exécuter la transformation (3) *saveModel* qui sert à sauvegarder le modèle cible (*Target*) généré pour l'utiliser au cours d'autres transformations. Notre processus de composition vise à spécifier les règles à exécuter et les déclencher d'une manière implicite et automatique dans un ordre spécifique en respectant les contraintes d'exécution. Ce qui montre une bonne illustration du processus d'identification, de paramétrage et d'ordonnement d'invocation des règles de passage.

3 Résultats et Évaluation

Dans cette section, nous essayons de composer l'application mobile *ShopReview* présentée précédemment au début du chapitre de la conception (cf. chapitre ??, section ??) à l'aide du processus de composition proposé. Nous allons présenter dans un premier temps les résultats obtenus en se basant sur l'exemple de l'application *ShopReview*. Après, nous montrons un plan d'évaluation afin de tester l'efficacité de notre mécanisme de composition pour gérer les points d'hétérogénéité au cours de la composition.

3.1 Les résultats du processus de composition CAMAP

Tout d'abord, le concepteur de l'application doit se focaliser sur la définition du modèle fonctionnel de l'application mobile *ShopReview* en utilisant la palette développée. Il définit donc l'ensemble des fonctionnalités souhaitées et également les différentes dépendances entre elles. Le modèle fonctionnel obtenu est instancié du méta-modèle de description CMA-FD proposé. Le schéma de la collaboration de l'application *ShopReview* désirée sera comme illustré sur la Figure 4.11.

Afin d'orienter l'application *ShopReview* vers un environnement mobile spécifique, nous devons fournir les différentes entités concrètes contextuelles qui seront utilisées pour implémenter les fonctionnalités requises. Cependant, le concepteur doit fournir une description contextuelle pour toute les entités logicielles choisies pour implémenter les fonctionnalités souhaitées ainsi l'appareil mobile à utiliser suivant les modèles d'ontologies proposés.

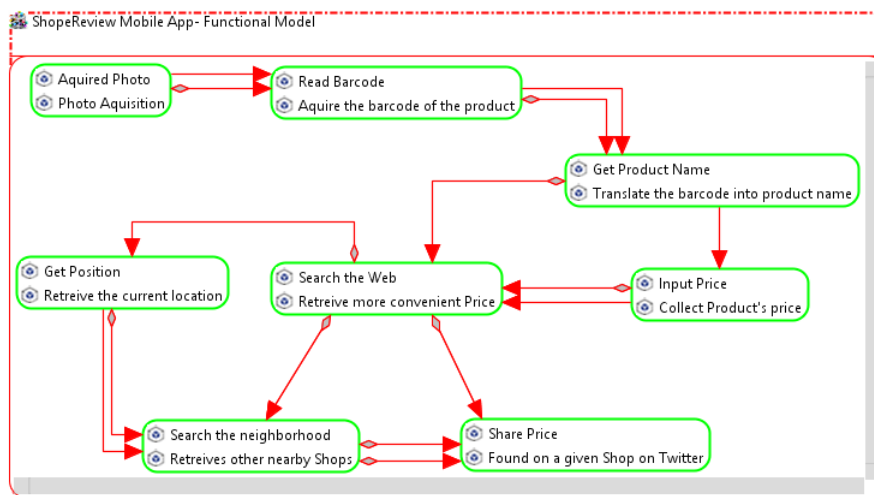


FIGURE 4.11 – Le modèle fonctionnel de l’application mobile *ShopReview*

Après avoir élaboré le modèle fonctionnel et défini les modèles d’ontologies appropriés, notre processus de composition vise à affecter les différents attributs définis dans le modèle de paramètres pour chaque fonctionnalité avec les informations suivantes : le type d’implémentation, les contraintes d’exécution, les données d’entrée et les données de sortie afin de les attacher par la suite aux fonctionnalités définies dans le modèle fonctionnel. Pour ce faire, le *moteur de transformation 1* déclenche implicitement l’exécution des différentes règles de passage qui sont nécessaires (ex. *ImpType*, *Add-Ex-Profile*, etc.) pour générer le modèle fonctionnel raffiné, i.e. implémentant l’algorithme illustré dans le Tableau 4.3.

Nous présentons deux scénarios de composition afin de montrer le potentiel de notre mécanisme à détecter les points d’hétérogénéité et de les gérer en attachant les médiateurs nécessaires.

- (1) Le premier *scénario de composition* consiste à associer les fonctionnalités identifiées par les entités concrètes contextuelles suivantes : Fun1, Fun3, Fun4, Fun5, Fun6, Fun7 ➡ Component CE ; et Fun2, Fun8 ➡ Service CE.
- (2) Dans le second *scénario de composition*, nous les attachons comme suit : Fun1, Fun2, Fun4, Fun6 ➡ Component CE ; et Fun3, Fun5, Fun7, Fun8 ➡ Service CE.

Par ces deux scénarios, nous n’avons pas illustré les valeurs réelles des différentes métriques indiquées dans le profil d’exécution des entités concrètes mais seulement leur différence relative. Les modèles fonctionnels raffinés obtenus sont illustrés dans la Figure 4.12(a) et la Figure 4.12(b) respectivement pour le *scénario de composition 1* et le *scénario de composition 2*.

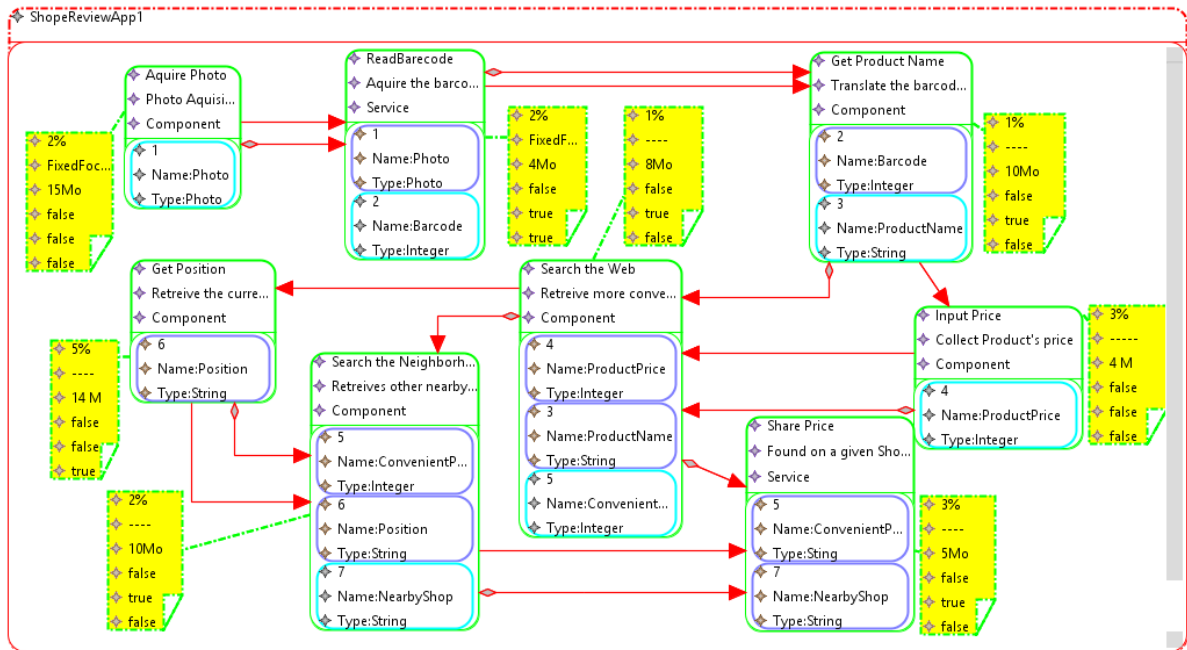
Après avoir généré la modèle fonctionnel raffiné de l’application *ShopReview*, le *moteur de transformation 2* déclenche implicitement l’exécution des règles de passage nécessaires qui sont définies dans un ordre spécifique pour obtenir le modèle architectural de l’application *ShopReview* tout en respectant les différentes contraintes de composition présentées précédemment.

A titre d’exemple, parmi les règles de passage introduites par le *moteur de transformation 2* nous présentons :

- (1) La règle *RefinedFun2ArchCMA* dont l’objectif est de remplacer chaque fonctionnalité par son entité concrète contextuelle correspondante ;

- (2) la règle *AddPrecedenceLink-Ex-Med* qui vise à ajouter un lien de précedence entre deux entités concrètes où ce lien est associé par un médiateur exogène afin d'indiquer et de gérer le point d'hétérogénéité présenté par cette relation de composition ;
- (3) la règle *AddPrecedenceLink* vise à ajouter juste un lien de précedence entre deux entités concrètes ce qui prouve qu'il n'y a pas un problème d'hétérogénéité en terme du type des entités (i.e. composition endogène) et que ces deux entités sont de même type.

(a) Résultat du Scénario 1



(b) Résultat du Scénario 2

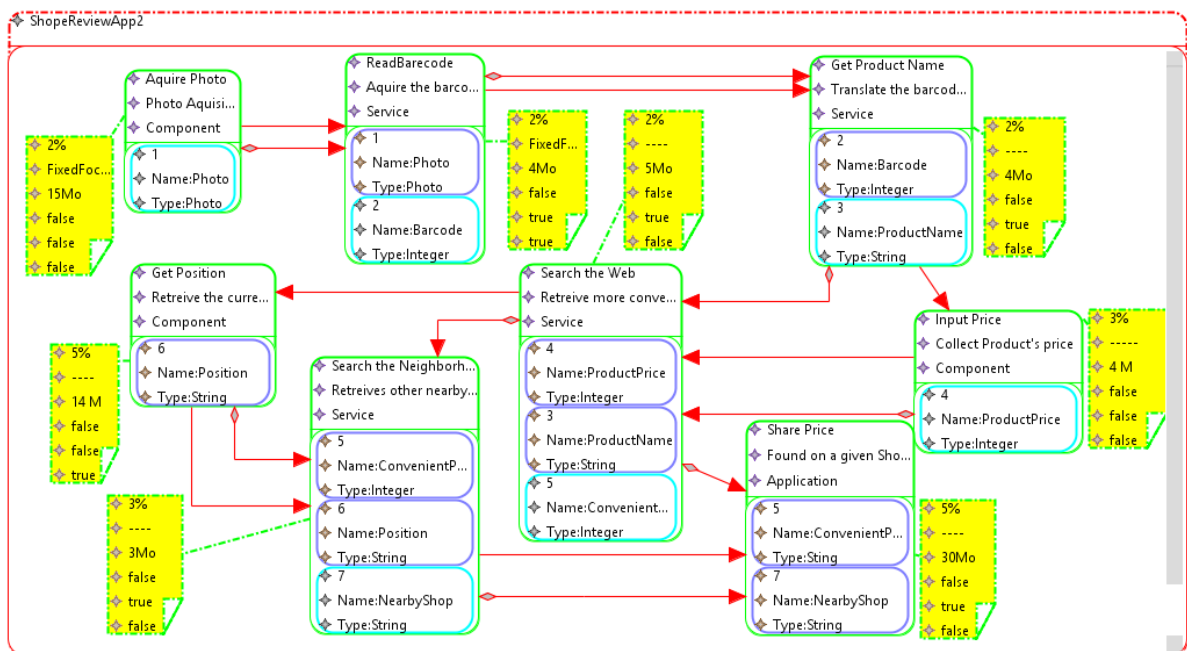


FIGURE 4.12 – Le modèle fonctionnel raffiné de l'application mobile *ShopReview*

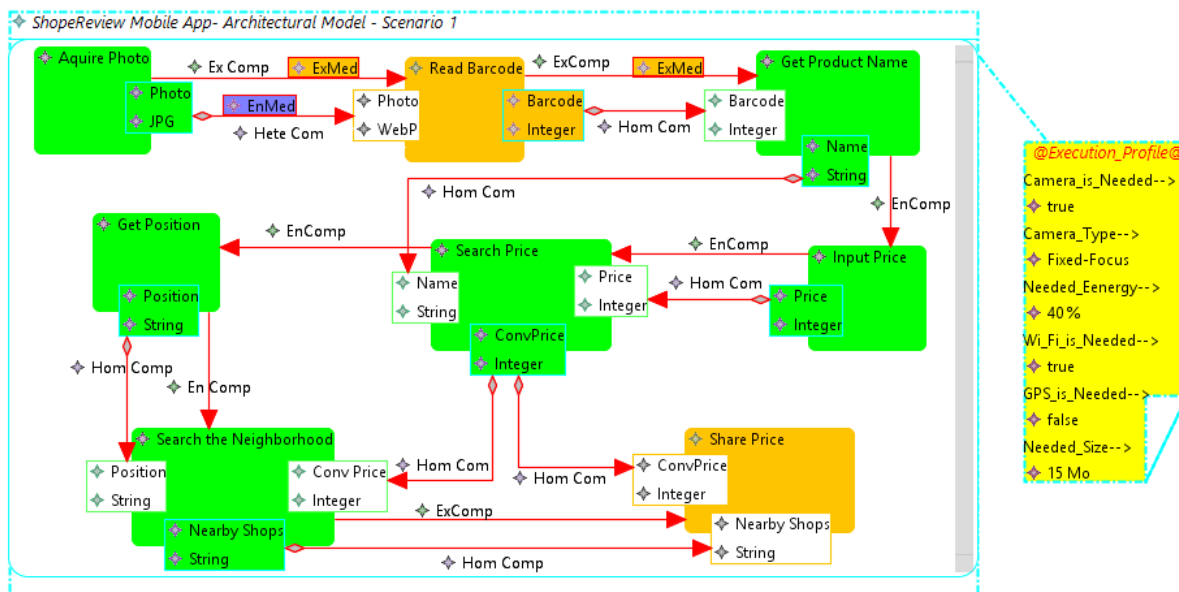
Les différentes règles de passage qui sont nécessaires pour obtenir le modèle architectural de l'application mobile *ShopReview* ainsi que leurs ordres d'exécution sont définis par le *moteur de transformation 2* tel qu'illustré dans le Tableau 4.8.

Tableau 4.8 – Les règles de passage déclenchées

Description de la règle	Règles déclenchées pour le scénario 1	Règles déclenchées pour le scénario 2
Remplacer les Funs par leurs SEs	<i>RefinedFun2ArchCMA</i>	<i>RefinedFun2ArchCMA</i>
<i>Gérer les points d'hétérogénéité qui portent sur les entités logicielles</i>		
Relier SE1 avec SE2	<i>AddPrecedenceLink-Ex-Med</i>	<i>AddPrecedenceLink</i>
Relier SE2 avec SE3	<i>AddPrecedenceLink-Ex-Med</i>	<i>AddPrecedenceLink-Ex-Med</i>
Relier SE3 avec SE4	<i>AddPrecedenceLink</i>	<i>AddPrecedenceLink-Ex-Med</i>
Relier SE4 avec SE5	<i>AddPrecedenceLink</i>	<i>AddPrecedenceLink-Ex-Med</i>
Relier SE5 avec SE6	<i>AddPrecedenceLink</i>	<i>AddPrecedenceLink-Ex-Med</i>
Relier SE6 avec SE7	<i>AddPrecedenceLink</i>	<i>AddPrecedenceLink-Ex-Med</i>
Relier SE7 avec SE8	<i>AddPrecedenceLink</i>	<i>AddPrecedenceLink-Ex-Med</i>
<i>Gérer les points d'hétérogénéités qui portent sur les données échangées</i>		
.....
Générer Le profil d'exécution (CMA)	<i>Add-Execution-Profile</i>	<i>Add-Execution-Profile</i>

Le *moteur de transformation 2* gère aussi les points d'hétérogénéité qui portent sur les données échangées, à savoir le déclenchement des règles de transformation qui servent à attacher les liens d'utilisation avec les médiateurs endogènes si les données échangées ne sont pas compatibles (i.e. *AddUseLink-En-Med*). Par conséquent, il nous fournit l'architecture de la CMA souhaitée y compris les adaptateurs nécessaires. La Figure 4.13 (a) et la Figure 4.13 (b) montrent les résultats de cette tâche respectivement pour les scénarios de composition 1 et 2.

(a) Résultat du Scénario 1



(b) Résultat du Scénario 2

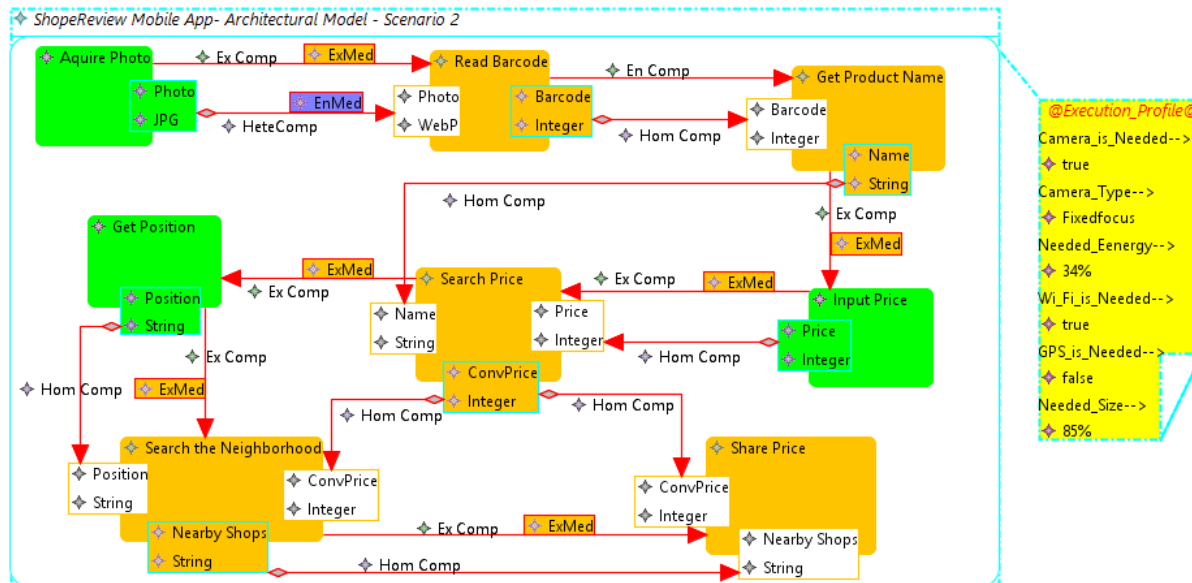
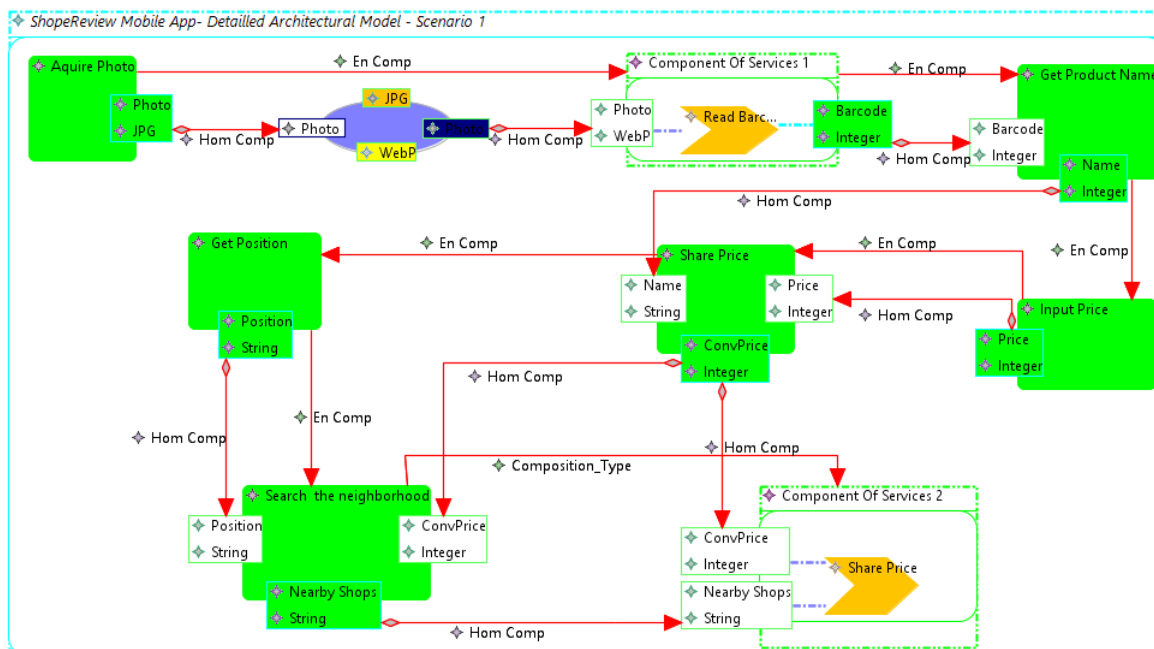


FIGURE 4.13 – Le modèle architectural de l’application mobile ShopReview

Le *moteur de transformation 3* vise à raffiner les différents modèles obtenus en spécifiant les tâches des différents médiateurs intégrés selon les différentes contraintes d’exécution proposées. La Figure 4.14(a) et la Figure 4.14(b) montrent les résultats de transformation respectivement pour les scénarios de composition 1 et 2.

(a) Résultat du Scénario 1



(b) Résultat du Scénario 2

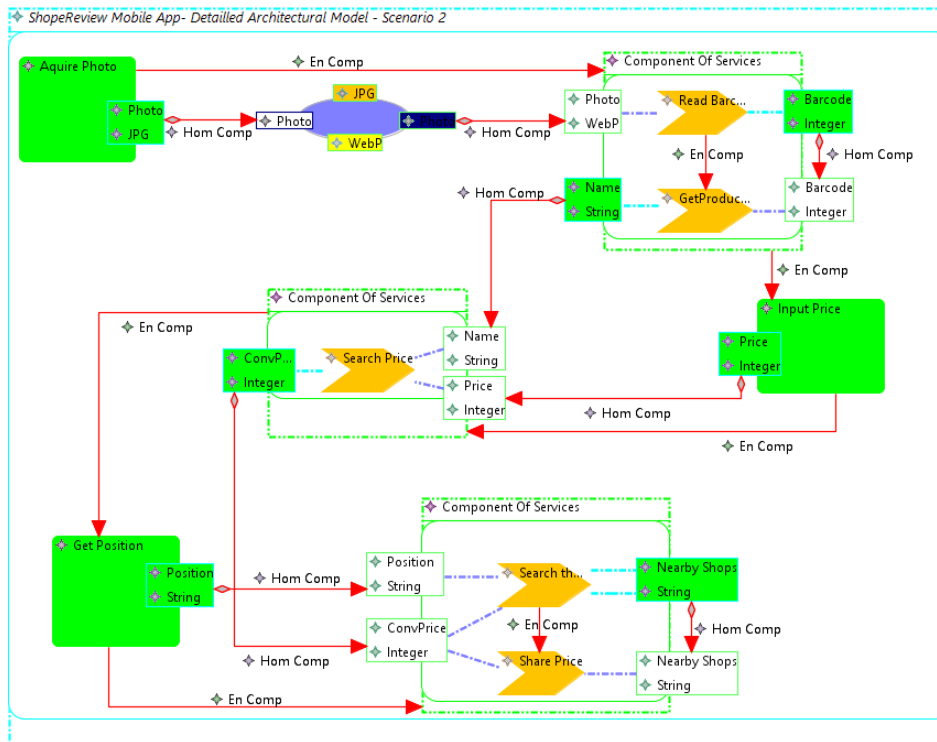


FIGURE 4.14 – L’architecture détaillée de l’application mobile *ShopReview*

3.2 L’évaluation du processus de composition CAMAP

Afin d’évaluer le processus proposé nous avons adopté le simulateur OMNeT ++ pour calculer le temps de passage du modèle fonctionnel au modèle architectural dont l’objectif est de montrer la capacité de notre processus à déterminer les règles de passage nécessaires et à déclencher implicitement ces règles identifiées. Ces dernières visent à composer l’application souhaitée tout en détectant les points d’hétérogénéité et en associant les différents médiateurs nécessaires. La Figure suivante montre la topologie de l’application *ShopReview* élaborée par le simulateur :

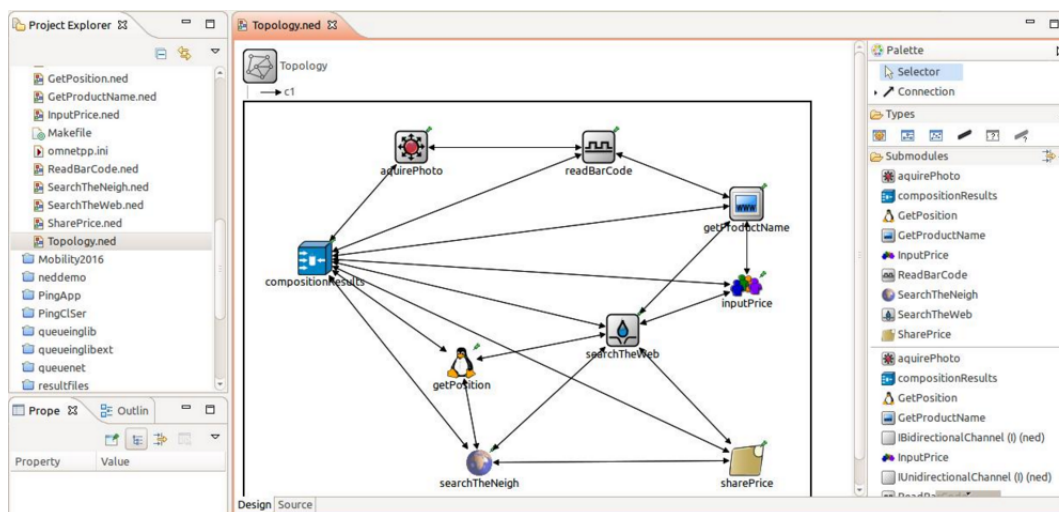


FIGURE 4.15 – Topologie de l’application *ShopReview* sur la plateforme OMNET++

Le Graph1 présenté dans la Figure 4.16(a) illustre la séquence des différentes règles de passage appelées pour obtenir l'application mobile composite en utilisant les valeurs des paramètres contextuels présentés par les deux scénarios 1 et 2 ainsi que leur temps d'exécution. Pour le scénario de composition 1, notre processus de composition a déclenché trois règles de transformation de type *AddPrecedenceLink* avec des médiateurs exogènes et quatre sans médiateurs. Contrairement au scénario de composition 2 qui présente six points d'hétérogénéité. Les règles de passage numéro 3, 4, 5, 6, 7 représentent les points de divergence entre les deux courbes ce qui montre à la fois que les deux scénarios compositions ne présentent pas les mêmes types de composition entre les fonctionnalités. Notre processus de composition détecte une combinaison hétérogène entre la Fun2 et la Fun3 pour le scénario de composition 1 (composer une entité de type service avec une autre de type composant tout en appelant la règle *AddPrecedenceLink-Ex-Med* : 0.04 sec) alors que cette combinaison est homogène pour le scénario de composition 2 (composer une entité de type service avec une autre du même type tout en appelant la règle *AddPrecedenceLink* : 0,02 sec). En outre, notre processus détecte d'autres combinaisons hétérogènes pour le scénario de composition 2 qui sont homogènes pour le scénario de composition 1. Par conséquent, nous constatons que le nombre de règles appelées est équivalent pour les deux scénarios de composition ce qui prouve que « Le nombre de règles de passage à exécuter dépend du nombre de fonctionnalités identifiées » tandis que le choix de leur type dépend de la notion d'hétérogénéité.

Le deuxième graphe présenté dans la Figure 4.16(b) illustre le temps de passage du modèle fonctionnel raffiné au modèle architectural de la CMA souhaitée. Il représente le temps d'exécution des deux scénarios de composition en terme de nombre des règles appelées où :

$$\text{Temps du passage} = \sum_{i=0}^k (\text{Temps d'exécution de la règle}_i)$$

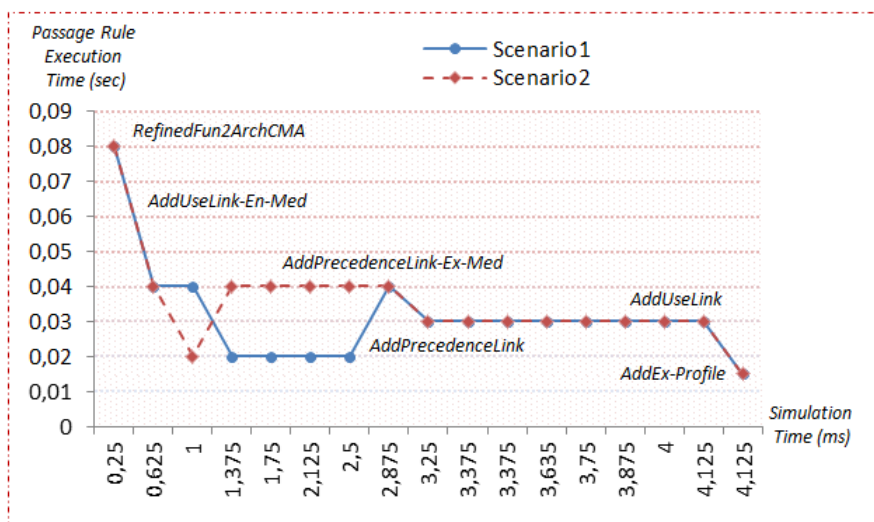
$$\text{Temps d'exécution de la règle RefinedFun2ArchCMA} = \text{Temps de remplacement} \times \text{Nb Fun}$$

$$\text{Temps d'exécution de la règle Add-PL} = \sum_{i=0}^n (\text{AddPrecedenceLinkExMed}_i \text{ ou } \text{AddPrecedenceLink}_i)$$

$$\text{Temps d'exécution de la règle Add-UL} = \sum_{i=0}^m (\text{AddUseLinkEnMed}_i \text{ ou } \text{AddUseLink}_i)$$

k=Nombre de règles de passage, n= Nombre de liens de précedence, m= Nombre de liens d'utilisation, Replaced Time= 0.01 sec (il représente le temps nécessaire pour remplacer chaque fonctionnalité par son entité concrète appropriée).

(a) Graph1 : Temps d'exécution des règles de passage



(b) Graph2 : Temps de passage global

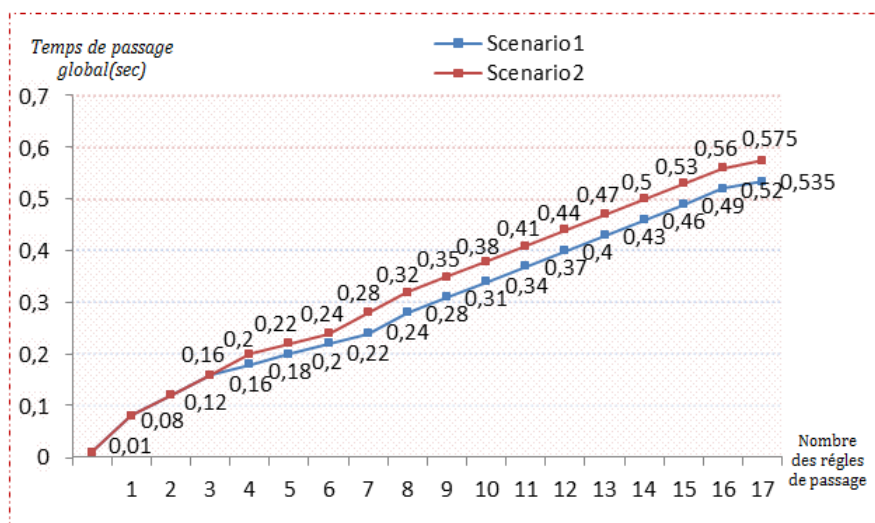


FIGURE 4.16 – Un plan d’évaluation pour le processus de composition proposé

En se basant sur les résultats obtenus par le Graph1, nous constatons que le *scénario de composition 1* présente trois points d’hétérogénéité tandis que le deuxième scénario présente six points d’hétérogénéité ce qui conduit à augmenter le temps d’exécution de la tâche de composition dans le *scénario 2* à partir du premier point de divergence, comme illustré sur la Figure 4.16 (b)(0,16 sec). Ce résultat montre que le temps de passage est fortement corrélé avec les types de composition entre les entités où « Plus les points d’hétérogénéité augmentent, plus le temps de composition augmentera ». Par conséquent, notre processus de composition a la possibilité pour détecter les combinaisons hétérogènes. Ainsi, il identifie et déclenche implicitement les règles de passage appropriées en fonction des points d’hétérogénéité trouvés. Par conséquent, il a la capacité de composer une application mobile hétérogène qui est adaptative en temps très proche du temps de passage optimal (0,535 sec pour le *scénario 1*, 0,575 sec pour *scénario 2*). Cela revient au fait que le nombre des règles à exécuter est toujours identique pour la composition de la même application dans plusieurs scénarios de composition mais le type des règles à exécuter qui se diffère selon les types des combinaisons entre les entités. Pour cette raison le temps de passage s’augmente un peu plus lors l’exécution des règles qui traitent les points d’hétérogénéité.

CONCLUSION

Dans ce chapitre, nous avons donné les détails et les choix techniques pour mettre en œuvre un prototype destiné à supporter le mécanisme de composition proposé. Nous avons présenté dans un premier temps l'implémentation des différents méta-modèles proposés pour représenter l'application mobile composite à plusieurs niveaux d'abstraction. Pour ce faire, nous avons adopté la technologie EMF pour élaborer les méta-modèles et la technologie GMF pour obtenir une représentation graphique pour chaque instance de ces méta-modèles.

Nous avons discuté de l'implémentation de notre processus de composition à savoir le passage entre les différents modèles proposés. Nous avons aussi présenté les différents algorithmes de passage. Notre choix technologique pour réaliser le processus de composition proposé était l'usage du langage de transformation ATL et de la technologie Java. A ce stade, nous avons proposé d'utiliser un modèle de paramètres pour adjoindre la notion de généricité aux différentes règles écrites en ATL et donc pouvoir manipuler les valeurs des paramètres contextuels selon le cas traité. Dû au fait que notre processus de composition nécessite un déclenchement implicite des règles des passages dans un ordre variable selon les types de composition traités, nous avons transformé les règles ATL génériques en code Java.

Dans une dernière section dans ce chapitre, nous avons présenté les résultats obtenus pour construire l'application *ShopReview* après avoir clairement posé les scénarios considérés. A ce stade, nous avons présenté la modélisation de l'application mobile composite *ShopReview* suivant deux scénarios de composition à travers les modèles proposés, à savoir son implémentation via notre processus de composition.

Afin de prouver l'efficacité de notre processus, nous avons élaboré un plan d'évaluation concernant la composition de l'application *ShopReview*. A cet effet, nous avons simulé le temps de passage entre les différents modèles de la CMA dont l'objectif est de montrer la capacité de notre processus à gérer la composition hétérogène des applications mobiles adaptatives.

CHAPITRE : CONCLUSION GÉNÉRALE

1. CONTRIBUTION
2. PERSPECTIVES

Actuellement, l'apparition croissante et l'adoption massive sur l'utilisation des Smartphones et tablettes ainsi l'affleurement grandissant des systèmes d'exploitation conduisent à la richesse du domaine mobile. Cette richesse est difficile à prendre en compte pour les entreprises de développement mobile dans la création des applications mobiles à cause de l'hétérogénéité matérielles et logicielles présentés par les appareils mobiles. Dû au fait que les appareils mobiles sont caractérisés par des ressources limitées, la composition des applications mobiles soumis à plusieurs contraintes de l'environnement d'exécution. Cependant, afin d'obtenir des applications mobiles adaptatives, il est nécessaire de faire face aux paramètres et aux ressources hétérogènes offertes par les appareils mobiles.

En plus, le nombre ainsi que les différents types d'applications qui sont disponibles sur une variété de plateformes pour les appareils mobiles sont indéniablement explosifs. Ce phénomène est susceptible d'encourager l'utilisateur d'un appareil mobile d'attendre de nouvelles applications selon ses propres besoins en se basant sur celles qui sont déjà existantes où la même application obtenue soit disponible sur à peu près tout type de plateformes même quand il y a des différences entre les plateformes ainsi que dans les applications qui fonctionnent sur ces plateformes- à savoir personnaliser le comportement de l'application souhaitée selon les différents environnements d'exécution et les besoins des utilisateurs. Afin de mieux répondre aux attentes de l'utilisateur, la compréhension de ces différences - ou hétérogénéités - devient alors critique, tout comme les résoudre, mais il est assez difficile, à savoir sa résolution devient difficile.

1 Contribution

Dans ce travail de recherche, nous avons proposé un processus de composition pour construire des applications mobiles adaptables à leur contexte d'utilisation en ignorant les détails d'implémentation des entités logicielles constituantes. L'objectif est d'élargir l'innovation dans le monde mobile tout en comblant les besoins des utilisateurs et assurant le déploiement correct et le fonctionnement propre des applications mobiles composites obtenues sur les appareils mobiles choisis.

Pour cela, nous visons également à personnaliser le comportement de l'application mobile souhaitée selon les différentes informations contextuelles. A cet effet, un bon choix des entités logicielles constituantes dépend des informations contextuelles du dispositif mobile sur lequel elle s'exécute. Cette technique permet de résoudre les différences qui peuvent être produites à force des besoins des utilisateurs et des informations contextuelles de l'environnement d'exécution en détectant les différents points d'hétérogénéité et en associant les médiateurs appropriés lorsqu'une coordination hétérogène est nécessaire. Pour ces fins, nous avons suivi une approche de modélisation. Nous avons proposé des langages de description pour définir l'application mobile souhaitée à plusieurs niveaux abstraites en mettant en œuvre le passage entre eux en utilisant des mécanismes de transformation. Un outillage soutenant le processus de composition proposé est mis en œuvre et un exemple pour montrer son applicabilité et son efficacité est présenté.

Dans ce manuscrit, nous avons pris en compte les problèmes d'hétérogénéité qui sont produites par les objets de composition et les environnements d'exécution lors de la tâche de composition d'applications mobiles par le traitement de deux sous-problèmes : la description contextuelle de l'environnement de composition et la représentation architecturale de l'application mobile composite.

Une caractéristique importante de notre démarche est l'adoption d'une approche de modélisation afin de pouvoir comprendre et représenter les différences ainsi que les points communs entre les différentes entités logicielles à composer en considérant l'hétérogénéité

des appareils mobiles. Pour réaliser ce mécanisme de composition des langages de description sont utilisés pour définir l'application mobile souhaitée à différents niveaux d'abstraction où les différents passages entre eux sont établis à l'aide des mécanismes de transformation. Par conséquent, le processus de composition proposé permet de faire face et gérer les issues de l'hétérogénéité (composition hétérogène) qui peuvent être produites à force des exigences de l'utilisateur et les informations contextuelles de l'environnement d'exécution où il vise à détecter les différents points d'hétérogénéité et donc associer les médiateurs appropriés lorsqu'une coordination hétérogène est nécessaire.

Afin de pouvoir composer des applications mobiles adaptatives la compréhension et la détermination des caractéristiques contextuelles de l'environnement d'exécution deviennent alors des points importants. En outre, plusieurs entités logicielles peuvent être utilisées pour implémenter les exigences définies pour une telle application mobile souhaitée où chacun d'entre eux opère selon des conditions bien déterminées. A cet effet, le développeur de l'application doit tenir compte de ces contraintes d'exécution lors de la tâche de composition afin d'utiliser les entités logicielles qui correspondent aux besoins identifiés et qui s'adaptent le mieux avec le contexte d'exécution choisi.

Pour ce faire, nous avons proposé des modèles descriptifs qui adoptent la notation d'ontologies pour définir le contexte de composition d'une application mobile. Nous avons présenté deux descriptions à base d'ontologies : la première vise à décrire le contexte de l'environnement d'exécution qui représente la situation actuelle de l'appareil mobile tandis que la seconde correspond à la description contextuelle des objets de la composition. Par le premier modèle descriptif, nous avons traité le contexte de l'appareil mobile grâce à deux dimensions : un axe statique qui est représenté par les fonctionnalités matérielles et logicielles de l'appareil mobile et un autre dynamique qui est spécifié par l'état actuel des ressources existantes sur cet appareil alors que le deuxième modèle descriptif vise à définir l'aspect fonctionnel des entités logicielles ainsi que l'aspect non fonctionnel à savoir ses conditions d'exécution.

Le processus proposé permet de traiter plusieurs types de composition en terme de collaborations - hétérogènes/homogènes - effectuées. Donc, il couvre plusieurs points d'hétérogénéité que ce soit en termes d'objets de composition ou d'environnement d'exécution. Par conséquent, il permet la composition des applications mobiles adaptatives en utilisant des entités logicielles hétérogènes en commençant par la définition des différentes fonctionnalités souhaitées jusqu'à l'obtention d'un modèle architectural détaillé pour l'application composite incluant tous les adaptateurs nécessaires. Le processus de composition proposé permet aussi de donner la possibilité d'obtenir plusieurs versions pour la même application mobile composite selon les différentes informations contextuelles de l'appareil mobile sur lequel nous voulons déployer l'application mobile désirée.

Nous avons évalué notre processus par une petite étude empirique en introduisant des scénarios de composition pour l'application mobile *ShopReview*. Le plan d'évaluation élaboré a révélé une certaine capacité et potentiel de notre processus de composition à découvrir et à gérer les points d'hétérogénéité posés lors de la composition où nous avons adopté le simulateur OMNET++ pour effectuer cette évaluation en calculant le temps de passage du modèle fonctionnel au modèle architectural de l'application mobile composite.

2 Perspectives

Dû au fait que la réalisation du processus de composition global proposé dépasse le cadre d'une seule thèse et traite plusieurs axes de recherches que nous considérons larges en quelque sorte (la découverte, la sélection, la génération du code. etc.), nous nous sommes concentré en particulier que sur la tâche de composition. Cependant, nous envisageons de traiter les

étapes restantes dans nos futurs travaux pour accomplir la réalisation du processus global proposé. Les paragraphes qui suivent font une présentation succincte sur les perspectives de notre travail.

Nous avons constaté que le mécanisme de composition abordé a le potentiel pour déployer et migrer la même application mobile composite à travers différentes plateformes mobiles. En se basant sur les différents mécanismes présentés par l'ingénierie dirigée par les modèles en particulier via le mécanisme de transformation de type modèle vers code, nous envisageons prochainement à générer l'application composite concrète (i.e. le code de l'application) vers une plateforme spécifique en utilisant le modèle architectural détaillé qui est obtenu par le processus de composition présenté dans ce manuscrit.

En ce qui concerne les applications composites hétérogènes nous avons introduit la structure conceptuelle pour les différents médiateurs proposés pour éliminer cette hétérogénéité et donc permettre une composition consistante. Cependant, nous visons aussi à implémenter ces différents médiateurs proposés. Pour ce faire, nous devons réaliser les médiateurs exogènes en fournissant la structure concrète de la nouvelle entité *Component-of-services* ainsi que les médiateurs endogènes en fournissant la structure concrète au service de médiation proposé. Ce dernier nécessite de construire une bibliothèque de services de médiation selon les transformations requises des données échangées afin d'appeler par la suite celui qui est nécessaire pour assurer la compatibilité des données transmises.

Pour faire face à l'hétérogénéité des appareils mobiles, nous avons proposé de composer des entités logicielles adaptatives où nous avons supposé que l'ensemble de ces briques logicielles contextuelles sont déjà existantes. A cet effet, un autre point essentiel sur lequel nous devons mettre l'accent est d'automatiser la réalisation de la tâche de sélection. Dans le travail actuel nous avons proposé des notations d'ontologies pour décrire le contexte de composition afin de pouvoir choisir et assurer la conformité des briques constituantes avec l'environnement d'exécution. En se basant sur ses modèles d'ontologies proposés, nous visons comme futur travail à implémenter un algorithme de sélection dirigé par le contexte qui vise à sélectionner des entités logicielles adaptatives parmi tous ceux qui correspondent aux besoins identifiés, i.e. construire une telle application mobile vers un contexte mobile spécifique. En outre, il vise à déterminer les différents chemins possibles de composition pour construire des applications mobiles personnalisées.

Les perspectives possibles d'amélioration du processus de composition présenté dans ce manuscrit touchent l'aspect de réalisation où nous envisageons à fournir aussi un moyen simple et lisible basé sur les interfaces graphiques pour réaliser la tâche de composition à savoir une plateforme de composition graphique. Cette dernière fait apparaître dans un premier temps la palette proposée qui permet de dessiner le modèle fonctionnel. Après, elle nous donne la possibilité de déclencher le processus de composition, i.e. la génération des modèles raffinés, par le biais des boutons graphiques.

PRODUCTION SCIENTIFIQUE ET PROJETS DE RECHERCHE

- REVUES INTERNATIONALES AVEC COMITÉ DE LECTURE

- (1) Djeddar A., Bendjenna H., Amirat A., Roose P. and Chung L.. Context-Driven Composition for Mobile Applications : A Metamodeling Approach. International journal of Embedded Systems (IJES), 2016.
- (2) Djeddar A., Bendjenna H., Amirat A. and Amer Alwan A. "Developing Context-aware Mobile Applications Using Composition Process based-on heterogeneous software entities. Journal of Advanced Computer Science and Technology Research, Vol.5 No.3, June 2015, 93-103.

- CONFÉRENCES INTERNATIONALES AVEC COMITÉ DE LECTURE

- (1) Djeddar A., Bendjenna H., Amirat A. and Roose P. "Selection algorithm of contextual software entities for composing mobile applications". IEEE International Conference on Information Technology for Organizations Development, Fez Morocco. March 30-April 05, 2016 .
- (2) Djeddar A., Bendjenna H., Amirat A. and Roose P. "Metamodeling Approach for Composing Mobile Applications". The International Conference on Pattern Analysis and Intelligent Systems (PAIS'2015). University of Larbi Tebessi, Algeria. October 26-27, 2015.
- (3) Djeddar A., Bendjenna H., Amirat A. and Oussalah M. "Composition Process Based-on Heterogeneous Software Entities For Mobile Applications". International Conference on Advanced Computer Science Applications and Technologies (ACSAT'2014). University of Philadelphia, Amman-Jordan. December 29-30, 2014.
- (4) Amirat A., Djeddar A. and Oussalah M. "Evolving and Versioning Software Architectures Using ATL Transformations". The International Arab Conference on Information Technology (ACIT'2014). University of Nizwa, Oman. December 9-11, 2014.

- WORKSHOP ET JOURNÉES DOCTORALES AVEC COMITÉ DE LECTURE

- (1) Djeddar A., Bendjenna H., Amirat A. and Oussalah M. "New approach for smart composition of mobile applications". The Symposium on Complex Systems and Intelligent Computing (CompSIC'2015). Université de Souk Ahrad, Algérie.
- (2) Djeddar A., Bendjenna H., Amirat A. and Oussalah M. "Composing Mobile Applications- Metamodeling Approach". Journées Ouvertes sur les Mathématiques et l'Informatique (JOMI'2015). Université de Larbi Tebessi, Algérie.
- (3) Djeddar A., Bendjenna H. and Amirat A. "Building adaptive Mobile Applications Using Composition Mechanism". Journées Doctorales en Informatique (JDI'2014). Université de Guelma, Algérie.

- PROJETS DE RECHERCHE

- (1) Membre dans un projet de recherche CNEPRU intitulé : "Processus de composition des applications mobiles hétérogènes". Projet B*02920140028 sous la responsabilité du Dr.Hakim Bendjenna. Université de Larbi Tebessi, Laboratoire LAMIS, 2015/2019.

BIBLIOGRAPHIE

Abi Lahoud, E. (2010). Composition dynamique de services : application à la conception et au développement de systèmes d'information dans un environnement distribué, Thèse de doctorat en informatique. Dijon : Université de BOURGOGNE, 11/02/2010. p. 235.

Abi-Char, P. E. et al. (2010). Privacy and Trust Issues in Context-Aware Pervasive Computing : State-of-the-Art and Future Directions, In : Zhang yan, Trust Modeling and Management in Digital Environments : From Social Concept to System Development. Published in the united state of america by IGI Global. p. 352-377, ISBN : 1615206833.

Abowd, G. D. et al. (1999). Towards a better understanding of context and context-awareness. In : Gellersen Hans-W, Handheld and ubiquitous computing, Springer Berlin Heidelberg. p. 304-307. ISBN : 3540665501.

Allen, R. J. et al. (1998). Formal modeling and analysis of the HLA component integration standard. In ACM SIGSOFT Software Engineering Notes, ACM. Vol. 23, No. 6, p. 70-79. ISBN : 1581131089.

Alonso, G. et al. (2004). Web services : Concepts, Architectures and Applications, Springer Verlag. p. 320 . ISBN : 3642078885.

Amirat, A. et al. (2014). Object-Oriented, Component-Based, Agent-Oriented and Service-Oriented Paradigms in Software Architectures. In : Oussalah, M. C., Software Architecture 1, Wiley Online Library. p. 1-53. ISBN : 1118930967.

AppBrain (2015). Most popular Google Play categories. (en ligne). In AppBrain : Android Statistics, Disponible sur : <http://www.appbrain.com/stats/android-market-app-categories>. Consulté le (12 mars 2015).

Apple, S. (2015). Profils Bluetooth pris en charge. (en ligne). Apple Inc, Disponible sur : <https://support.apple.com/fr-fr/HT204387>. Consulté le (25/04/2015).

Ay, F. (2007). Context modeling and reasoning using ontologies. (en ligne). University of Technology Berlin. Disponible sur : <http://www.ponnuki.de/cmaruo/cmaruo.pdf>.

Balasubramanian, K. et al. (2006). "Applying model-driven development to distributed real-time and embedded avionics systems." International Journal of Embedded Systems 2(3-4) : 142-155.

Barais, O. (2005). Construire et Maitriser l'évolution d'une architecture logicielle à base de composants, Thèse de doctorat en informatique. Lille 1 : l'université des Sciences et Technologies, 29 novembre 2005. 247 p.

Barros, A. et al. (2006). Standards for web service choreography and orchestration : Status and perspectives. In Business process management workshops, Springer. p. 61-74. ISBN : 3540325956.

Beauche, S. and P. Poizat (2008). Automated service composition with adaptive planning. In : F. George, L. Winfried, Service-Oriented Computing-ICSOC 2008, Springer. p. 530-537. ISBN : 9783642012471.

Beisiegel, M. et al. (2005). Service Component Architecture : Building Systems Using a Service Oriented Architecture. (en ligne) Whitepaper . Vol. 1, 31 p. Disponible sur : <http://www.iona.com/devcenter/sca/SCAWhitePaper109.pdf>.

-
- Benatallah, B., et al. (2005). *Service Composition : Concepts, Techniques*. In : Z. Stojanovic and A. Dahanayake, *Service-Oriented Software System Engineering : Challenges and Practices*, Idea Group. p. 48-66. ISBN : 1-59140-428-2.
- Benmammar, B. and F. Krief (2003). *La technologie agent et les réseaux sans fil* : HAL - CCSD - CNRS, 28 November 2004. p. 20. <hal-00003383>.
- Bettini, C. et al. (2010). "A survey of context modelling and reasoning techniques." *Pervasive and Mobile Computing*, Elsevier 6(2) : 161-180. ISSN : 1574-1192.
- Biermann, E. et al. (2006). *Graphical definition of in-place transformations in the eclipse modeling framework*. In : O. Nierstrasz, J. Whittle, D. Harel and G. Reggio, *Model Driven Engineering Languages and Systems*, Springer. p. 425-439. ISBN : 3540457720.
- Blanc, X. and O. Salvatori (2011). *MDA en action : Ingénierie logicielle guidée par les modèles*, Editions Eyrolles, Paris. 292 p. ISBN : 2212144148.
- Brel, C. (2013). *Composition d'applications multi-modèles dirigée par la composition des interfaces graphiques*, Thèse de doctorat en informatique. Antipolis : Université Nice Sophia, 28 juin 2013. 204 p.
- Brown, A. W. and K. C. Wallnau (1998). "The current state of CBSE." *IEEE software*, IEEE 15(5) : 37-46. ISSN : 0740-7459.
- Brown, P. J. (1995). "The stick-e document : a framework for creating context-aware applications." *ELECTRONIC PUBLISHING-CHICHESTER-*, Citeseer 8(2-3) : 259-272. ISSN : 0894-3982.
- Brown, P. J. (1998). "Triggering information by context." *Personal Technologies*, Springer 2(1) : 18-27. ISSN : 0949-2054.
- Brown, P. J. et al. (1997). "Context-aware applications : from the laboratory to the marketplace." *Personal Communications*, IEEE 4(5) : 58-64. ISSN : 1070-9916.
- Bruneton, E. et al. (2006). "The fractal component model and its support in java." *Software-Practice and Experience*, Wiley Interscience, London, New York 36(11) : 1257-1284. ISSN : 0038-0644.
- Buchholz, S. et al. (2004). *Comprehensive structured context profiles (cscp) : Design and experiences*. In *Pervasive Computing and Communications Workshops, Proceedings of the Second IEEE Annual Conference on*, IEEE. p. 43-47. ISBN : 0769521061.
- Cai, X. et al. (2000). *Component-based software engineering : technologies, development frameworks, and quality assurance schemes*. In *Software Engineering Conference, APSEC 2000 Proceedings. Seventh Asia-Pacific*, IEEE. p. 372-379. ISBN : 0769509150.
- Cariou, E. et al. (2010). *OCL contracts for the verification of model transformations*. In *Proceedings of the Workshop The Pragmatics of OCL and Other Textual Specification Languages at MoDELS 2009*. *Electronic Communications of the EASST*, vol. 24. ISBN : 1863-2122.
- Cariou, E. et al. (2004). *OCL for the specification of model transformation contracts*. In *OCL and Model Driven Engineering, UML 2004 Conference Workshop*, Vol. 12. p. 69-83.
- Cariou, N. et al. (2009). "Contrats de transformation pour la validation de raffinement de modèles." *IDM 2009 Actes des 5emes journées sur l'ingénierie Dirigée par les Modèles*, Citeseer 1501(9) : 1-16.
- Cavallaro, L. et al. (2009). *An automatic approach to enable replacement of conversational services*. In : L. Baresi, C. Chi and J. Suzuki, *Service-Oriented Computing*, Springer. p. 159-174. ISBN : 978-3-642-10382-7.
-

-
- Chakraborty, D. et al. (2005). "Service composition for mobile environments." *Mobile Networks and Applications*, Springer-Verlag, New York, Inc. 10(4) : 435-451. ISSN : 1383-469X.
- Chang, Y.-C. et al. (2008). Solving the service composition puzzle. In *Proceedings of the 2008 IEEE International Conference on Services Computing (SCC 2008)*. IEEE Computer Society : Washington, DC, USA. Vol. 2. p. 387-394. ISBN : 0769532837.
- Chen, G. and D. Kotz (2000). A survey of context-aware mobile computing research, Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College. Vol. 1, No. 2.1, pp. 2-1.
- Cheverst, K. et al. (2000). Providing tailored (context-aware) information to city visitors. In *Adaptive hypermedia and adaptive web-based systems*, Springer. p. 73-85. ISBN : 3540679103.
- Christensen, E. et al. (2001). Web services description language (WSDL) 1.1. (en ligne). Disponible sur : [Http://www.w3.org/TR/wsdl](http://www.w3.org/TR/wsdl).
- Combemale, B. (2008). "Ingénierie Dirigée par les Modèles (IDM) état de l'art." *Management* : p. 1-19. Submitted on 29 Mar 2009, «hal-00371565» Disponible sur : <http://hal.archives-ouvertes.fr/hal-00371565/en/>.
- Compuware (2012). *Mobile Apps : What Consumers Really Need and Want in The Technology Performance*. (en ligne). In : *A Global Study of Consumers' Expectations and Experiences of Mobile Applications*, Company Compuware, p. 8. Disponible sur : <https://info.dynatrace.com/rs/compuware/images/Mobile-App-Survey-Report.pdf> : p. 8.
- Crnkovic, I. et al. (2006). "Component-based development process and component lifecycle." *Journal of Computing and Information Technology* 13(4) : 321-327.
- Crnkovic, I. et al. (2007). A classification framework for component models. In *Proceedings of the 7th Conference on Software Engineering Research and Practice (SERPS 2007)*, Sweden, p. 3-12. Disponible sur : <http://crosbi.znanstvenici.hr/prikazi-rad?lang=EN&rad=393766>.
- Cugola, G. et al. (2014). "Selfmotion : A declarative approach for adaptive service-oriented mobile applications." *Journal of Systems and Software*, Elsevier 92(0) : 32-44. ISSN : 0164-1212. Disponible sur : <http://www.sciencedirect.com/science/article/pii/S0164121213002653>.
- Czarnecki, K. and S. Helsen (2003). Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, USA. Vol. 45, No. 3, p. 1-17.
- Daniel, F. and B. Pernici (2006). "Insights into web service orchestration and choreography." *International Journal of E-Business Research* 2(1) : 58-77. ISSN : 1548-1131.
- Dashofy, E. M. and A. Van Der Hoek (2002). Representing product family architectures in an extensible architecture description language. In : Frank J. van der Linden, *Software Product-Family Engineering*, Springer : p. 330-341. ISBN : 978-3-540-21941-5.
- Derdour, M. (2012). *Modélisation et implémentation d'un système d'information de gestion de flux multimedia pour des architectures logicielles intégrant des appareils sans-fil mobiles*. Thèse de doctorat en Science : Réseaux., Annaba : Université Badji Mokhtar. 220 p.
- Derdour, M. et al. (2010a). "Typing of adaptation connectors in MMSA approach case study : sending MMS." *International Journal of Research and Reviews in Computer Science* 1(4) : 39-49. ISSN : 2079-2557.
- Derdour, M. et al. (2010b). "MMSA : metamodel multimedia software architecture." *Advances in Multimedia*. Hindawi Publishing Corporation, Article ID 386035, 17 pages, 2010. doi :10.1155/2010/386035.
- Desertot, M. et al. (2006). FROGi : Fractal components deployment over OSGi. In : W.
-

Löwe and M. Südholt, *Software Composition*, Springer Berlin Heidelberg. p. 275-290. ISBN : 978-3-540-37657-6.

Dey, A. K. (2001). "Understanding and using context." *Personal and ubiquitous computing*, Springer-Verlag 5(1) : 4-7. ISSN : 1617-4909.

Dey, A. K. et al. (2001). "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications." *Human-computer interaction*, L. Erlbaum Associates Inc. 16(2) : 97-166. ISSN : 0737-0024.

De Castro, V. et al. (2011). "Applying CIM-to-PIM model transformations for the service-oriented development of information systems." *Information and Software Technology*, Elsevier 53(1) : 87-105. ISSN : 0950-5849.

Di Nitto, E. et al. (2008). "A journey to highly dynamic, self-adaptive service-based applications." *Automated Software Engineering* 15(3-4) : 313-341. ISSN : 0928-8910.

Diaw, S. et al. (2010). "Etat de l'art sur le développement logiciel basé sur les transformations de modèles." *Technique et Science Informatiques* 29(4-5) : 505-536.

Djeddar, A. et al. (2015). "Developing Context-aware Mobile Applications Using Composition Process based-on heterogeneous software entities." *Journal of Advanced Computer Science and Technology Research* 5(3) : 93-103.

Djeddar, A. et al. (2014). *Composition Process Based on Heterogeneous Software Entities for Mobile Applications*. In *Advanced Computer Science Applications and Technologies (ACSAT)*, 3rd International Conference on, IEEE. p. 113-118.

Dustdar, S. and W. Schreiner (2005). "A survey on web services composition." *International journal of web and grid services*, Inderscience Publishers 1(1) : 1-30. ISSN : 1741-1106.

Eclipse Soa tools (2009). *Soa tools platform : Sca tools project*. (en ligne). Disponible sur : <http://www.eclipse.org/stp/sca/>.

Emmanouilidis, C. et al. (2013). "Mobile guides : Taxonomy of architectures, context awareness, technologies and applications." *Journal of Network and Computer Applications*, Elsevier 36(1) : 103-125. ISSN : 1084-8045.

Engelstad, P. E. et al. (2006). "Service discovery architectures for on-demand ad hoc networks." *International Journal of Ad Hoc and Sensor Wireless Networks*, Old City Publishing (OCP Science) 2(1) : 27-58.

Furno, A. and E. Zimeo (2014). "Context-aware composition of semantic web services." *Mobile Networks and Applications*, Springer 19(2) : 235-248. ISSN : 1383-469X.

Garlan, D. and D. E. Perry (1995). "Introduction to the special issue on software architecture." *IEEE Trans Software Eng, Citeseer* 21(4) : 269-274.

Geebelen, K. et al. (2008). *Dynamic reconfiguration using template based web service composition*. In *Proceedings of the 3rd workshop on Middleware for service oriented computing*, ACM. p. 49-54. ISBN : 1605583685.

Gerber, A. et al. (2002). *Transformation : The missing link of MDA*. In : A. Corradini, H. Ehrig, H. Kreowski and G. Rozenberg, *Graph Transformation*, Springer : 90-105. DOI : 10.1007/3-540-45832-8_9.

Geronimo (2015). *Le succès d'une application mobile est lié à la rétention de ses utilisateurs*. (en ligne), Disponible sur : <http://www.geronimo-agency.com/le-succes-dune-application-mobile-est-lie-a-la-retention-d-e-ses-utilisateurs/>.

Ghezzi, C. et al. (2002). *Fundamentals of software engineering*, (2 nd). Prentice Hall PTR,

Upper Saddle River, USA. p. 600. ISBN : 0133056996.

Gomaa, H. and G. Farrukh (1998). Composition of software architectures from reusable architecture patterns. In Proceedings of the third international workshop on Software architecture, ACM. p. 45-48. ISBN : 1581130813.

Gu, X. et al. (2004). SpiderNet : An integrated peer-to-peer service composition framework. In High performance Distributed Computing, Proceedings. 13th IEEE International Symposium on, IEEE. p. 110-119. ISBN : 0769521754.

Gupta, A. (2015). What is the difference between WLAN 802.11 b/g/n and 802.11ac ? Which is better ?. (en ligne). In "thehackerzplanet . blogspot . com", 9 october 2015. Disponible sur : <https://www.quora.com/What-is-the-difference-between-WLAN-802-11-b-g-n-and-802-11ac-Which-is-better>

Hadley, M. et al. (2003). "SOAP Version 1.2 Part 1 : Messaging Framework." W3C REC REC-soap12-part1-20030624. (en ligne) June : 240-8491. Disponible sur : <http://www.w3.org/TR/soap12-part1.7>

Hashimi, S. Y. et al. (2011). Introducing the Android Computing Platform. In : Pro Android 3, Apress : edition April 21, 2011, p. 1-20. ISBN 978-1-4302-3222-3. 1.

He, Q. et al. (2013). "A decentralized service discovery approach on peer-to-peer networks." Services Computing, IEEE Transactions on 6(1) : 64-75. ISSN : 1939-1374.

Hoareau, C. and I. Satoh (2009). "Modeling and processing information for context-aware computing : A survey." New Generation Computing, Springer 27(3) : 177-196. ISSN : 0288-3635.

Hock-Koon, A. (2011). Contribution à la compréhension et à la modélisation de la composition et du couplage faible de services dans les architectures orientées services, Thèse de doctorat en informatique : Génie logiciel. Université de Nantes. 205 p. 28 avril 2015, <tel-01146320>.

Hock-Koon, A. and M. Oussalah (2010). "Composite service metamodel and auto composition." Journal of Computational Methods in Sciences and Engineering, IOS Press 10(1-2S2) : 215-229. ISSN : 1472-7978.

Horn, U. et al. (1999). "Services mobiles interactifs-La convergence de la radiodiffusion et des communications mobiles." UER-revue technique, Union européenne de radio-télévision(281) : 14-19. ISSN : 1019-6595.

Hourdin, V. et al. (2008). SLCA, composite services for ubiquitous computing. In Mobility'08, Proceedings of the International Conference on Mobile Technology, Applications and Systems, ACM, Singapore. p. 1-8. ISBN : 1605580899.

Indulska, J. et al. (2003). Experiences in using cc/pp in context-aware systems. Mobile Data Management, Springer. p. 247-261. ISBN : 3540003932.

Jones, M. (2013). Developing mobile apps for your users can help them be more productive, but before you start building apps, learn about the different kinds. (en ligne). In Everything you need to know about developing mobile apps, TechTarget, Disponible sur : <http://searchmobilecomputing.techtarget.com/feature/Everything-you-need-to-know-about-developing-mobile-apps> consulté le (25/05/2015).

Kalasapur, S. et al. (2007). "Dynamic service composition in pervasive computing." Parallel and Distributed Systems, IEEE Transactions on 18(7) : 907-918. ISSN : 1045-9219.

Khalaf, R. et al. (2003). Service-Oriented Composition in BPEL4WS. In : Proceeding of the 12th International World Wide Web Conference (WWW), Alternate Paper Tracks, Budapest, Hungary. p. 27-28.

Kiss, C. (2007). "Composite capability/preference profiles (cc/pp) : Structure and vocabularies 2.0." W3C working draft 30.

Larkin, J. H. and H. A. Simon (1987). "Why a diagram is (sometimes) worth ten thousand words." *Cognitive science*, Elsevier 11(1) : 65-100. ISSN : 0364-0213.

Lee, J. et al. (2008). "Dynamic service composition : A discovery-based approach." *International Journal of Software Engineering and Knowledge Engineering*, World Scientific 18(2) : 199-222. ISSN : 0218-1940.

Lenat, D. B. et al. (1990). "Cyc : toward programs with common sense." *Communications of the ACM*, ACM 33(8) : 30-49. ISSN : 0001-0782.

Lenders, V. et al. (2005). "Service discovery in mobile ad hoc networks : A field theoretic approach." *Pervasive and Mobile Computing*, Elsevier 1(3) : 343-370. ISSN : 1574-1192.

Li, L. et al. (2009). Trust-oriented composite service selection and discovery. In : L. Baresi, C. Chi and J. Suzuki, *Service-Oriented Computing*, Springer Berlin Heidelberg : p. 50-67. ISBN : 978-3-642-10382-7.

Lopez-Velasco (2009). Sélection et composition de services Web pour la génération d'applications adaptées au contexte d'utilisation, Thèse de doctorat en informatique. Grenoble I : Université Joseph-Fourier. 18 novembre 2008. 252 p.

Lopez-Velasco, C. (2009). Sélection et composition de services Web pour la génération d'applications adaptées au contexte d'utilisation, Thèse de doctorat en Informatique : Mathématiques, Sciences et Technologie de l'Information. Grenoble I : Université Joseph-Fourier, 18 novembre 2008. 253 p. <tel-00388991>.

Luckham, D. C. et al. (1995). "Specification and analysis of system architecture using Rapide." *Software Engineering, IEEE Transactions on* 21(4) : 336-354. ISSN : 0098-5589.

Luckham, D. C. and J. Vera (1995). "An event-based architecture definition language." *Software Engineering, IEEE Transactions on* 21(9) : 717-734. ISSN : 0098-5589.

Ma, Q. et al. (2008). A semantic QoS-aware discovery framework for web services. In *Web Services 2008 (ICWS'08)*. IEEE International Conference on, IEEE. p. 129-136. ISBN : 0769533108.

Martin, D. et al. (2004). "Describing web services using OWL-S and WSDL." Release 1.2 of OWL-S. (en ligne). Disponible sur : <http://www.daml.org/services/owl-s/1.2/owl-s-wsdl.html>.

Martin, D. et al. (2005). Bringing semantics to web services : The OWL-S approach. In : J. Cardoso and A. Sheth, *Semantic Web Services and Web Process Composition*, Springer : p. 26-42. ISBN : 978-3-540-24328-1.

Marvie, R. (2002). Séparation des préoccupations et méta-modélisation pour environnements de manipulation d'architectures logicielles à base de composants, Thèse de doctorat en Informatique. Lille I : Université des Sciences et Technologie de Lille, 9 décembre 2002. 208 p.

Medvidovic, N. et al. (1999). A language and environment for architecture-based software development and evolution. *Software Engineering In Proceedings of the 1999 International Conference on, IEEE*. p. 44-53. ISBN : 1581130740.

Medvidovic, N. and R. N. Taylor (2000). "A classification and comparison framework for software architecture description languages." *Software Engineering, IEEE Transactions on* 26(1) : 70-93. ISSN : 0098-5589.

Mehta, N. R. et al. (2000). Towards a taxonomy of software connectors. In *Proceedings of the*

22nd international conference on Software engineering, ACM. p. 178-187. ISBN : 1581132069.

Meier, R. (2010). Développement d'applications professionnelles avec Android 2, Pearson Education France. 640 p. ISBN : 274402452X.

Mellor, S. J. et al. (2003). "Model-driven development : guest editors' introduction." IEEE software, IEEE Computer Society 20(5) : 14-18. ISSN : 0740-7459.

Merla, C. B. (2010). "Context-aware match-making in semantic web service discovery." International Journal of Advanced Engineering Sciences and Technologies 9(2) : 243-247.

Meyer, B. (1985). "On formalism in specifications." IEEE software 2(1) : 6-26. ISSN : 0740-7459.

Mietzner, R. et al. (2008). Defining composite configurable SaaS application packages using SCA, variability descriptors and multi-tenancy patterns. In Internet and Web Applications and Services 2008. (ICIW'08), Third International Conference on, IEEE. p. 156-161. ISBN : 0769531636.

Minsky, M. (1965). "Matter, mind and models." In Proc. International Federation of Information Processing Congress (IFIP), (Original work from 1954). 1 : 45-49.

Mukhtar, H. et al. (2008). A policy-based approach for resource specification in small devices. In Mobile Ubiquitous Computing, Systems, Services and Technologies 2008 (UBICOMM'08), The Second International Conference on, IEEE. p. 239-244. ISBN : 0769533671.

Mukhtar, H. et al. (2009). User preferences-based automatic device selection for multimedia user tasks in pervasive environments. Networking and Services, 2009. ICNS'09. Fifth International Conference on, IEEE.

Neuburg, M. (2013). iOS 7 Programming Fundamentals : Objective-c, xcode, and cocoa basics, O'Reilly Media. 422 p. ISBN : 1491946903.

Nickul, D. et al. (2007). "Service oriented architecture (SOA) and specialized messaging patterns." A technical White Paper published by Adobe Corporation USA. (en ligne) , Disponible sur : <http://www.adobe.com/enterprise/pdfs/Services_Oriented_Architecture_from_Adobe.pdf>

Nimmith, T. and A. Blandine (2011). Etude de Marché : Les Applications Mobiles. Rapport de stage (en ligne). Université Paris 8., In "Technologies", 7 février 2011, 26 p. Disponible sur : <http://fr.slideshare.net/crossmedias/le-march-des-applications>Published.

Nomade, S. (2012). Les Applications pour mobiles "Une croissance continue". (en ligne). Smart Nomade : la division mobile de smart analysis, Disponible sur : <http://www.smart-nomade.com/les-applications-pour-mobiles/>.

Oeillet, A. (2013). Gartner : Plus de 100 milliards d'applications mobiles téléchargées en 2013. (en ligne). Disponible sur : <http://www.clubic.com/application-mobile/actualite-585970-gartner-100-applications-mobil-es-telechargees-2013.html>. (Page consultée le 19/09/2013).

Oussalah, M. et al. (2004). An explicit definition of connectors for component-based software architecture. In Engineering of Computer-Based Systems, Proceedings. 11th IEEE International Conference and Workshop on, IEEE. p. 44-51. ISBN : 0769521258.

Padellec, P. L. (2015). Le marché des applications mobiles : quelles évolutions? (en ligne). In Veinteractive Blog, 02 février 2015, Disponible sur : <http://www.veinteractive.com/fr/blog/le-marche-des-applications-mobiles-quelles-evolution-s/>.

Papazoglou, M. P. et al. (2007). "Service-oriented computing : State of the art and research challenges." Computer, IEEE 40(11) : 38-45. ISSN : 0018-9162.

Papazoglou, M. P. and W.-J. Van Den Heuvel (2007). "Service oriented architectures : approaches, technologies and research issues." *International Journal on Very Large Data Bases (VLDB)*, Springer 16(3) : 389-415. ISSN : 1066-8888.

Parnas, D. L. (1972). "On the criteria to be used in decomposing systems into modules." *Communications of the ACM*, ACM 15(12) : 1053-1058. ISSN : 0001-0782.

Pascoe, J. (1998). Adding generic contextual capabilities to wearable computers. *Wearable Computers, 1998. Digest of Papers. Second International Symposium on*, IEEE. p. 92-99. ISBN : 0818690747.

Pascoe, J. et al. (1998). Human-Computer-Giraffe Interaction-HCI in the Field. In *Workshop on Human Computer Interaction with Mobile Devices*.

Peltz, C. (2003). "Web services orchestration and choreography." *IEEE Computer*, IEEE 36(10) : 46-52. ISSN : 0018-9162.

Perchat, J. (2015). Composants multiplateformes pour la prise en compte de l'hétérogénéité des terminaux mobiles. Thèse de doctorat en Informatique. Valenciennes : Université de Valenciennes et du Hainaut-Cambresis, 08-01-2015, 209 p.

Preuveneers, D. et al. (2004). Towards an extensible context ontology for ambient intelligence. In *Proceedings of the 2nd European Symposium on Ambient Intelligence (EUSAI 2004)*, Springer : p. 148-159. ISBN : 3540237216.

Prochart, G. et al. (2007). Fuzzy-based support for service composition in mobile ad hoc networks. In *Pervasive Services, IEEE International Conference on*, IEEE. p. 379-384. ISBN : 1424413257.

Rake, J. et al. (2009). Enhancing semantic service discovery in heterogeneous environments. In : W. Abramowicz, *Business Information Systems*, Springer Berlin Heidelberg. p. 205-216. ISBN : 978-3-642-01189-4.

Rao, J. and X. Su (2005). A survey of automated web service composition methods. In : J. Cardoso and A. Sheth, *Semantic Web Services and Web Process Composition*, Springer. p. 43-54. ISBN : 978-3-540-24328-1.

Rasch, K. et al. (2011). "Context-driven personalized service discovery in pervasive environments." *World Wide Web*, Springer 14(4) : 295-319. ISSN : 1386-145X.

Rey, G. and J. Coutaz (2002). Le contexteur : une abstraction logicielle pour la réalisation de systèmes interactifs sensibles au contexte. In *Proceedings of the 14th French-speaking conference on Human-computer interaction (Conférence Francophone sur l'Interaction Homme-Machine)*, ACM. p. 105-112. ISBN : 1581136153.

Roques, P. (2008). *UML 2 : Modéliser une application web*, (4 ed). Eyrolles Eddition. 246 p. ISBN : 2212852185.

Rosa, R. E. V. and V. F. Lucena Jr (2011). Smart composition of reusable software components in mobile application product lines. In *Proceedings of the 2nd International Workshop on Product Line Approaches in Software Engineering*, ACM. p. 45-49. ISBN : 1450305849.

Rozier, U. (2015). Test du Samsung Galaxy S6, tout est mieux et différent. (en ligne). FrAndroid, Disponible sur : http://www.frandroid.com/marques/samsung/274199_test-du-samsung-galaxy-s6-tout-est-mieux-et-different.

Saha, D. and A. Mukherjee (2003). "Pervasive computing : a paradigm for the 21st century." *Computer*, IEEE 36(3) : 25-31. ISSN : 0018-9162.

Schilit, B. N. and M. M. Theimer (1994). "Disseminating active map information to mobile

hosts." Network, IEEE 8(5) : 22-32. ISSN : 0890-8044.

Shaw, M. et al. (1995). "Abstractions for software architecture and tools to support them." Software Engineering, IEEE Transactions on 21(4) : 314-335. ISSN : 0098-5589.

Soucé, J.-M. and L. Duchien (2002). Etat de l'art sur les langages de description d'architecture. Rapport technique. Juin 2012, Ref : Livrable1.1-2. 62 p.

Srinivasan, L. and J. Treadwell (2005). "An overview of service-oriented architecture, web services and grid computing." HP Software Global Business Unit!. (en ligne). Vol. 2. Disponible sur : <http://h71028.www7.hp.com/ERC/downloads/SOA-GridHP-WhitePaper.pdf>.

Statista (2015). Number of apps available in leading app stores. (en ligne). Firme de Recherche Statista : <http://www.statista.com/>, Disponible sur : <http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>.

Statista (2015). Number of mobile app downloads worldwide from 2009 to 2017. (en ligne). Firme de Recherche Statista : <http://www.statista.com>. Statistique sur les Applications mobiles téléchargées, Disponible sur : <http://www.statista.com/statistics/266488/forecast-of-mobile-app-downloads/>.

Steinberg, D. et al. (2008). EMF : eclipse modeling framework 2.0, (2nd) Addison-Wesley Professional. ISBN : 0321331885.

Stojanovic, Z. and A. Dahanayake (2005). "Service-Oriented Software System Engineering : Challenges and Practices." Journal of Digital Information Management, Digital Information Research Foundation 3(3) : 210. ISSN : 0972-7272.

Strang, T. and C. Linnhoff-Popien (2004). A context modeling survey. In Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004, The Sixth International Conference on Ubiquitous Computing, Nottingham/England, 2004. (En ligne). Disponible sur : <http://elib.dlr.de/7444/1/Ubicomp2004ContextWSCameraReadyVersion.pdf>.

Strang, T. et al. (2003). CoOL : A Context Ontology Language to enable Contextual Interoperability. In : J.-B. Stefani, I. Dameure and D. Hagimont (eds.), Distributed Applications and Interoperable Systems, Paris. p. 236-247. ISBN : 978-3-540-20529-6.

Suraci, V. et al. (2007). Context-aware semantic service discovery. In Mobile and Wireless Communications Summit, 2007. 16th IST, 1-5 july., IEEE. p. 1-5. ISBN : 9638111666.

Surianarayanan, C. et al. (2015). "Towards quicker discovery and selection of web services considering required degree of match through indexing and decomposition of non-functional constraints." International Journal of Computational Science and Engineering 10(1-2) : 45-69.

Szyperski, C. et al. (1999). Component-oriented programming. In Object-oriented technology ecoop'99 workshop reader, Springer Berlin Heidelberg. p. 184-192. ISBN : 354066954X.

Tigli, J. and S. Lavirotte (2006). "Adaptation dynamique à l'environnement d'exécution : un enjeu pour l'informatique mobile et ambiante. [talk]." Séminaire invité à l'IMAG, Grenoble, France, novembre 2006.

Toma, I. et al. (2005). A P2P discovery mechanism for web service execution environment. In Second WSMO Implementation Workshop.

TOPCASED-WP5, s. (2008). "Guide méthodologique pour les transformations de modèles." Rapport de recherche, version 0.1, 18 Novembre, IRIT/MACAO.

Van Hien, N. (2007). "Application du modèle MDA (Model-Driven Architecture) dans le développement du logiciel." Institut de la Francophonie pour l'Informatique, Hanoi, 20 juillet.

Varshavsky, A. et al. (2005). A cross-layer approach to service discovery and selection in MANETs. In Mobile Adhoc and Sensor Systems Conference(MASS), IEEE International Conference on, 7-7 Nov. 2005, Washington, DC, IEEE Computer Society. 8 pp.-466. ISBN : 0780394658.

Verma, K. et al. (2005). The METEOR-S approach for configuring and executing dynamic web processes. Technical Report TR6-24-05, The University of Georgia, Computer Science Department, LSDIS Lab, Athens, Georgia.

Wang, X. H. et al. (2004). Ontology based context modeling and reasoning using OWL. In Pervasive Computing and Communications Workshops, Proceedings of the Second IEEE Annual Conference on, IEEE. p. 18-22. ISBN : 0769521061.

Want, R. et al. (1992). "The active badge location system." ACM Transactions on Information Systems (TOIS), ACM 10(1) : 91-102. ISSN : 1046-8188.

Weinreich, R. and J. Sametinger (2001). Component Models and Component Services : Concepts and Principles. In : George T. Heineman, William T. Councill, Component-based software engineering : putting the pieces together, Addison-Wesley Professional ; 1 edition. June 18, 2001 : 33-48. ISBN : 978-0768682076.

Weiser, M. (1991). "The computer for the 21st century." Scientific american, Nature Publishing Group. 265(3) : 94-104. ISSN : 0036-8733.

Wu, Z. et al. (2007). Automatic composition of semantic web services using process and data mediation. Technical Report, LSDIS lab, University of Georgia, February 28. Disponible sur : <http://corescholar.libraries.wright.edu/knoesis/8>.

Zhang, D. and B. Adipat (2005). "Challenges, methodologies, and issues in the usability testing of mobile applications." International Journal of Human-Computer Interaction, Taylor & Francis 18(3) : 293-308. ISSN : 1044-7318.

Zhang, X. et al. (2011). "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing." Mobile Networks and Applications, Springer-Verlag New York, Inc. 16(3) : 270-284. ISSN : 1383-469X.

Zhang, Z. et al. (2008). An integrated approach to service selection in mobile ad hoc networks. In Wireless Communications, Networking and Mobile Computing, 2008. WiCOM'08. 4th International Conference on, IEEE. p. 1-4. ISBN : 1424421071.