Service-Oriented Architecture for the Mobile Cloud Computing

Fatiha Houacine

le cnam

Ecole Doctorale d'Informatique, Télécommunications et Electronique
Centre d'étude et de recherche en informatique et communications

THÈSE présentée par :

Fatiha HOUACINE

Soutenue le : **25 Novembre 2016**

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

Discipline/ Spécialité: INFORMATIQUE

Service-Oriented Architecture for the Mobile Cloud Computing

THÈSE dirigée par :

Mme Samia BOUZEFRANE	HDR, Arts et Métiers Paris.

RAPPORTEURS :

M. Eric RENAULT	HDR, Telecom SudParis
M. Weishan ZHANG,	Pr, University of Petroleum, China

JURY :

Mme Leila SAIDANE « président du jury »	Pr, Manouba University
Mme Chantal TACONET	HDR, Telecom SudParis
M Gaétan HAINS	Pr, Paris Est Créteil University
Mme Anne WEI	HDR, CNAM
M Nikolaos GEORGANTAS	Chargé de recherche, INRIA

Résumé

Le développent des objets mobiles et connectés accélèrent l'émergence d'applications mobiles qui exigent des niveaux élevés de réactivité et qui, en retour nécessitent des ressources de calcul intensif. Ce qui accentue la complexité des applications. Cela introduit le besoin de définir de nouvelles méthodes pour réduire la complexité du logiciel et pour fournir un support facilitant leur réutilisation.
Le paradigme du Cloud présente un grand potentiel pour apporter de nouveaux scénarios d'applications avec des applications locales simplifiées et en déportant la complexité vers des machines centrales.

Cette thèse vise à définir un cadre d'architecture pour le support des applications Cloud mobiles, ce travaille focalise notamment sur trois problématiques principales:

1. La nécessité d'un cadre traitant des contraintes d'environnement mobile (limitation de ressources, l'hétérogénéité...)
2. Environnement variable et besoin de portabilité
3. Challenge lies à la sécurité notamment le besoin de contrôle d'intégrité des services exécutés à distance. Le Cloud Mobile offre une extension de ressources sur les serveurs centraux et dans une vision plus large, les services MCC à travers la combinaison de dispositifs capacités physiques (mobilité, capteurs., etc.) et les entités virtuelles logicielles distribuées peuvent offrir de nouvelles perspectives.

Dans un tel contexte la modularité est la clé d'un support efficace d'applications MCC. Pour parvenir à cette modularité, l'architecture orientée services permet de résoudre ce défi principal. Avec ce concept, les services en SOA sont indépendants, ce qui garantit l'évolutivité et scalabilité du système, même avec des applications complexes. En outre, la séparation des métadonnées et de l'interface à partir des ressources de service, les dépendances entre les services sont gérées dynamiquement et les services peuvent être réutilisables.

On a identifié dans la plate-forme OSGi (Open Service Gateway initiative) un excellent support pour la conception SOA. Initialement introduite pour les systèmes embarqués, OSGi est nativement un système léger qui correspond à des appareils mobiles et les limites des systèmes embarqués.
Comme première contribution, on propose de mettre en place une architecture orientée services (SOA) basée sur la plateforme OSGi pour mobile Cloud computing, qui prend en charge les modules de services basés sur Java en cours d'exécution sur les deux plates-formes mobiles et Cloud. On introduit l'intégration de OSGi pour les deux rôles de client et fournisseur de service.

Les principes SOA ont été repris et introduits les "Andromodules" une nouvelle plate-forme de conception modulaire pour le développement d'application. Cette conception est directement mise en œuvre sur le système d'exploitation mobile pour fournir des performances optimales.
La seconde partie présente une solution d'architecture MCC globale adaptée à trois environnements différents: Le premier schéma est une conception centralisée où les mobiles consomment les services. Le second schéma représente une conception distribuée basée sur la coopération Mobile-à-Mobile. Et enfin face à un environnement hostile, un modèle 3-tiers est considéré, où les cloudlets agissent comme des entités intermédiaires entre les appareils mobiles et les serveurs de Cloud centrale.

Une Analyse de la performance de la solution est proposée pour chaque cas et celle-ci est contrastée avec les performances de solutions existantes alternatives. On focalise ensuite sur les enjeux de la sécurité dans un cadre distribué, et notamment l'intégrité des services MCC exécutés au sein des entités distantes.
On propose de combiner les concepts de sécurité OSGi, des protocoles de communication de sécurité, et en incluant la sécurité lors dés la phase d'architecture.

Mots clés: Cloud Computing, OSGi, Mobile Cloud, Application Framework, Mobile design

Résumé en anglais

Mobile devices and Internet of things accelerate the emergence of various real-time mobile applications that demand high levels of responsiveness and that, in return, demand intensive computing resources making the complexity of the applications in embedded systems increasing.

This introduces the need of new methodologies to reduce the complexity of the software and to supply a support facilitating its re-use. Cloud computing paradigm has a great potential to bring new mobile application scenarios with simplified local mobile apps.

This PhD thesis, aims to define a mobile design framework for Mobile cloud Service, it focus on three main problems:

1. The need for a framework dealing with mobile environment constraints (lack of resources, heterogeneity.
2. Environment variation and portability needs
3. Framework security challenges: Software integrity control while services are executed remotely.

Mobile Cloud computing offers through Cloud server's ubiquitous, on-demand resources and in a larger vision, MCC services look like and how the combination of physical devices capabilities (mobility, sensing, large deployment, etc.) and the virtual entities distributed Cloud services can offer important new perspectives.

In such distributed and dynamic context, modularity and low coupling are the key in building efficient MCC applications. To achieve this modularity, Service Oriented Architecture permit to resolve this main challenge since SOA breaks up the application into small interacting parts so that these components can be offloaded and run on multiple devices as separate services. With this concept, services in SOA are independent and low coupled, which ensures the scalability of the system even with complex applications. In addition, with the metadata and interface separation from service resources, the dependencies between services are managed dynamically and pieces of application, i.e., services can be reusable. We identified in OSGi platform (Open Service Gateway initiative) an excellent support for dynamic SOA design for MCC services. Originally introduced for embedded systems, OSGi is natively a lightweight system that matches mobile devices and embedded systems limitations.

As a first contribution, We propose to establish a Service-Oriented Architecture (SOA) based on the OSGi (Open Service Gateway initiative) platform for mobile Cloud computing, that supports the Java-based services modules running on both mobile and Cloud platforms.

This is achieved by incorporating OSGi into Android software development platform that interacts with Remote-OSGi on the Cloud. We propose the integration of OSGi as a mobile Cloud framework and we explore the suitable models for both client and provider roles. We presented also Andromodules a new modular design platform for application development. This design is directly implemented on the mobile OS to provide optimal performances. Secondly, we present a global MCC architecture solution adapted to three different designs depending on both the application domain constraints and the execution environment:

The first scheme is a Centric mobile Cloud design that offers to mobile devices offloading capabilities into powerful Cloud-centric servers and remote services consuming.

The second scheme represent a distributed design based on Mobile-to-Mobile (M2M) cooperation. And finally dealing with hostile environment, a 3-tiers model is considered, where cloudlets act as intermediary nodes between mobile devices and centric Cloud servers. We explore the performance of our service-oriented framework in each case and contrast it with alternative existing solutions.

Finally this thesis focus on the security of a distributed framework, and the integrity of MCC services executed within remote entities. This section, propose to secure the framework, by combining OSGi security concepts, security communication protocols and by including the security in the design of the framework itself.

Key words: Cloud Computing, OSGi, Mobile Cloud, Application Framework, Mobile design

Résumé

Contexte et objectifs

La popularité des appareils mobiles tels que les smartphones et, plus généralement, les objets connectés, offre des supports de communication et d'information omniprésents, créant une dépendance dans tous les domaines allant de la vie quotidienne aux services aux entreprises... etc.
Les terminaux mobiles accélèrent l'émergence de diverses applications mobiles en temps réel qui exigent des niveaux de réactivité élevés et qui, en retour, exigent des ressources informatiques intensives compte tenu de la complexité des applications embarquées, ce qui introduit le besoin de nouvelles méthodologies pour réduire la complexité du logiciel et de fournir un support facilitant sa réutilisation.

Dans ce contexte, le cloud computing a un grand potentiel pour apporter de nouveaux scénarios d'applications mobiles avec des applications mobiles locales simplifiées.
Avec la puissance du Cloud, les applications mobiles se déplacent progressivement vers le Cloud et s'éloignent de l'installation et s'exécutent directement depuis les périphériques eux-mêmes. Au lieu de cela, les applications sont accessibles et exécutées directement à partir du Cloud via des interfaces mobiles simples.
Dans cette thèse, nous nous concentrons sur trois problèmes principaux

La nécessité d'un cadre traitant des contraintes mobiles: L'environnement Cloud mobile peut être un environnement hautement hétérogène avec des capacités logicielles / matérielles et une variation technologique parmi les appareils mobiles.
Dans un tel environnement de cloud computing, la complexité de l'intégration des applications entre plusieurs fournisseurs de services Cloud est également accrue. Par conséquent, la gestion des services pour les fournisseurs de services mobiles distribués devient un défi de recherche critique.

2. Variabilité de l'environnement et besoins de portabilité: le développement de composants multiplateformes (Cloud, mobile et hybride) pour les applications Cloud-mobile est une tâche difficile. En effet, les composants mobiles doivent pouvoir se déplacer entre différents smartphones tandis que les composants Cloud doivent être portables sur toutes les infrastructures Cloud.

3. Sécurité du Framework: un autre challenge dans les infrastructures middleware Cloud est l'intégrité logicielle. Les services distribués sont construits sur plusieurs nœuds et puisque les services sont exécutés à distance, le client mobile dispose de moyens limités pour garantir par exemple que le service exécuté est le bon service attendu.

Cette thèse propose une solution d'ingénierie logicielle pour les services de Cloud Mobile Computing incluant trois contributions principales:

Contributions

Contribution 1: Nous proposons d'établir une architecture AOS (Architecture Orientée Service) basée sur la plate-forme OSGi (Open Service Gateway) pour le Cloud Computing mobile, prenant en charge les modules de services Java fonctionnant sur les plateformes mobiles et Cloud. Ceci est réalisé en incorporant OSGi dans la plate-forme de développement de logiciels Android qui interagit avec Remote-OSGi sur le Cloud. Nous proposons l'intégration d'OSGi en tant que Framework de Cloud mobile et nous explorons les modèles appropriés pour les rôles client et fournisseur.

Contribution 2: nous adaptons le cadre MCC proposé à différents contextes d'architecture. Le premier contexte est le modèle centrique traditionnel, où les appareils mobiles actent comme consommateurs de services uniquement. Le deuxième contexte est le modèle distribué où la puissance de l'interaction mobile-mobile offre des opportunités illimitées de services de valeur. Enfin, le troisième contexte est une architecture à trois niveaux qui introduit la notion de Cloudlet. Pour chaque modèle, nous explorons la performance de notre cadre axé sur les services et le comparons à d'autres solutions existantes.

Contribution 3: Sécuriser la vie privée et l'intégrité des données et des applications utilisateurs de Cloud Computing mobile est l'un des problèmes clés. Dans ce travail, nous nous intéressons à la sécurité d'un framework distribué et à l'intégrité des services MCC exécutés au sein d'entités distantes. Nous proposons de sécuriser le framework, en combinant les concepts de sécurité OSGi, les protocoles de communication de sécurité et en incluant la sécurité dans la conception du framework lui-même.

Cloud computing mobile

Le Cloud computing Mobile (MCC) est défini comme un service permettant aux utilisateurs mobiles ayant des ressources restreintes d'étendre de manière adaptative les capacités de traitement et de stockage en déportant de manière transparente les tâches exigeantes en calcul et stockage sur les ressources Cloud traditionnelles. ". Cette vision est correcte, mais elle considère le Cloud Computing mobile comme un service à sens unique sur un modèle "client / serveur" centralisé. Le cloud computing mobile ne peut être simplement illustré par la fusion des technologies de l'informatique mobile et du cloud computing.

Dans les développements récents du Cloud Computing mobile [6, 7], les appareils mobiles sont impliqués en tant que fournisseurs de services Cloud pour offrir des services de détection basés sur le Cloud. Par exemple, un appareil mobile peut capter les informations environnantes à travers différents capteurs embarqués et connectés, tels que l'état des canaux de communication sans fil, les informations sur les nœuds voisins, les informations environnementales (par exemple, la qualité de l'air). Ceci donne une vision plus large de ce à quoi pourraient ressembler les services MCC et comment la combinaison des capacités des dispositifs physiques (mobilité, détection, déploiement étendu, etc.) et des entités virtuelles distribuées offre des nouvelles perspectives.
De nombreux systèmes physiques mobiles sont considérés localement et sont gérés de manière unique et répétitive. Avec le Cloud computing, ces systèmes peuvent être enregistrés, suivis, interrogés / recherchés à distance, contrôlés, reproduits et analysés en corrélant les activités

pertinentes. L'activation de ces fonctionnalités peut grandement améliorer la connaissance de la situation des futures applications mobiles par prédiction ou personnalisation

Les architectures orientés Services et le MCC

Avec le développement des technologies d'accès sans fil, les appareils mobiles peuvent accéder au cœur du réseau avec une bande passante plus stable et plus garantie. De nos jours, les services Web sont devenus un paradigme de communication universel, et les appareils mobiles avec services Web sont considérés comme des plates-formes de services pour l'architecture orientée services. Il est intuitif d'étendre les services SOA aux appareils mobiles, qui sont essentiels dans les environnements qui dépendent de connexions non robustes dans les environnements mobiles.
En combinant la puissance du paradigme SOA et Cloud computing dans le contexte MCC, la complexité et les contraintes des systèmes sont abordées grâce à la modularité, la dynamicité et la réutilisabilité des composants SOA.
Le concept de SOA est basé sur une architecture qui définit un modèle d'interaction entre trois entités principales :
1- Le fournisseur de services: Publie une description d'un service et de sa mise en œuvre.
2- Le registre des services: conserve et met à jour l'arrivée ou le départ d'un service et le rend accessible aux consommateurs de services.
3- Le consommateur de service: consulte le registre de services et appelle les services souhaités.
En fonction de ces entités, les applications peuvent rechercher les services dans le registre et appeler le service requis.

Dans SOA, les utilisateurs peuvent accéder aux services sans avoir aucune connaissance sur l'implémentation du service et sa complexité qui sont absentes grâce à l'interface simple exposée par le service lui-même. Ceci est utile pour le consommateur de service qui nécessite une simplicité d'accès au service et pour le fournisseur de services qui doit protéger le code source du service. Grâce à la modularité, SOA permet aux utilisateurs de fournir des applications distribuées dynamiques construites presque entièrement à partir de services logiciels existants. En outre, les services SOA utilisent des protocoles définis qui décrivent comment les services transmettent et analysent les messages à l'aide de métadonnées de description, qui décrivent en détails non seulement les caractéristiques de ces services, mais également les données qui les pilotent.

Les caractéristiques de l'architecture orientée services peuvent permettre de résoudre certains des principaux défis de l'ingénierie logicielle dans le cloud computing. La modularité est l'un des concepts clés de la SOA, car la SOA décompose l'application en petites parties qui interagissent pour que ces composants puissent être déchargés et exécutés sur plusieurs périphériques en tant que services distincts. Avec ce concept, les services SOA sont indépendants et faiblement couplés, ce qui garantit l'évolutivité du système même avec des applications complexes.
De plus, avec la séparation des métadonnées et des interfaces des ressources de service, les dépendances entre les services sont gérées dynamiquement et les parties d'application, c'est-à-dire, les services peuvent être réutilisables. La gestion du service est basée sur le contrat de service pour assurer l'abonnement au service et l'accord de communication. L'encapsulation de service traite de l'hétérogénéité de l'environnement, en permettant à tout logiciel d'être exécuté

en tant que partie d'une entité d'architecture. La ségrégation entre l'interface de service et son contenu, également appelée abstraction de boîte noire, devant la complexité de l'environnement, limite la connaissance de la logique de service au contrat.

OSGi une plateforme SOA adaptée

OSGi (Open Service Gateway Initiative) est une plateforme de services basée sur le langage Java. OSGi introduit une architecture orientée services qui permet une plus grande indépendance entre les différents composants d'un programme. Il est introduit par l'OSGi Alliance [46], une organisation fondée en mars 1999, qui regroupe plus de 35 industriels et laboratoires (IBM, Nokia, Philips, BMW, Motorola, Sun Microsystems, etc.).

La première motivation de l'OSGi Alliance était de définir un modèle d'administration des facilités de transition dans lequel l'administration est transparente pour l'utilisateur bénéficiant des services offerts. Initialement conçu pour les environnements réseau de réseau domestique et de capteurs (ce qui explique sa légèreté), OSGi a rapidement couvert de nombreux contextes d'application tels que l'automatisation, l'automobile, les télécommunications et l'industrie.

Un bundle est une unité logique sous la forme d'un fichier Jar contenant le code compilé, les ressources de service et les métadonnées.
Les métadonnées sont stockées dans le fichier Manifest.mf. Le fichier Manifest.mf est utilisé par l'infrastructure pour définir les exigences et les dépendances de l'ensemble, ainsi que les packages et les services partagés.
Un bundle peut exposer son interface sans échanger les classes de ressources.
Dans le contexte Java, un bundle est un fichier Jar simple.

OSGi fournit un système d'interaction hautement évolué entre les bundles basés sur les paquets Import et Export. Les packages sont l'unité d'échange entre les bundles.
Il n'y a pas de partage par défaut entre les bundles OSGi. L'interaction du package est basée sur un modèle publish / subscribe. Cela signifie qu'un ensemble peut être un serveur fournissant des services ou un client nécessitant des services.
Le même package peut agir simultanément en tant que serveur et client pour différents services.
Pour être en mesure d'exporter ou d'importer, un package doit spécifier ses dépendances dans le fichier Manifest.mf en utilisant les entrées Import-Package et Export-Package

Adaptation OSGI au contexte mobile local

AndroLix : Dans une première partie, nous avons conçu un intergiciel appelé AndroLix qui fournit une communication inter-OSGi à travers le langage de définition d'interface Android (AIDL) dans les plates-formes mobiles Android.

L'accès à Felix –l'implémentation du framework OSSi - se fait via un mécanisme de communication RPC "appel de procédure à distance". AndroLix fournit aux clients une description AIDL contenant différentes méthodes implémentées par ce service. Dans AndroLix, chaque application Android est définie comme un couple composé d'un bundle Felix OSGi et d'une application Android apk. Le bundle spécifie dans l'entrée Import-package non seulement les packages dont il a besoin, mais aussi les paquets nécessaires à l'apk correspondant.

Le rôle du bundle associé est de représenter l'application dans le framework Felix, c'est-à-dire que le bundle garantit la résolution des dépendances de l'application apk avant son installation et peut accéder à d'autres bundles Felix s'ils sont requis par l'application apk.

Une architecture SOA OSGi pour les services Cloud mobiles

Dans l'AndroLix précédemment introduit, OSGi est intégré aux plates-formes Android pour une exécution locale. Notre première contribution est d'étendre ce modèle au contexte de distribué. Le fournisseur et le consommateur d'un service sont situés dans deux nœuds séparés, et la communication entre les deux nœuds est assurée via un réseau sans fil. Pour permettre l'interaction entre les bundles de plates-formes Android (consommateur de services) et les services Cloud (fournisseur de services), nous utilisons Remote OSGi: R-OSGi .
R-OSGi est une plate-forme middleware distribuée qui étend la spécification OSGi centralisée et standard de l'industrie pour prendre en charge la gestion des modules distribués. Nous avons utilisé R-OSGi en bundle du côté Cloud et nous avons intégré le bundle associé au sein des plates-formes Android en tant que service bundle proxy.
Comme le montre la Figure 5.3, le groupe de clients effectue une connexion à distance au service via l'ensemble R-OSGi qui appelle le service distant et vérifie son existence avant de commencer l'exécution de l'application. L'infrastructure de client de service prend soin d'importer le service distant en établissant un point de terminaison de proxy local, qui est une instanciation de l'ensemble R-OSGi et en le connectant au point de terminaison distant. R-OSGi permet à une application OSGi centralisée d'être distribuée de manière transparente aux limites de service en utilisant des proxies.
Le proxy R-OSGi est indiscernable des services OSGi locaux. Le protocole R-OSGi sur le proxy est utilisé pour effectuer des invocations distantes au service d'origine, situé sur le nœud Cloud, et les événements du serveur Cloud sont transférés de manière transparente vers le périphérique mobile et se produisent comme s'ils étaient émis par un bundle local.

Solution modulaire et légère pour les plateformes de développement Android

La précédente solution OSGi Felix tente de fournir une modularité en incorporant des frameworks en tant que couches intermédiaires entre l'application et le système d'exploitation. En effet, dans cette solution, pour exécuter les services d'un bundle local, l'application Android (apk) déclenche un bundle proxy en exécutant Android API, puis le proxy demande au middleware OSGi qui appelle le bundle demandé. Le bundle recherche des packages élémentaires de la machine virtuelle Java, puis la couche OS mobile déclenche la capacité physique. Ce processus d'exécution permet des impacts négatifs en termes de performances des applications de bout en bout. Cela nous motive à faire une nouvelle proposition pour incorporer des primitives SOA OSGi directement dans la couche OS Android.
Dans cette partie, nous proposons une nouvelle adaptation de SOA dans le contexte mobile en intégrant des primitives SOA OSGi directement dans la couche OS Android. Notre solution vise à développer un progiciel basé sur des composants qui implémente le modèle de composant de service avec des capacités de modularité et de réutilisabilité. Par conséquent, la modularité et la dynamicité sont assurées sans introduire de middlewares multicouches. La solution proposée implémente un modèle basé sur des composants appelés Andro-modules, qui rend la couche système d'exploitation Android orientée nativement vers le service.

Le fichier Manifest contient des informations d'ordre exécutif et non exécutif. Les informations non exécutives sont liées à la version de l'Andromodule, à l'auteur et à d'autres descriptions, tandis que les informations exécutives correspondent aux Andromodules prérequis, aux services ou API disponibles publiquement aux autres Andromodules et aux exigences d'exécution minimale des performances. . Notez que l'idée de l'Andromodule est analogue à l'unité bundle OSGi sauf que l'Andromodule est exécuté directement comme un code Dalvik, évitant le processus de dexification utilisé dans la première solution avec AndroLix middleware pour convertir le code Byte en code DEX.
L'architecture basée sur Andromodule est une plate-forme légère par rapport à l'architecture basée sur OSGi, car les applications Android interagissent directement avec Andromodules pour l'exécution sans passer par différentes couches de middleware. En outre, les Andromodules font partie des modules OS pour lesquels la gestion est lancée lors du lancement du système d'exploitation Android. En conséquence, aucun impact sur les performances de l'application en cours d'exécution. Le système de gestion des services, implémenté en tant que classe Android, comprend de nombreuses méthodes utilisées pour gérer le cycle de vie des Andromodules, pour enregistrer et partager des services entre les fournisseurs de services Andromodule, et pour découpler les consommateurs de services. Les principales fonctions du système de gestion de service sont les suivantes:

• installModule (emplacement): installe un module à partir d'un emplacement spécifié et lui attribue automatiquement un ID.
• start (Id): vérifie l'existence d'un module correspondant à l'ID donné, tente de résoudre ce module et le lance s'il réussit à le résoudre. La résolution de dépendance de module dépend des informations qui existent dans le fichier XML. En outre, il appelle la méthode activate ().
• activate (Id): vérifie l'existence d'un module correspondant à l'ID donné et enregistre le service dans le registre.
• stop (Id): vérifie l'existence d'un module correspondant à l'ID donné et arrête le service s'il est dans un état de démarrage ou actif.
• uninstallModule (Id): vérifie l'existence d'un module correspondant à l'ID donné et le désinstalle s'il est résolu ou arrêté.

Les différents designs

Dans la seconde partie, nous adaptons le framework MCC proposé à différents contextes architecturaux: le premier est un modèle centrique traditionnel où les appareils mobiles ne consomment que des services, le second est un modèle distribué où l'interaction des mobiles vers les mobiles offre des services à valeur ajoutée opportunités, et la troisième est une architecture à trois niveaux basée sur la notion de Cloudlet. Pour chaque contexte, nous explorons la performance de notre cadre axé sur les services et le comparons à d'autres solutions existantes

Design centralisé

La définition du cloud computing mobile est généralement liée à une architecture centralisée. Ce modèle vise à:
Prolonger les capacités des dispositifs limités au calcul de stockage;

Fournir un accès distant transparent aux données et applications sur un serveur riche en ressources
Dans le contexte mobile, la communication peut être interrompue à cause de la déconnexion du réseau à n'importe quel moment. Il n'y a pas d'action possible à faire du côté du fournisseur si le client est inaccessible. Cependant, nous devons prendre en compte cette possibilité du côté du client en proposant une adaptation de la gestion d'état de bundle pour traiter les déconnexions de réseau, en particulier dans des environnements hostiles et mobiles.
Pour gérer la continuité de service dans ce cadre MCC OSGi en examinant le comportement des services démarrés et en cours d'exécution en cas de déconnexion avec les bundles distants, nous proposons de compléter l'état du bundle en ajoutant un processus de vérification de joignabilité localement au bundle.
Ce processus est exécuté comme suit: après la première résolution de bundle, l'accès à l'ensemble distant est vérifié en utilisant les attributs de délai et de fréquence. Si l'ensemble distant est considéré comme inaccessible, l'ensemble client est défini sur un état "gelé" jusqu'à ce que l'ensemble requis soit à nouveau accessible. Si le délai d'expiration de l'application est écoulé, un "drapeau blanc" est renvoyé au groupe de clients qui passe à l'état d'arrêt sans intercepter l'application. La solution détaillé est décrite dans le chapitre 5 de cette thèse.

Contexte mobile à mobile distribué

Dans un modèle entièrement distribué avec interaction mobile ad hoc, une découverte dynamique du meilleur fournisseur environnant est nécessaire pour éviter un goulot d'étranglement centralisé. Pour résoudre les problèmes de communication inter-OSGi, nous adoptons une solution XMPP basée sur des groupes de découverte: Le groupe de découverte enregistre un programme d'écoute de diffusion utilisé pour écouter la diffusion Android envoyée à l'infrastructure OSGi à partir du processus interne du service Android. Lors de la réception d'une diffusion, l'ensemble de découverte enverra le message à un autre service s'exécutant dans l'infrastructure OSGi. Dans notre cas, il sera envoyé au bundle client XMPP.

Modèle basé sur Cloudlet

Les cloudlets sont proposées pour créer un cloud proche pour accéder aux ressources distantes proches.
Un Cloudlet est un nouvel élément architectural issu de la convergence de l'informatique mobile et du Cloud computing. Il représente le niveau intermédiaire d'une hiérarchie à trois niveaux: appareil mobile, Cloudlet et Cloud.
Un Cloudlet peut être vu comme un "centre de données dans une boîte" pour rapprocher le Cloud de l'appareil mobile tout en s'appuyant sur une bande passante élevée (par exemple, WLAN Wi-Fi à 1 saut) et de faibles latences

Comparaison OSGI, Elijah VM et conteneurs

Nous utilisons dans ce qui suit une solution Elijah and une solution basée sur les dockers pour comparer les interactions et les performances avec la solution OSGi

Projet Elijah

Elijah est le résultat d'un projet de Simanta et al., dans [76], présentent une architecture de référence basée sur Cloudlets.
Basés sur un gestionnaire de machine virtuelle, les Cloudlets sont considérés comme des éléments de déchargement de code servant à servir les appareils mobiles à proximité d'un seul saut.
Une superposition d'application est créée une fois par application
La VM de base est un fichier d'image disque VM obtenu à partir du noyau central et enregistré dans un Cloudlet qui exécute un gestionnaire VM compatible avec la VM de base. Le gestionnaire de machine virtuelle démarre la machine virtuelle de base et l'application est installée. Après l'installation, la machine virtuelle est arrêtée. Une copie de l'image de disque VM de base modifiée est enregistrée en tant qu'image de disque VM complète. La superposition de l'application est calculée en tant que diff binaire entre l'image disque VM complète et l'image disque VM de base.

Solution Cloudlet à base de conteneurs

Dans la virtualisation basée sur conteneur, la planification (scheduling) se produit uniquement à un niveau. Les conteneurs contiennent des processus, tandis que le noyau et les bibliothèques sont partagés entre les conteneurs. Par conséquent, le planificateur est utilisé uniquement au niveau du processus. Le comportement est similaire à un système d'exploitation classique; la seule différence est que, dans la virtualisation par conteneur, les processus sont affectés à un conteneur, et entre chaque conteneur, l'isolation se produit à deux niveaux:
• Isolement entre les processus; et
• Isolement entre les processus et le matériel.

La virtualisation basée sur un conteneur Docker améliore les performances car il n'y a pas de surcharge de l'hyperviseur.

Modèle Cloudlet basé sur OSGi

Le modèle cloudlet basé sur OSGi s'appuie sur l'interaction de deux Frameworks :
Framework sur la Cloudlet
Le Cloudlet peut être partagé entre plusieurs utilisateurs mobiles ou plusieurs applications nécessitant la séparation des environnements d'exécution. Le système Cloudlet est séparé en différentes zones virtuelles (VM). Le Cloudlet peut télécharger l'image requise à partir du Cloud, mais il peut également construire localement la VM correspondante du périphérique mobile en récupérant d'abord le framework OSGi depuis le Cloud. En utilisant l'interface de gestion, le Cloud assure l'interaction et la gestion du Cloudlet.

Framework mobile
Un Framework OSGi est installé dans le terminal mobile et inclut une interface Cloudlet pour la découverte, ainsi qu'une interaction de service avec le framework OSGi hébergé dans la VM correspondante dans le Cloudlet. Dans ce modèle Cloudlet basé sur OSGi, l'appareil mobile appelle le Cloudlet une fois le Cloudlet découvert. Après une authentification mutuelle, le Cloudlet télécharge depuis le Cloud un sous-environnement de l'utilisateur, ce qui permet au framework OSGi mobile d'interagir avec son image sur le Cloudlet.

Performances et sécurité

Des tests de performances comparant le modèle OSGi dans différents contextes architecturaux, ont permis de montrer qu'une solution SOA peut être une réponse performante et agile pour les services mobiles en cloud.

Enfin dans la troisième partie de cette thèse, nous étudions les problèmes de sécurité dans deux axes principaux:
- sécuriser les clients OSGi mobiles en contrôlant les invocations de services distants dans un contexte Cloud distribué, en adaptant la structure du framework par conception.
- Sécuriser les fournisseurs de Cloud à partir d'invocateurs de services malveillants en exécutant des mécanismes de négociation de stratégies communes entre le fournisseur et le client des services distants

La thèse est conclue par un ensemble de points qui ont été peu ou pas traiter durant ce travail. Les perspectives de travaux futures s'annoncent riches dans un contexte où les objets connectés, la mobilité et la banalisation totale des couches physiques au profits des services virtuels devient le socle de base des services IT.

Contents

List of Figures **18**
1. Introduction **21**
1.1 Introduction 21
1.2 Addressed problems 22
1.3 Thesis Contribution 22
2. Dissertation outline **22**
Chapter 2: Mobile Cloud Computing **25**
2.1 Introduction 25
2.2 Cloud computing paradigm 25
2.2.1 Definition 25
2.2.2. Cloud Service Delivery Models 25
2.2.2.1 Infrastructure as a Service (IaaS) 26
2.2.2.2 Platform as a Service (PaaS) 26
2.2.2.3 Software as a Service (SaaS) 26
2.2.3 Cloud Deployment Models 27
2.2.3.1 Private Cloud 27
2.2.3.2 Public Cloud 27
2.2.3.3 Community Cloud 27
2.2.4 Cloud computing characteristics 28
2.3 Mobile Cloud Computing (MCC) 28
2.3.1 MCC definition 28
2.3.2 A larger vision of MCC 29
2.3.3 MCC components 29
2.4 Mobile Cloud Computing Challenges 30
2.4.1 Mobility constraints 30
2.4.2 Device resources limitation 30
2.4.3 Distributed Application complexity 30
2.5 Conclusion 31
Chapter 3: Service Oriented Architecture **32**
3.1 Introduction 32
3.2 SOA definition 32
3.3 Service as a building unit 33
3.4 Service interaction 33
3.5 SOA and Cloud Computing 34
3.5.1 Accessibility and Interoperability 34
3.5.2 Scalability 34
3.5.3 Service advertising and discovering 35
3.5.4 Dynamicity of service composition 35
3.5.5 Service Management 35
3.5.6 Security 35
3.6 Stat of the art 35
3.7 Conclusion 36
Chapter 04: OSGi Framework **38**
4.1 Introduction 38
4.2. OSGi Implementation 38
4.3 OSGi framework description 39
4.3.1 OSGi Bundle 39
4.3.2 Bundle life cycle 41
4.3.3 Different states of a bundle 41
4.3.4 The module layer 41

4.3.5 The service layer 42
4.4 OSGi framework interactions 42
4.4.1 Bundles framework interaction 42
4.4.2 Bundle-to-Bundle interaction 42
4.5 Benefits of using OSGi platform 43
4.5.1 Complexity management 43
4.5.2 Lightweight framework 43
4.5.3 Optimized life cycles 44
4.7. Conclusion 44
Chapter 05: SOA adaptation to Mobile Cloud Computing 47
5.1 Introduction 47
5.2 Mobile local OSGi integration 47
5. 3 OSGi versus Android platforms 48
5.4 OSGi on Android: AndroLix presentation 49
5.5. A Remote OSGi-based SOA for mobile Cloud services 49
5.6. MCC-OSGi interaction models 50
5.6.1 The mobile platform as a service provider and the Cloud as a client 51
5.6.2 The Android platform as a client and the Cloud as a service provider 51
5.6.3 Mobile platforms as both service provider and client 53
5.7 Modular and lightweight SOA for Android development platforms 53
5.7.1 Motivation for a new service based architecture 54
5.7.2 Andromodule based Architecture 55
5.8 Conclusion 56
Chapter 06: Service Architecture for multi-environment Mobile Cloud Services .. 57
6.1 Introduction 57
6.2. Centric Cloud service oriented design 57
6.3. Distributed M2M Cloud services 60
6.3.1 Application context 60
6.3.2 OSGi-based proposed solution 61
6.3.3 Service advertisement and discovery components 62
6.4. Cloudlet based model 65
6.4.1 Application context 65
6.4.2 VM based Elijah solution 65
6.4.3 Container-based Cloudlet solution 67
6.4.4 OSGi-based Cloudlet model 68
6. 5. Conclusion 69
Chapter 07: Implementation and Performance Analysis 70
7.1 Introduction 70
7.2. Mobile client/Cloud provider performance analysis 70
7.2.1 MobiCloud network connection design 70
7.2.2 Mobile Cloud OSGi integration 71
7.2.3 Performance evaluation 72
7.3 M2M - OSGi versus Andromodules performance analysis 76
7.4. Cloudlet based model 77
7.5 Conclusion 79
Chapter 08: Enhancing MCC Framework Security 81
8.1. Introduction 81
8.2 Security considerations 81
8.2.1 Global challenges 81
8.2.1.1 Server Cloud system 81
8.2.1.2 Network Security 82
8.2.1.3. Data Security and privacy issues 82
8.2.1.4. User and mobile-device authentication 82
8.2.2 Framework issues 82
8.2.2.1 Running environment 82

8.2.2.2 Weak service identification 83
8.2.2.3 Software integrity 83
8.2.2.4 Manifest advertisement 83
8.3 Objectives 83
8.4 Centric MCC framework security 84
8.4.1 Bundle-structure Enhancement 85
8.4.2 Local security policy: module and lifecycle permissions control 85
8.4.4 Bundle lifecycle evolution 87
8.5 Mobile to Mobile framework security 88
8.6 Cloudlet model and integrity check 90
8.8 Conclusion 92
Chapter 09: Conclusion 93
Chapter 10: Publications 95
References 96

List of Figures

Figure 2.1 Cloud Computing Structure
Figure 2.2 Cloud Delivery models
Figure 2.3. Mobile Cloud Service Global Architecture
Figure 3.1. Service interaction
Figure 3.2: SOA functions
Figure 4.1: The different layers of the OSGi framework
Figure 4.2: Bundle manifest file example
Figure 4.3: The different states of a bundle during its life cycle
Figure 5.1. Global design of the Android platform
Figure 5.2. AndroLix – Android interaction
Figure 5.3. Distributed OSGi interactions
Figure 5.4. Global MCC design for OSGi based framework
Figure 5.5. Android as a SP and the Cloud as a client
Figure 5.6. Android as a client and the Cloud as a service provider
Figure 5.7. Dexification process
Figure 5.8. Symmetric interaction (each entity is an SP and a consumer)
Figure 5.9. OSGi AndroLix versus native Andromodules
Figure 5.10. Andromodule based architecture
Figure 6.1. Centric Cloud design
Figure 6.2. OSGi centric Cloud architecture
Figure 6.3. The proposed bundle life cycle
Figure 6.4. MCC global design
Figure 6.5. Mobile-to-Mobile OSGi interaction
Figure 6.6. Service Registry in OSGi
Figure. 6.7 Bundle discovery interaction
Figure 6.8. Cloudlet general design
Figure 6.9. VM Overlay in Cloudlet architecture
Figure 6.10. Elijah based overlay architecture
Figure 6.11. Containers vs VM layers
Figure 6.12 OSGi-based Cloudlet architecture
Figure 7.1 MobiCloud network connections
Figure 7.2. MobiCloud OSGi configuration
Figure 7.3. Felix starting time when increasing the number of bundles
Figure 7.4. Execution times with bundles variation
Figure 7.5 Execution times with dependencies variation
Figure 7.6 Remote bundle invocations
Figure 7.7. Execution-time (in ms) evolution with bundle size
Figure 7.8. Memory consuming
Figure 7.9 Memory consumption evolution with bundle size
Figure 7.10 Bundle execution time (in ms)
Figure 7.11 Docker image
Figure 7.12 Time in seconds for downloading from the mobile device and launching the overlay, OSGi bundle, and Docker layer on the Cloudlet
Figure 8.1. Segregation limitation
Figure 8.2. Enhanced secure bundle

Figure 8.3. Bundle domain protections
Figure 8.4. Local Permission implementation
Figure 8.5. Bundle packages and dependencies control
Figure 8.6: Classical Bundle life cycle
Figure 8.7. Enhanced bundle life cycle
Figure 8.8. Mobile to mobile interaction
Figure 8.9. Mobile to mobile delegation security model
Figure 8.10. MAC–based integrity control for distributed model (without Cloudlet)
Figure 8.11 MAC –based integrity control using Cloudlet model

PART 1: Introduction

Chapter 1: Introduction and addressed issues

1. Introduction

1.1 Introduction

The popularity of mobile devices such as smartphones and, more generally, connected objects are offering ubiquitous communication and information services, creating such a dependency in all domains ranging from our daily life to enterprise services, and even including critical environments (industry, health, military, etc.).
Mobile devices accelerate the emergence of various real-time mobile applications that demand high levels of responsiveness and that, in return, demand intensive computing resources making the complexity of the applications in embedded systems increasing, which introduces the need of new methodologies to reduce the complexity of the software and to supply a support facilitating its re-use.
In this context, Cloud computing has a great potential to bring new mobile application scenarios with simplified local mobile apps. With the power of the Cloud, mobile applications gradually move to the Cloud and move away from being installed and run directly from the devices themselves. Instead, apps are accessed and executed directly from the Cloud through simple mobile interfaces.

Mobile Cloud Computing (MCC) aims to augment computing capabilities of mobile devices, to conserve local resources especially battery, to extend storage capacity, and to enhance data safety so that to enrich the computing experience of mobile users. However, mobile devices with their miniature nature, limited battery, and mobility constraints impose intrinsic limits on their processing abilities, energy and storage capacity.
Hence, the evolution of mobile-devices capabilities and functions gives birth to new views of mobile Cloud computing where mobile devices act not only as Cloud clients consuming services but as service providers as well [1]. Mobile devices, in this context, aim to build collaborative new applications such as information search, data processing, data mining, monitoring, sensing, etc. However, to achieve these scenarios, significant research effort is required because existing mobile Cloud service models are mostly one-directional.

In the future mobile application scenarios, it is highly desirable for information processing to be pervasive, spreading information dynamically over multiple devices. These devices can be part of the system in an open model or managed by the system in a more controlled model. In both cases, systems require middleware that provides the abstraction of the distribution of the system on multiple nodes and offer the ability to use remote services in a dynamic way.
It is often a challenge to deploy middleware on such distributed Cloud systems in a safe, secure, and trusted manner with minimum effort, especially when considering mobility challenges related to mobile networks, to limited mobile resources, to mobile devices with heterogeneous features, to runtime libraries, to operating systems, and to outdated software components that do not include support for remote middleware installation. These challenges are addressable by creating transparency for the usage of the remote mobile resources.

1.2 Addressed problems

In this PhD thesis, we focus on three main problems:

1. **The need for a framework dealing with mobile constraints:** mobile Cloud environment can be a highly heterogeneous environment with software/hardware capabilities and technology variation among mobile devices. In such a Cloud-computing environment, the complexity of the applications integration among multiple Cloud service providers is also increased. Hence, the service management for distributed mobile service providers becomes a critical research challenge.
2. **Environment variation and portability needs:** developing cross-platform components (i.e. Cloud, mobile, and hybrid) for Cloud-mobile applications is a difficult task. In fact, mobile components should be able to move among various smartphones while Cloud components must be portable to all Cloud infrastructures.
3. **Framework security:** another challenging point in Cloud middleware infrastructures is the software integrity. Distributed services are built on multiple nodes and since services are executed remotely, the mobile client has limited means to guarantee for example that the executed service is the right expected one.

1.3 Thesis Contribution

Based on the above discussion, this dissertation proposes a software-engineering solution for Mobile Cloud Computing services including three main contributions:

Contribution 1: We propose to establish a Service-Oriented Architecture (SOA) based on the OSGi (Open Service Gateway initiative) platform for mobile Cloud computing, that supports the Java-based services modules running on both mobile and Cloud platforms. This is achieved by incorporating OSGi into Android software development platform that interacts with Remote-OSGi on the Cloud. We propose the integration of OSGi as a mobile Cloud framework and we explore the suitable models for both client and provider roles.

Contribution 2: we adapt the proposed MCC framework to different architecture contexts. The first context is the traditional centric model, where mobile devices are only consuming services. The second context is the distributed model where the power of mobile-to-mobile interaction offers unlimited value-services opportunities. Finally, the third context is a three-tier architecture that introduces the Cloudlet notion. For each model, we explore the performance of our service-oriented framework and contrast it with alternative existing solutions.

Contribution 3: Securing mobile Cloud computing user's privacy and integrity of data and applications is one of the key issues. In this work, we focus on the security of a distributed framework, and the integrity of MCC services executed within remote entities. We propose to secure the framework, by combining OSGi security concepts, security communication protocols and by including the security in the design of the framework itself.

2. Dissertation outline

The dissertation consists of eight chapters that are organized as follows.

The introduction brings forward and defines the researched problem and the contribution with the proposed Mobile Cloud Computing framework.

The second chapter discusses the state-of-the-art and reviews the corresponding literature related to Mobile Cloud Computing (MCC). It introduces the MCC characteristics and describes the related challenges.

Chapter 3 exposes the Service Oriented Architecture (SOA) concept and explains how SOA can be a key solution to provide distributed mobile Cloud services and how OSGi platform can be an adaptive and efficient framework to provide such implementation.

Chapter 4 introduces OSGi: The Open Service Gateway Initiative. OSGi modularity and dynamism are explained by introducing the concept of *bundles* and service management. We explain in this chapter our contribution and how OSGi can be incorporated as an MCC middleware.

Chapter 5 contains the first contribution of this dissertation. In fact, we address the local mobile context to integrate OSGi and the way to adapt the SOA concept to a basic Cloud interaction: service provider – service consumer. We, then, compare two solutions based on SOA paradigm. The first solution is an OSGi based middleware while the second one is a native Android embedded implementation.

Chapter 6 contains the second part of our contribution that addresses the MCC architecture for multi-environment contexts. Firstly, we present the traditional model for centric Cloud services and then we highlight our SOA-OSGi based framework solution. Secondly, we adapt the model to a distributed Mobile-to-Mobile Cloud computing in order to deal with dynamic mobile and connected objects interaction. Thirdly, we show that our model is also suitable to a 3-tier Cloudlet design, and we explain how this model can fulfil network constraints and deal with hostile environments.

Chapter 7 discusses the performances of the proposed framework solutions according to the proposed design. The proposed solution performance is also compared to different alternative solutions such as Android-based solution, Virtual-Machine (VM) solution, Overlay-based solution as introduced by Elijah project, and Docker containers solution.

Chapter 8 explains the security issues of the proposed framework. Some security enhancements are proposed to improve the security of the MCC framework. For each previous design, a short risk analysis is discussed before out lighting the suitable enhancement and potential impacts.

Chapter 9 is a general conclusion that summarizes the results of this research and discusses further points that should be addressed as a future work.

PART II:
The context and the state of the art

This part is dedicated to the background of this PhD thesis. Hence, it is composed of three chapters, each one is related to a specific aspect required to understand our context.

The first one deals with the concepts of Cloud computing and mobile Cloud computing. The second one describes the features of service-oriented architecture, and the third one details OSGi framework. Each chapter includes a state of the art to refer to existing MCC solutions based on the features described in the chapter.

Chapter 2: Mobile Cloud Computing

2.1 Introduction

The last few years have witnessed a phenomenal growth in the wireless industry, both in terms of mobile technology and its subscribers. With the emergence of high performance mobile networks (LTE 4G, 3G, IEEE 802.11ac, WiFi) the Cloud-computing paradigm appears like the optimal solution to outsource computing and storage and goes beyond the mobile devices limitations. Thanks to Cloud outsourcing and trustful wireless connectivity, mobile devices become the interface with anywhere and anytime services.

To have an understanding of Mobile Cloud Computing (MCC), it is necessary to get a complete grasp on Cloud computing paradigm. This chapter explains the Cloud computing main lines before providing a view for MCC.

2.2 Cloud computing paradigm

2.2.1 Definition

The most widely used metaphor for Internet illustrations is the Cloud. Cloud computing is a computing paradigm that delivers IT as a remote service. The main objectives of this new paradigm are to increase capacity and capabilities at runtime without investing in new infrastructures, licensing new software, or training new recruits. Cloud computing permits customers to utilize Cloud services on the fly in a pay-as-you-use manner.

We can break down a Cloud structure into different layers as in the following. Each layer consists of a level of provided IT services.

- Cloud's infrastructure layer;
- Software layer;
- Platform layer; and
- Application layer.

The Cloud's hardware or infrastructure layer consists of physical servers and network equipment's (like switches, routers, firewalls, etc.) while the Cloud service provider is responsible for running, managing, and upgrading Cloud hardware resources according to the requirements of the users and services. The infrastructure layer is also responsible for allocating hardware resources to users in an efficient, quick, and smooth way. The software layer contains the system software to manage the Cloud hardware resources. The system software permits platforms and applications layers to run and utilize underlying resources.

2.2.2. Cloud Service Delivery Models

Depending on the access level of the user to the Cloud resources, the services may be:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS), or
- Software as a Service (SaaS).

Based on these different layers (see Figure 2.1), the Cloud can provide different remote services: Hardware as a Service (HaaS), Communication as a Service (CaaS), Security as a Service (SecaaS), Business as a Service (BaaS), etc.

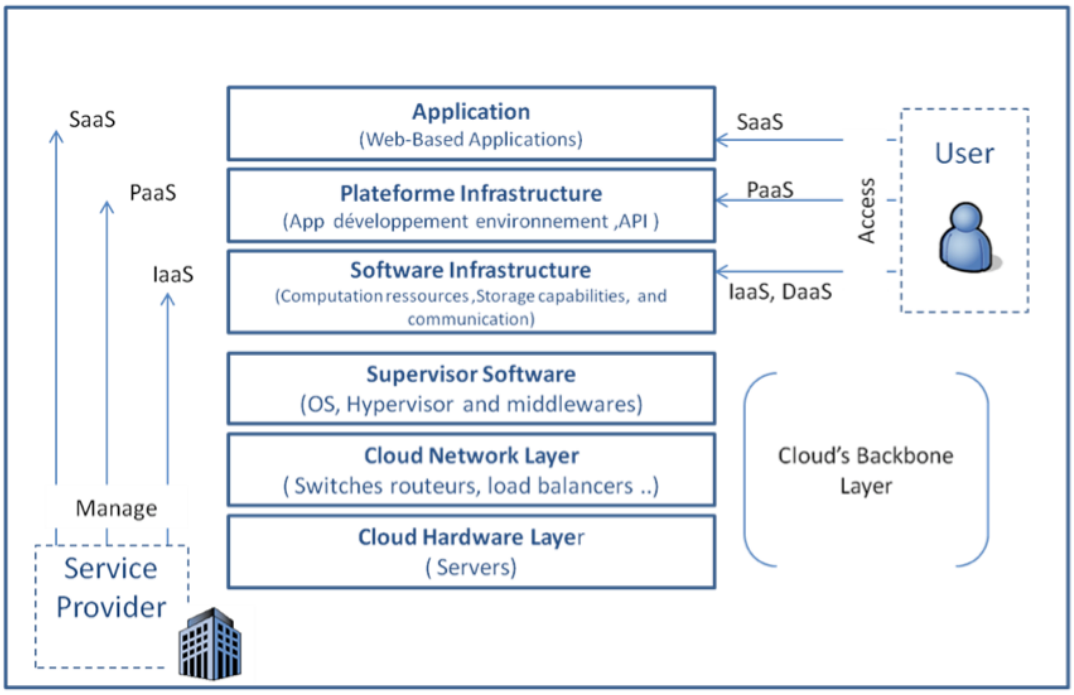


Figure 2.1 Cloud Computing Structure

2.2.2.1 Infrastructure as a Service (IaaS)

This is the basic layer of the Cloud stack. It serves as a foundation for the other layers for their execution. The keyword behind this stack is *Virtualization*. The whole Cloud infrastructure (servers, routers, hardware based load-balancing, firewalls, storage & other network equipment's, etc.) are provided by the IaaS provider. The customer buys these resources as a service on a need basis.
IaaS examples: Amazon Web Service (AWS), Microsoft Azure, Google Compute Engine (GCE).

2.2.2.2 Platform as a Service (PaaS)

The PaaS provider will deliver the platform on the web using a browser access in the most of the cases. As there is no need to download any software, the PaaS definitely empowered small and mid-size companies or even an individual developer to launch their own SaaS leveraging the power of these platform providers, without any initial investment.

PaaS offerings facilitate the deployment of applications or services without the cost and complexity of buying and managing the underlying hardware and software and provisioning hosting capabilities.

Here are some PaaS examples: AppearIQ, Apprenda, AWS Elastic Beanstalk, Cloudera, Google App Engine, IBM Bluemix, Microsoft Azure Web Sites, Salesforce, etc.

2.2.2.3 Software as a Service (SaaS)

In SaaS models, licensing application delivery is based on subscription and customer on demand. Applications and software are centrally hosted in the Cloud servers. SaaS has become a common delivery model for many business applications, including office and messaging software, management software, gamification, virtualization, accounting, collaboration, etc. Local on-premise applications had a very high CapEx (Capital Expenditure). This static delivery model also requires a higher number of skilled developers to maintain the application.
SaaS examples: Google Gmail, Microsoft 365, Salesforce, Citrix GoToMeeting, Cisco WebEx.
Figure 2.2 illustrates the main Cloud IaaS, PaaS and SaaS solutions.

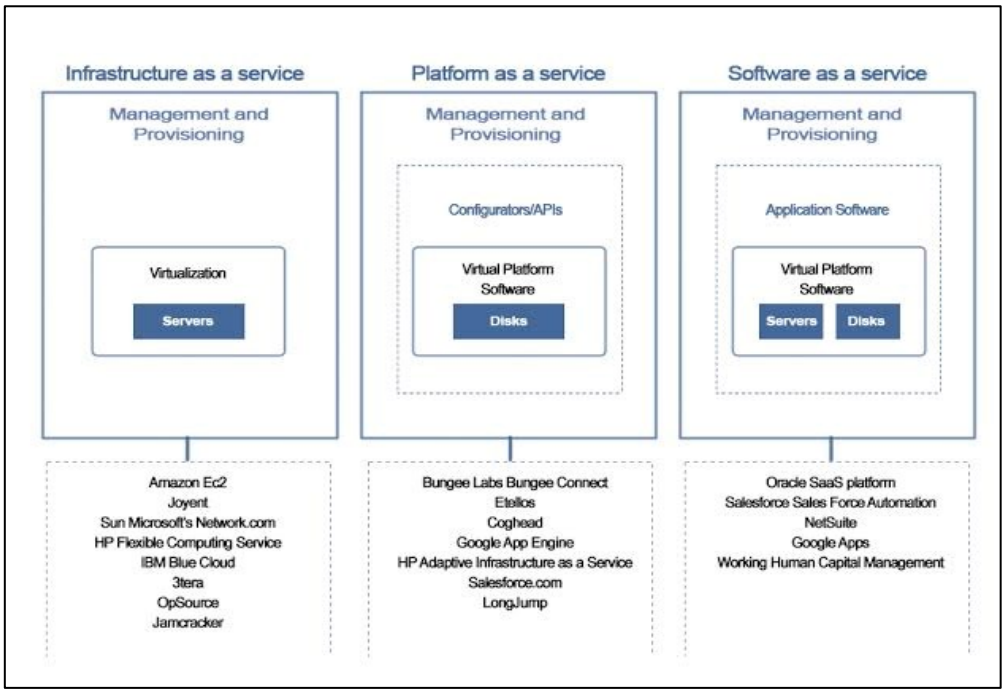


Figure 2.2 Cloud Delivery models

2.2.3 Cloud Deployment Models

The Cloud has three different deployment models and each model has its own benefits and trade-offs. We distinguish three types of models: the private, the public and the hybrid one.

2.2.3.1 Private Cloud

This type of Cloud is setup specifically for an organization within its own data center. The organizations manage all the Cloud resources. The private Cloud offers more security as compared to other models.

2.2.3.2 Public Cloud

This Cloud is available to all external users, through Internet, who want to register with the Cloud provider in order to use Cloud resources on a pay-per-use model. Although security mechanisms are deployed (authentication, access control, IDS, IPS, etc.), the public Cloud is generally less secure then the private Cloud since the public Cloud is accessible through the Internet by any remote user.

2.2.3.3 Community Cloud

The Cloud infrastructure is provisioned for a specific community of consumers that have shared interests (e.g., service policy, security requirements, etc.). According to NIST definition, the community Cloud "may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises" [2].

2.2.3.4 Hybrid Cloud

A hybrid Cloud is a composition of two or more Clouds (private, community or public) that remain distinct entities but are bound together, offering the benefits of multiple deployment

models. As stated in [2] “a hybrid Cloud can also mean the ability to connect collocation, managed and/or dedicated services with Cloud resources”. This is a type of private Cloud, which uses the resources of one or more public Clouds. It is a mix of both private and public Cloud.

2.2.4 Cloud computing characteristics

We identify many features that characterize Cloud computing as in the following.

2.2.4.1 Dynamism

Computing needs change with the capacity usage, the economy impacts and sometimes with the seasonal traffic burst as well. With Cloud computing paradigm, we just need to dynamically configure and size virtual machines and associated computing processing resources to take care of fluctuating demands.

2.2.4.2 Abstraction

Many business units use IT to provide their services but should not worry about security, operating systems, software platforms, updates and patches, etc. With Cloud computing, these chores are left to the IT provider, and a full abstraction is guaranteed on the IT management issues.

2.2.4.3 Resource Sharing

The Cloud architecture is implemented in such a way that it provides the flexibility to share applications as well as other network resources. “This will lead to a need based flexible architecture where the resources will expand or contract with little configuration changes” as stated in [3]. The problem of enabling effective peer-to-peer resource sharing such as capacity planning in a multi user context is an important challenge.

2.2.4.4 Robustness

As stated in [4], “the system as a whole can be made more robust if the system software is spread across VMs”. Robustness is translated within the Cloud thanks to the isolation mechanisms used to confine software failures allowing to avoid a VM software failure to affect another VM.

2.2.4.5 Utilization

With respect to the efficient use of system resources, virtualization can effectively address system bottlenecks that may arise in Cloud environments.

2.3 Mobile Cloud Computing (MCC)

2.3.1 MCC definition

The authors, in [5], define the MCC as: “a service that allows resource constrained mobile users to adaptively adjust processing and storage capabilities by transparently partitioning and offloading the computationally intensive and storage demanding jobs on traditional Cloud resources by providing ubiquitous wireless access”. This vision is correct, however it considers mobile Cloud computing as a one way service on a centric “client/server” model. Mobile Cloud computing cannot be simply illustrated as merging mobile computing and Cloud computing

technologies. The NIST [11] provides a more completed definition of Cloud computing, in [9], as in the following:

From Cloud and Mobility *where Cloud provides ubiquitous, on-demand, elastic, self-configurable, cost effective computing. Mobile devices are accessible, convenient gadgets, with regional wireless communication services and limited data services that have limited computing and power resources.* ***To Cloud Mobility*** *where low-end mobile devices access diverse and scalable cloud computing resources and globally connected mobile enabled resources to receive unlimited mobile application services.*

2.3.2 A larger vision of MCC

In recent developments of mobile Cloud computing [6, 7], mobile devices are involved as Cloud service providers to offer Cloud-based sensing services. For example, a mobile device can sense its surrounding information through various embedded and connected sensors, such as wireless communication channel status, neighboring nodes information, environmental information (e.g., air quality), personal information (e.g., medical and health information using bio sensors), etc. The authors of [8] give a larger vision of what could MCC services look like and how the combination of physical devices capabilities (mobility, sensing, large deployment, etc.) and the virtual entities distributed Cloud services can offer important new perspectives.

Many mobile physical systems are locally considered, and are managed in a single way. With Cloud computing, these systems can be recorded, traced, remotely queried/searched, controlled, reproduced, and analyzed by correlating relevant activities. Enabling these features can greatly enhance the situation awareness of future mobile applications by prediction or customization [8]. Figure 2.3 gives an overview of the general MCC architecture.

2.3.3 MCC components

According to the MCC architecture, we can identify mainly three components as in the following.

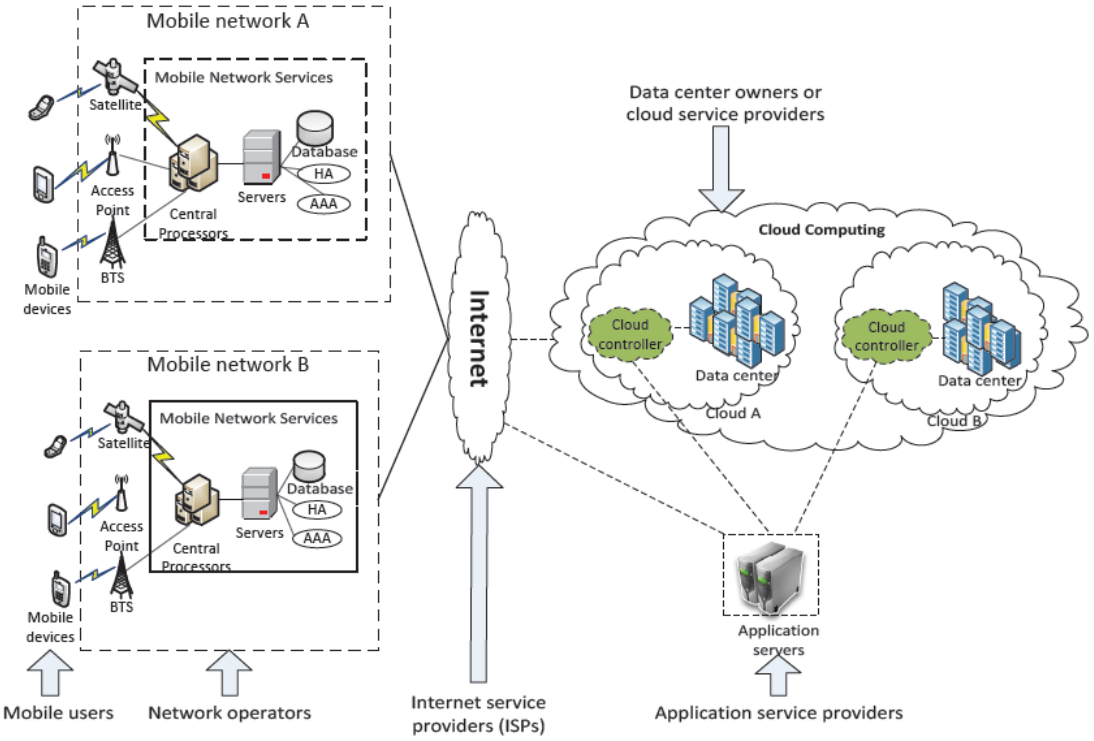


Figure 2.3. Mobile Cloud Service Global Architecture [88]

- **Mobile Users.** Mobile clients interact with a Cloud service provider using native mobile applications or embedded browser applications. Embedded browser applications are developed using standard web development languages (e.g. HTML and JavaScript). Native applications are developed using mobile platform supported programming languages and a set of APIs provided by the Cloud service provider.

- **Mobile devices** are connected to the mobile networks via network access points like 3G/4G BTS (base transceiver station), WiFi access point, or satellite. After establishing functional interfaces connection between the networks and the mobile devices, mobile users join the service layer by sending requests and transmitting information to the **Cloud Virtual Servers.**

- The **Cloud Virtual Servers** provide different remote services to mobile users after an AAA (for authentication, authorization, and accounting) access control based on the provider home agent and subscribers databases.

2.4 Mobile Cloud Computing Challenges

While mobile Cloud computing makes a great contribution to our daily life, it also brings numerous challenges and problems. In short, the core of such challenges and problems is just how to combine the two technologies seamlessly. On one hand, we want to ensure that mobile devices adequately make best use of advantages of Cloud computing to improve and extend their functions. On the other hand, we want to overcome the disadvantages of limited resources and computing ability in mobile devices in order to access Cloud computing with high efficiency like traditional PCs and servers. These constraints, intrinsic to mobility, complicate the design of mobile information systems and require to rethink traditional approaches to information access [10]. In the following, we highlight this complexity through many factors that we try to investigate, such as mobility constraints, device resources limitation and distributed application complexity.

2.4.1 Mobility constraints

Mobile devices are more vulnerable to loss, damage or stole. In addition, mobile connectivity is highly variable in performance and reliability. Although the network performances evolution, the wireless coverage is not uniform. Hence, some areas offer reliable and high bandwidth wireless coverage while others offer a limited wireless connectivity. Furthermore, the same user can move from a well-connected area to a poor connectivity area in an inherent hazardous way.

2.4.2 Device resources limitation

Mobile devices like Smartphones, mobile connected objects, etc. are resource-poor in terms of energy, memory and computation. As stated in [10], “these constraints complicate the design of mobile information systems and require us to rethink traditional approaches to information access”.

2.4.3 Distributed Application complexity

An increasing number of both mobile devices and potential applications are introducing new obstacles for application and service development. Static weighing services and security applications cannot be moved into mobile devices. In addition, as applications become larger and more complex, they become harder to maintain, and difficult to extend without causing their erosion. As mentioned by [12], “there’s a lack of portability among software vendors, making it difficult to reuse or share vendor components, even at the skill level”. In fact, a simple

change in the components implementation can cause breakages throughout the application, at apparently dissociated locations. When updating applications, it's impossible to know whether an interface can be changed without impacting or breaking existing clients.

Due to the software complexity in such a distributed context, software release management is one of the MCC service challenges; as there are several closely related versions of the same utility functions in the applications source code.

2.5 Conclusion

Together with an explosive growth of the mobile applications and emerging of Cloud computing paradigm, mobile Cloud computing (MCC) is seen as a suitable technology for mobile services. MCC aims to overcome obstacles related to the mobile devices performances (e.g., battery life, storage, and bandwidth), environment (e.g., heterogeneity, scalability, and availability), and security (e.g., reliability and privacy) discussed in mobile computing context.

In this chapter, we illustrated the Cloud computing and its fundamentals, before explaining how mobile Cloud computing is more complex than a simple addition of Cloud paradigm and mobile network. We highlighted also our analysis of the main challenges and requirements for an efficient MCC service platform.

The next chapter gives a view of the service oriented architecture (SOA) model, and explains how SOA can fulfill the main MCC service needs. A state of the art of service-oriented MCC is then presented.

Chapter 3: Service Oriented Architecture

3.1 Introduction

Applications and services in IT are becoming more and more important in our society. This pervasiveness, coupled with the arrival of new technologies and increased economic competition, has led to the production of shorter time-to-market applications with growing complexity. However, this augmented complexity has entailed struggling software development activities, resulting in error prone and poorly evolved systems. To circumvent this problem, new methodologies are needed to facilitate design, implementation and maintenance of services and applications, while providing means to capitalize software development. Service Oriented Architecture (SOA) provides answers to these needs by offering effective structuring mechanisms for achieving greater software understanding, reuse and reliability.

Service-oriented architecture is an evolution of the distributed computing based on the request/reply design paradigm for synchronous and asynchronous applications. One very successful approach for handling complexity is modularization. Already from the beginning of computer science, programming languages have facilities to split systems into modules that hide the details that we do not need when we use or reuse the modules.

With the development of wireless access technologies, mobile devices can access to the network core with stable and more bandwidth provisioned. Nowadays, web services have become a universal communication paradigm, and mobile devices with web services are considered as service platforms for the service-oriented architecture. It is intuitive to extend SOA services to mobile devices, which are essential in environments that depend on non-robust connections in mobile environments. By bridging the power of SOA and Cloud computing paradigm in the MCC context, the complexity and the constraints of the systems are addressed thanks to the offered modularity, dynamicity and reusability of the components.

Today, SOA is a set of ideas for architectural design and there are some proposals for SOA frameworks, including concrete architectural languages and software tools to design systems with the SOA paradigm.

In this chapter, we present the SOA-based architecture. The main concept, the requirements and an analysis are given before introducing the state of the art with the main SOA frameworks in the Cloud-computing context.

3.2 SOA definition

In [13], Oasis Group's defines SOA as: “a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations”.

The Open Group's uses another definition: “Service-Oriented Architecture (SOA) is an architectural style that supports service-orientation. Service-orientation is a way of thinking in terms of services and service-based development and the outcomes of services. A service is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports, etc.).”

To be more specific than the mentioned definitions, we choose to adopt the definition inspired by [14] as in the following. In SOA, software applications are built on basic components called services. A service in SOA is an exposed piece of functionality with three properties:

- The interface which is the contract of the service;
- The service can be dynamically located and invoked;
- The service is self-contained, that is, the service maintains its own state.

3.3 Service as a building unit

A service is an integrated set of components such as processes, hardware, software, equipment and staff, which has the ability to satisfy a need or a particular objective. In computer science, the term service means a program or a feature offered by an entity to customers who may also be other entities. For example, on the Internet the various means of information transportation such as Web, e-mail, FTP file transfer are services.

In SOA, services are loosely coupled units of functionalities that are self-contained. Each service implements at least one action, such as executing an online application for a user, sending a result to a remote destination, or modifying information's status.

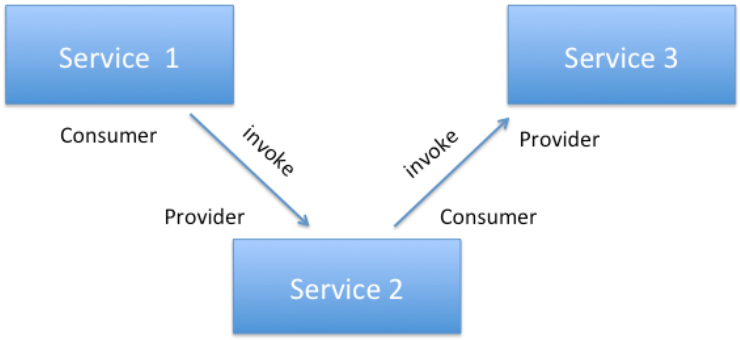


Figure 3.1. Service interaction

3.4 Service interaction

The concept of SOA is based on an architecture that defines an interaction model between three primary entities as in Figure 3.1:

1- **The service provider:** This publishes a description of a service and its implementation.
2- **The service registry:** keeps and updates the arrival or departure of a service and makes it searchable to consumers of services.
3- **The service consumer:** This consults the service registry and invokes the desired services.

Based on these entities, the applications can look for the services in the registry and invoke the required service.

In SOA, users can access services without having any knowledge about the service implementation and its complexity that are abstracted away thanks to the simple interface exposed by the service itself. This is useful for the service consumer that requires service access simplicity and for the service provider that needs to protect the service source code [15]. Thanks to the modularity, SOA allows users to provide dynamic distributed applications built almost entirely from existing software services. In addition, the SOA services use defined protocols that describe how services pass and parse messages using description metadata, which in sufficient details describes not only the characteristics of these services, but also the data that drive them.

SOA services communicate with messages formally defined via XML Schema (also called XSD) and Web Services Description Language (WSDL) is the standard used to describe

the services. There are basically three functions, as illustrated in Figure 3.2, that must be supported in a service-oriented architecture:

1. Describe, publish and register a service;
2. Discover a service; and
3. Invoke, consume and interact with a service.

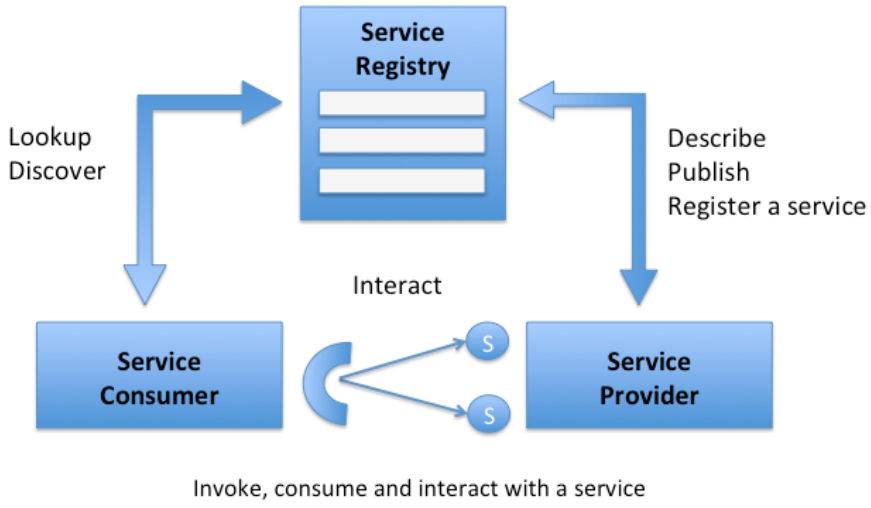


Figure 3.2: SOA functions

3.5 SOA and Cloud Computing

As seen in chapter 2, Cloud computing offers remote resources and capabilities to provide powerful extended, offloaded or distributed services. The delivery of software as a set of distributed services can help to solve problems like software reuse, deployment and evolution. SOA provides design solution to organize service interaction. Thus, combining SOA and Cloud computing paradigm offers an alternative for the main previously discussed Cloud computing challenges through the following described properties.

3.5.1 Accessibility and Interoperability

Services are invoked based on SOA standards. The communication between service consumers and providers typically happens in heterogeneous environments, without any provider knowledge requirement. Hence, the interoperability requires making interactions between the services of the platforms independently of the communication technologies used; even if the most popular approach in this field is Web service technology.

Generally, a specific XML description format is used and connectors to services are created depending on the communication protocol used (CORBA, RPC, Java RMI), thus maintaining an interoperability layer without the use of Web services.

3.5.2 Scalability

SOA, by design, is intended to provide scalability since it allows users to distribute applications to multiple locations and among different servers. However, service dependency i.e -a service that calls out other services or many services that call the same service instance- can present a bottleneck of scalability. SOA framework design must take into account service access sizing and service replication. Another bottleneck in service design is the data access. To overcome this, SOA paradigm proposes a distributed caching in the SOA service-layer implementation.

3.5.3 Service advertising and discovering

The number of Cloud services grows every day. SOA mechanisms and tools give an opportunity to manage and organize the Cloud services. In mobile Cloud computing applications, each entity must be able to discover locally all the services for devices in its context and to detect their presence and disappearance dynamically [16]. Service discoverability needs to be the fully dynamic, reactive, and decentralized, as introduced by services for devices like Jini [17], SLP, or Web Services for Devices (WS4D) like UPnP (SSDP), XMPP or DPWS (WS-Discovery).

3.5.4 Dynamicity of service composition

The service composition refers to an ad hoc composition technology of Web applications that allows users to draw upon content retrieved from external data sources to create entirely new services. The service composition is used in MCC frameworks to integrate service components together on different platforms to offer new services. As stated in [30], one of the main challenges is the way to assemble or compose services in order to obtain a complete and valid application.

3.5.5 Service Management

The service management includes multiple aspects such as:

- *System management*: service monitoring, alerting, exception management, and root cause analysis,
- *Life cycle management*: provisioning, version control, service dependencies and depreciation and
- *Security and policy management*: identity and access management, key management, run-time governance, content-aware networking, etc.

In complex environments, applications may generate millions of service interactions to different providers. In this context, the management of metadata messages while providing dynamicity is a major and complex challenge.

3.5.6 Security

With the Cloud distributed context, security cannot be built only on local applications, since the application exposes a part of its functions as services to other untrusted applications. Indeed, in MCC context, it is necessary to build a trust environment involving the mobile devices and the Cloud.

3.6 Stat of the art

SOA and Cloud computing complement and support each other. Thus, several initiatives attempt bridging SOA and Cloud computing as in the following. Many SOA approaches have been presented for mobile phones such as in [18, 19 and 20]. However, in these works, the service notion is synonym of Web services based on SOAP protocol with the objective of securing mobile communications. Frameworks proposed in [21, 22] have more service-oriented features. Generally, the traditional heavyweight protocols, such as SOAP, WSDL and ESB, etc., are not suitable for mobile Cloud computing environments. Consequently, there is need for constructing a lightweight and flexible mobile SOA solution.

REST has gained more attention compared to other SOA models based on SOAP and WSDL. Szepielak [23] discussed a REST-based SOA ecosystem, but does not focus on mobile service providers. Ennai [24] presented a service-oriented framework for supporting dynamic

service discovery and binding, context-aware service portioning, and asynchronous push service invocation. Natchetoi et al. [25] presented a lightweight service-based architecture for business applications running on J2ME enabled mobile devices.

The Open Service Gateway Initiative (OSGi) [26, 27] framework is a module system and service platform for the Java programming language. OSGi has been designed to provide compact service components, i.e., bundles, which can be used to build a scalable and dynamic service-based solution. Several existing works have been reported on providing the service-oriented using OSGi [28, 29]. In order to extend the OSGi model to distributed systems, a remote OSGi distribution infrastructure is proposed to allow remote accessing OSGi services through a proxy-based approach based on OSGi bundles.

In [32] and [33], the authors describe MCC as a service paradigm where the data processing and storage are moved from mobile resource-constrained devices to powerful and centralized computing servers located in the Cloud. Moreover, with the reliability and high bandwidth offered by current mobile networks like 3G/4G, IEEE 802.11 b-g-n-ac Wi-Fi, users experience is considerably improved. Hence, accessing remote services floating in the Cloud, based on a thin native client or web browser becomes a widespread model for rich applications.

To provide efficiently Cloud services, recent works [34, 35, 36, 37, 38] define the Service Oriented Architecture as an essential foundation for remote Cloud services delivery, thanks to the flexibility and the modularity of this approach.

In SOA paradigm, each service is designed to perform one or more activities by implementing one or more operations. As a result, each service is built as a piece of code. As well stated in [39], "this makes it possible to reuse the code by changing only the way an individual service interoperates with other services that make up the application".

Many SOA approaches have been presented for MCC in a centric approach such as in [40, 41, 42, 43]. However, in previous works, the service notion is synonym of Web services generally based on SOAP protocol. Indeed, Longo et al. [43] propose to support Service-Oriented Computing (SOC) as a new suitable paradigm for Cloud computing, while Wu et al., in [44], refer to a service-oriented development model for Cloud computing as a Cloud-based design manufacturing (CBDM). Frameworks for service consumers are enabled to configure, select, and utilize customized product realization resources and services ranging from computer-aided engineering software to reconfigurable manufacturing systems using technical and functional service descriptions.

Pokahr and Braubach, in [45], propose a component-based framework using Jadex platform. Components are expected to interact by using services offered by other components. The framework manages the distribution of components on Cloud nodes and maps service interaction to an appropriate RPC-based communication in case that components reside on different nodes. Non-functional service descriptions like SLA and QoS are used to provide more elasticity.

3.7 Conclusion

Service Oriented Architecture characteristics can permit to resolve some of the main software engineering challenges in Cloud computing. Modularity is one of the key concepts behind SOA, since SOA breaks up the application into small interacting parts so that these components can be offloaded and run on multiple devices as separate services. With this concept, services in SOA are independent and low coupled, which ensures the scalability of the system even with complex applications.

In addition, with the metadata and interface separation from service resources, the dependencies between services are managed dynamically and pieces of application, i.e., services can be reusable. The service management is based on the service contract to ensure

service subscription and communication agreement. The service encapsulation deals with environment heterogeneity, by allowing any software to be run as a part of an architecture entity. The segregation between the service interface and its content also referred as black-box abstraction, in front of environment complexity, limits knowledge of the service logic to the contract.

To illustrate the service-oriented concept, the next chapter describes OSGi framework which is a Java based SOA middleware.

Chapter 04: OSGi Framework

4.1 Introduction

The previous chapter introduced the SOA paradigm, and its main features. We have seen that the modularity offered by SOA would address the constraints of the Cloud computing system, which is distributed across multiple entities. As observed in the SOA state of the art, the main SOA frameworks do not address the specific constraints related to the mobile environment.

First, the heterogeneity of mobile systems and devices requires a solution that ensures portability across different models and different operating systems. In addition, mobile devices with their limited resources require the adaptation of SOA frameworks to offer much lighter and more dynamic versions. In this optic, OSGi has attracted our attention. OSGi (Open Service Gateway Initiative) is a service platform based on Java language. OSGi introduces a service-oriented architecture that allows a larger independence between the different components of a program. It is introduced by the OSGi Alliance [46] which is an organization founded in March 1999, that includes more than 35 companies' industrials and laboratories (IBM, Nokia, Philips, BMW, Motorola, Sun Microsystems, etc.).

The first motivation of the OSGi Alliance was to define an administration model of bridging facilities in which the administration is transparent to the user beneficing of the offered services. Initially designed for home network and sensor network environments, (which explain its lightness) OSGi quickly covered many application contexts such as automation, automotive, telecommunication, and industry.

In this chapter we will present in more details, the OSGi framework, its characteristics and main advantages. We also study the state of the art of OSGi-based solutions in the context of Cloud computing and particularly in mobile Cloud computing.

4.2. OSGi Implementation

The OSGi platform is an execution platform running under Java virtual machine. There are many implementations of the OSGi platform nowadays, and among these we have:

- **Eclipse Equinox** [47]: is the most widely used platform due to its use in Eclipse IDE. It implements the version release 4.1 of the OSGi specification under the EPL (Eclipse Public License) license.

- **Knopflerfish** [48]: is the most popular and mature of implementations of OSGi versions release 3 and 4.1 with a graphic interface to visualize the bundles and the services. This implementation is developed and maintained by Makewave AB, a Swedish based company under the BSD-style license.

- **Felix** [49]: A communitarian implementation of the version release 4.x. It is characterized by the reduced memory footprint, which makes it an ideal candidate to the embedded field. It is delivered under the Apache License.

- **ProSyst** [50]: ProSyst is an OSGi certified, low-footprint implementation compliant to OSGi release version 4.2, 4.3, 5.0. It is optimized for the use in commercial embedded and resource constrained platforms and products.

Various development platforms, which have distinct goals, use OSGi internally to implement their solutions, such as the following products:

- IBM WebSphere Application Server
- Oracle (formerly Sun) GlassFish Application Server
- Eclipse Virgo (SpringSource dm Server)
- JBoss Application Server
- Apache Camel
- Apache Sling
- Apache ServiceMix
- Apache Karaf.

4.3 OSGi framework description

As shown in Figure 4.1, the OSGi framework is composed of different layers that are described in this sub-section.

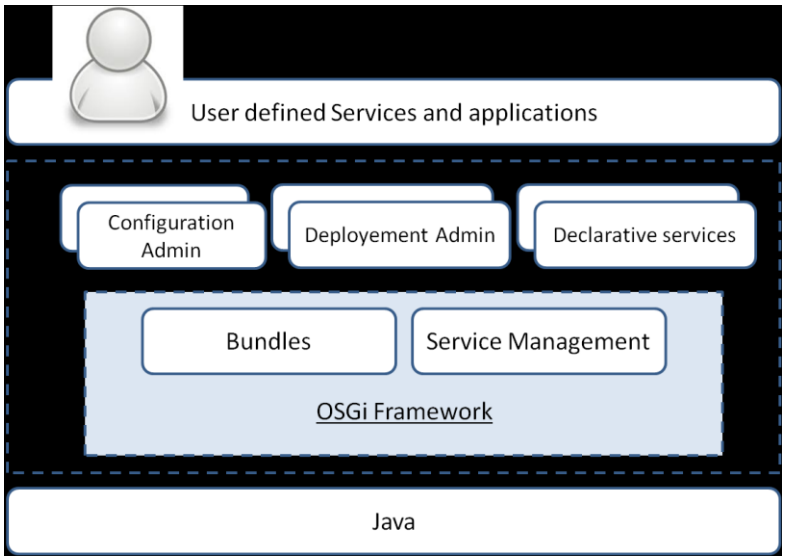


Figure 4.1: The different layers of the OSGi framework

4.3.1 OSGi Bundle

A bundle is a logical unit in the form of a Jar file including the compiled code, service resources and meta-data. The meta-data is stored in the *Manifest.mf* file. The *Manifest.mf* file is used by the framework to define the requirements and dependencies of the bundle as well as the packages and shared services. A bundle can expose its interface without exchanging the resource classes. Figure 4.2 shows an example of a Manifest file.

In Java context, a bundle is a plain Jar file. However, while in standard Java everything in a Jar file is completely visible to all other Jar files, OSGi hides everything in the Jar file unless explicitly exported.

Table 4.1 summarizes the main standard headers included in the Manifest file and offered by the OSGi specification:

Header	Description
Bundle-ManifestVersion	Bundle version
Bundle-Name	Bundle name
Bundle-SymbolicName	Symbolic naming of the bundle
Bundle-Version	Bundle version number
Bundle-RequiredExecutionEnvironment	Required execution environment
Import-Package	Imported packages along with their required version numbers
Export-Package	Exported packages
Bundle-Vendor	Bundle provider
DynamicImport-Package	Specifies the names and the versions of the packages that are used by the bundle. This header is different from the *Import-Package* by the fact that it's not required for the dependencies to be presented at the launching of the bundle. They are only required to be present during the execution
Bundle-NativeCode	List of the used native libraries
Require-Bundle	Identifies the required bundles for the execution
Bundle-Activator	Specifies the name of the class implementing the *interfaceBundleActivator* which defines the treatments to be performed at the starting and the stopping of the bundle
Bundle-Classpath	Specifies the bundle classpath

Table 4.1: The different headers offered by OSGi

```
Bundle-Name: HelloWorld
Bundle-Description: bundle qui affiche hello World au démarrage et Good Bye World à son arret.
Bundle-Version: 1.0.0
Bundle-Activator: tutoriel.exemple1.Activator
Import-Package: org.osgi.framework
```

Figure 4.2: Bundle manifest file example

4.3.2 Bundle life cycle

The cycle of life layer (lifecycle) defines how the bundles are dynamically installed and managed in the OSGi framework. The life-cycle layer describes the operations applied to a bundle, which allow the administration and the development of bundles in a well-defined and dynamic manner. Bundles may be added or deleted without the need for the framework reboot. These operations are: install, update, start, stop and uninstall. The life-cycle layer also defines how bundles access to their execution context and interact with the framework.

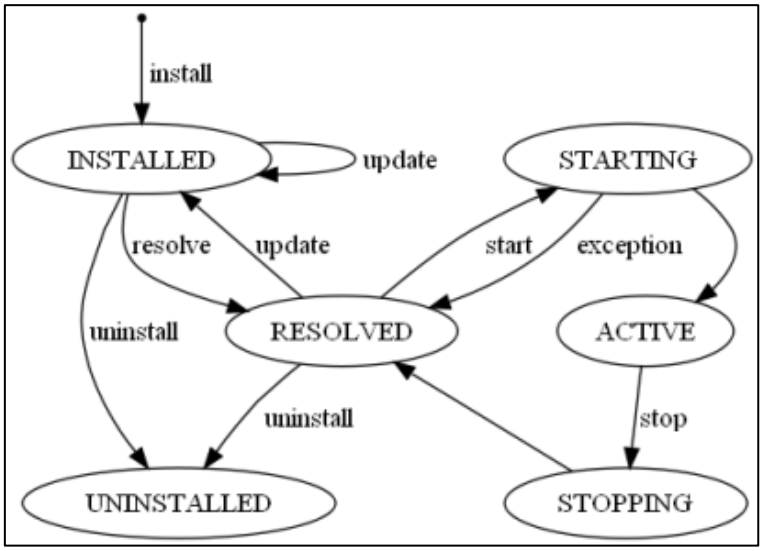


Figure 4.3: The different states of a bundle during its life cycle

4.3.3 Different states of a bundle

As shown in Figure 4.3, there are different states that a bundle can traverse:

- ***INSTALLED STATE***: after being installed, a bundle passes to the *INSTALLED* state. The installation consists in downloading the Jar file from the bundle and put it in the cache directory of the bundles in the platform (Bundle-cache).
- ***RESOLVED STATE***: as soon as the dependencies of the bundle are satisfied, it goes automatically to the state of *RESOLVED*. The bundle stays in this state as long as no command has been addressed.
- ***ACTIVE STATE***: when the bundle receives a starting command (start), it goes to the next state *STARTING* then implicitly to the *ACTIVE* state, in which the bundle is ready to be used.
- ***STOPPING STATE***: when a bundle receives a stopping command, it goes into the *STOPPING* state then automatically to the *RESOLVED* one.
- ***UNINSTALLED STATE***: if an uninstall command is given, the process begins and the bundle goes into the *UNINSTALLED* state and the associated Jar file will be deleted from the bundles cache of the platform.

4.3.4 The module layer

The module layer defines how a bundle can import and export code. This layer also allows the management of bundle dependencies regarding the libraries and the matching versions. OSGi manages the coexistence of different versions of the same library in the container, thanks to the creation of an independent classloader for every component or bundle operations.

4.3.5 The service layer

The OSGi services are considered as native applications of the OSGi framework. OSGi services can belong to different categories as in the following:

- *Transversal functions* which are mostly always required, like logging service, monitoring and configuration services.
- *Protocol related services*, which could be used by the application, like HTTP, FTP, SMTP services.
- *Mandatory services* which are inherent to the framework like bundle wiring which manages the dynamic module system itself, and the start-level service which manages the bootstrap process of the framework.

The providers offer services via the interfaces declared as visible and the consumers may use the exposed services without having access to their implementation in a dynamic way, which is the strength of OSGi.

4.4 OSGi framework interactions

4.4.1 Bundles framework interaction

Each time the methods **Start** and **Stop** of the bundle **Activator** are called, the framework sets a **BundleContext** variable as a parameter. Every interaction with the framework is performed through the *BundleContext*, and the only way to access the context of a bundle is to implement the *BundleActivator*. The list below contains some of the possible interactions between the bundle and the framework via the *BundleContext*:

- Find a bundle using its ID.
- Install new bundles.
- Handle other bundles by programming commands, i.e., start them, stop them, install them, uninstall them, update them, etc.
- Save and restore a file in a storing space managed by the framework.
- Obtain the list of the installed bundles.
- Register or delete a **Bundle Listener** that indicates the state of any bundle in the framework.
- Register or delete a **Service Listener** that indicates the state of any service in the framework.
- Register or delete a **Framework Listener** that informs us about the different events on the framework.

4.4.2 Bundle-to-Bundle interaction

OSGi is a technology with the objective of applying the principles of the component-oriented programming and service-oriented architecture. In order to implement the modularity philosophy, it provides a highly evaluated interaction system between the bundles based on the packages Import and Export. The packages are the exchange unit between the bundles.

There is no sharing by default between OSGi bundles. The package interaction is based on a publish/subscribe model where the exchange approach is based on the client/server model. This means that a bundle may be a server that provides services or a client that requires services.

The same bundle can acts simultaneously as both server and client for different services. To be able to export or import, a bundle has to specify packages in the *Manifest.mf* file by using the entrances **Import-Package** and **Export-Package**.

a) Export contract

By default, a bundle is invisible to the outside and no other bundle has access to its services. But with the exportation, only the packages being specified in the Export-Package of the *Manifest.mf* file will be visible in the framework without even giving access to their sub-packages. The complete syntax for a given value for the Export-Package property is the following:

Export-Package: package-name; **parameter**:=value, package-name; **parameter**:=value,..

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-SymbolicName: helloworld
Export-Package: manning.osgi.helloworld
```

In this case, the bundle *helloworld* is exporting all of its public classes located in the package *manning.osgi.helloworld*.

b) Import contract

During the importation of packages, the bundle has to specify in the Import-Package field of the *Manifest.mf* file all the desired packages to be accessed, but only if these packages are exported by the bundles that contain them, if not, an error will be reported.

The complete syntax for a given value for the Import-Package property is the following:

Import-Package: package-name; parameter:=value, package-name; parameter:=value,...

To import a Java package, a bundle uses the manifest header Import-Package as in the following:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-SymbolicName: manning.osgi.client
Import-Package: manning.osgi.helloworld
```

The bundle *helloworld.client* is importing all public classes in the package *manning.osgi.helloworld*.

4.5 Benefits of using OSGi platform

4.5.1 Complexity management

Without the modularity offered by service-oriented design, a simple change to the implementation of one component causes breakages throughout the application, in a distributed context where bundles are installed on different frameworks. Thanks to the bundles and modules definition, OSGi decreases the software complexity by efficiently modularizing the code while dealing with smaller problems.

4.5.2 Lightweight framework

Many service frameworks are considered as too heavy solutions for resource restricted environments because the products try to be a one-size-fits-all solution to all the requirements of all businesses [51]. OSGi provides a lightweight basic framework, where services can be added in a customized way.

4.5.3 Optimized life cycles

Generally, iterative development life cycle becomes slow, and the programming model becomes complex, since the developers deal with more APIs than they really need to use. With OSGi flexibility and modularity, life cycle of new features can be shortened.

4.6 OSGi frameworks for Cloud computing – State of the Art

Wu et al., in [60], propose a framework called POEM, based on virtualization, to offload, compose and migrate images of the mobile devices to the Cloud. The mobile user's apps can use the local image as well as the remote one hosted on a dedicated VM within the Cloud. The authors implemented the framework using OSGi and XMPP technologies.

Giurgui et al. [52] developed an application middleware named AlfredO, to distribute different layers of an application between the device and the server. AlfredO is based on OSGi and allows developers to decompose and distribute the presentational and logic tiers between a mobile device and a server.

Pokahr and Braubach, in [53], propose a component-based framework using Jadex platform. Components are expected to interact by using services offered by other components. The framework manages the distribution of components on Cloud nodes and maps service interaction to an appropriate RPC-based communication in case that components reside on different nodes. Non-functional service descriptions like SLA and QoS are used to provide more elasticity.

In the context of Cloud computing, the OSGi Alliance in [54] investigates the way to use OSGi for Cloud-based applications. Schmidt et al. [55] presented OSGi for the Cloud (OSGi4C), a solution that combines OSGi and P2P overlay called JXTA. The authors addressed the bundle process selection issue when several choices are possible. Thanks to a P2P based mechanism, remote available bundles are detected and evaluation mechanisms permit a dynamic selection. Rellermeyer et al. in [56, 57] took advantage of the concepts developed for centralized module management, such as dynamic loading and unloading of modules, and showed how they can be used to support the development and deployment of distributed applications by integrating R-OSGi as a plugin within Eclipse IDE.

In [58], the authors outlined a Distributed OSGi (DOSGi) architecture for sharing electronic health records using public and private Clouds, which overcomes some of the security issues inherent to Cloud systems. Researches in [59] proposed to use OSGi as the architecture foundation of a client agent and use a peer-to-peer infrastructure to provide, share, and load OSGi bundles at runtime.

EZdroid project [61] is an OSGi-based solution that does not require the recompilation of the application during bundle updates and provides administrative functions using the OSGi in Android platforms. Wu et al. in [56] provide a solution where an OSGi-based architecture for a smart-home environment is established through common web services.

4.7. Conclusion

Originally introduced for embedded systems, OSGi framework is natively a lightweight system that matches mobile devices and embedded systems limitations. The OSGi framework allows dynamic deployment and un-deployment of applications without altering the local running applications and the remote depending ones thanks to the SOA modularity. Thanks to the transport abstraction through OSGi remote services and dependency management, we identified in OSGi platform an excellent support for dynamic services oriented design based on a Platform as a Service (PaaS) model. In addition applying OSGi to the MCC context can offer two important advantages:

- OSGi offers system updates easily and efficiently, without generating full Cloud VM images.
- OSGi also avoids lock-ins, because applications can be designed and validated using any OSGi implementation in a standalone environment before moving to the Cloud.

PART III:
Contribution

This part is dedicated to our contribution that includes four chapters. Chapter 5 focuses on the mobile side by designing an OSGi-based solution and a native Android solution.

Chapter 6 extends the OSGi-based approach to multi-environment Cloud computing in order to build an OSGi solution to MCC environment while handling mobile to mobile, mobile to Cloud and mobile to Cloudlet paradigms.

Chapter 7 analyses the proposed solutions by evaluating their performances and by comparing them with existing well known solutions.

Chapter 8 improves the OSGi based architecture for MCC with security-by-design features.

Chapter 05: SOA adaptation to Mobile Cloud Computing

5.1 Introduction

This chapter of this PhD thesis presents the vision of the MCC environment and the large role that mobile devices can play to provide rich services. It explains how service-oriented approach and particularly OSGi framework can be a suitable choice to support mobile Cloud software development and service provisioning while ensuring more control, re-use and reliability of the software. OSGi has proved to offer flexibility and reusability, due to the independency, interoperability and loose coupling of the SOA paradigm and the Java inherited portability. This chapter aims to outline our solution by integrating SOA design in the MCC context.

This chapter is composed of two main sections. The first section explains the OSGi integration to MCC environment including three main steps:

- Integration of OSGi in local mobile platform;
- Integration of OSGi in a remote context; and the
- Adaptation for the different interactions in MCC.

The second section discusses the SOA concept integration into Android mobile platforms, and introduces an experimental integration of OSGi concepts and primitives directly in the mobile operating system.

The main research goal, in the first part, is to establish an MCC-OSGi middleware that supports the Java-based services modules (i.e., bundles in OSGi terminologies) running on both mobile and Cloud OS platforms. To this end, we have developed the MCC-OSGi by incorporating OSGi into Android software development platform that interacts with Remote-OSGi on the Cloud. In the current chapter, we will describe how to extend the use of OSGi bundles from the local execution environment to a distributed mobile Cloud environment so that the mobile Cloud architecture is based entirely on OSGi service model. Two main contributions presented in this chapter are:

- The establishment of OSGi-based SOA architecture in the mobile Cloud computing environment by integrating R-OSGi bundles that support services discovery, selection and deployment of bundles.

- The integration of OSGi framework into Android mobile devices so that the MCC-OSGi bundles can run in Java Running Environment (JRE) without compatibility issues on both Android based mobile devices and Cloud virtual machines.

The second part aims to explain how incorporating SOA concepts and OSGi primitives directly in the mobile operating system can present an axis of simplification by reducing the generated overhead of mobile middleware.

5.2 Mobile local OSGi integration

5.2.1 The Android platform presentation

Android is an open source operating system based on Linux and dedicated to mobiles phones, initially designed by the Android start-up and then acquired by Google. The Android platform is organized around various layers. An application layer supplies standard applications

such as SMS application, a browser, a calendar, etc. Every application runs on a distinct virtual machine called Dalvik Virtual Machine (DVM) to avoid altering the functioning of the other applications. The executable files .dex (Dalvik Executable) are generated from class files. Android contains a framework layer that facilitates the application development based on several components (see Figure 5.1) such as GUI management (Views), data sharing (Content Provider), resource access (Resource Manager), life-cycle management (Activity Manager), etc. This framework is based on C/C++ libraries and on Android runtime to provide Java basic functionalities.

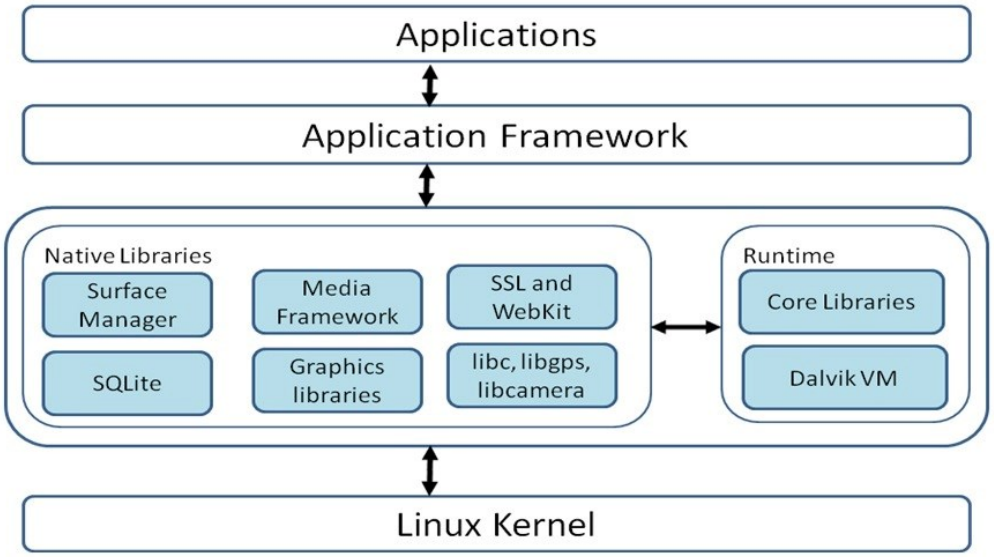


Figure 5.1. Global design of the Android platform

In Android platforms, a service is a component that runs in background. It is defined by an ".aidl" specification called Android Interface Definition Language (AIDL), from which a Java interface is automatically generated with an abstract Stub class. A service class contains an internal class extending Stub classes. To make the service available to other applications, the service-class name must be presented in the service entry of the Manifest file. Additionally, Android introduces the activity notion to define a treatment associated with a physical view to interact with the user. The activity is the entry point of the application through a GUI. And the Intent messaging is a facility for late run-time binding between activities in the same or different applications.

5. 3 OSGi versus Android platforms

In Android platforms, a DVM focuses strictly on mobile devices whereas JVM is designed to be a one-size-fits-all solution [65]. The choice of Android to use multiple JVM instances, one for each process, is one of the most major differences with the OSGi framework where the whole framework runs on one JVM. Thus, duplication of components in DVMs appears as one of the main drawbacks of Android platform. With a single VM in OSGi, the classes are shared and for each bundle is associated a distinct class loader that allows selective export of internal packages and selective import of required dependencies. Thanks to component-based model, designing applications is possible by assembling components and by re-using existing OSGi components (bundles). Loading classes dynamically is another major part thanks to the life cycle. The dynamic bundle management offered by OSGi permits updating easily and managing bundles without restarting the Android application.

5.4 OSGi on Android: AndroLix presentation

As published in [62, 64], we designed a middleware called AndroLix that provides an inter-OSGi communication through the Android Interface Definition Language (AIDL) in mobile Android platforms.

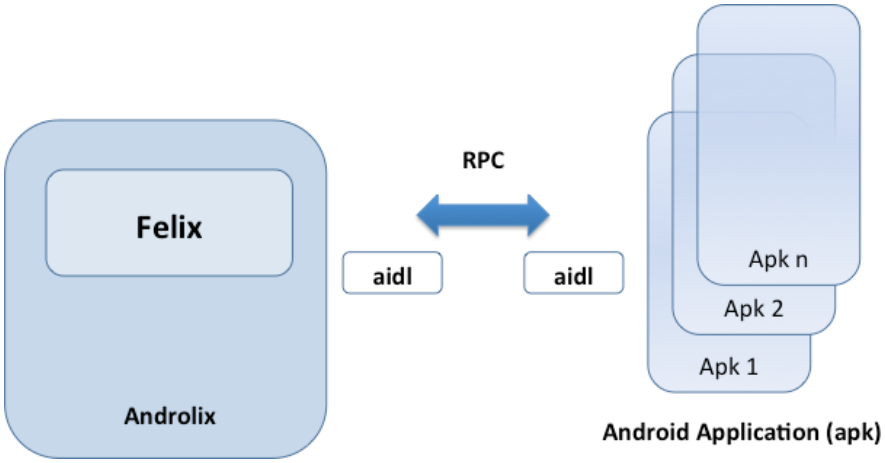


Figure 5.2. AndroLix – Android interaction

The access to Felix is made via a RPC "remote procedure call" communication mechanism as depicted in Figure 5.2. AndroLix provides to clients an AIDL description containing different methods that are implemented by this service. In AndroLix, each Android application is defined as a couple composed of a Felix OSGi bundle and an apk Android application. The bundle specifies in Import-package entry not only the packages it requires but also the packages needed by the corresponding apk. The role of the associated bundle is to represent the application within the Felix container, that is, the bundle guarantees the dependencies resolution of the apk application before its installation, and may access to other Felix bundles if they are required by the apk application.

5.5. A Remote OSGi-based SOA for mobile Cloud services

In the AndroLix previously discussed, OSGi is integrated to Android platforms for local execution. Our first contribution is to extend this model to Cloud computing context. The provider and the consumer of a service are located in two separated nodes, and the communication between the two nodes is ensured via a wireless network. To allow interaction between Android platform bundles (service consumer) and the Cloud services (service provider), we use remote OSGi: R-OSGi introduced in [66]. R-OSGi is a distributed middleware platform that extends the centralized, industry standard OSGi specification to support distributed module management. We used R-OSGi as a bundle on the Cloud side and we integrated the related bundle within Android platforms as a proxy bundle service.

As shown in Figure 5.3, the client bundle performs a remote connection to the service through the R-OSGi bundle that calls the remote service and checks its existence before starting the application execution. The service client framework takes care of importing the remote service by establishing a local proxy endpoint, which is an instantiation of the R-OSGi bundle, and connecting it to the remote endpoint. R-OSGi allows a centralized OSGi application to be transparently distributed at service boundaries by using proxies. The R-OSGi proxy is indistinguishable from local OSGi services. The R-OSGi protocol on the proxy is used to make remote invocations to the original service, which is located on the Cloud node, and events from Cloud server are transparently forwarded to the mobile device and occur as if they were issued by a local bundle.

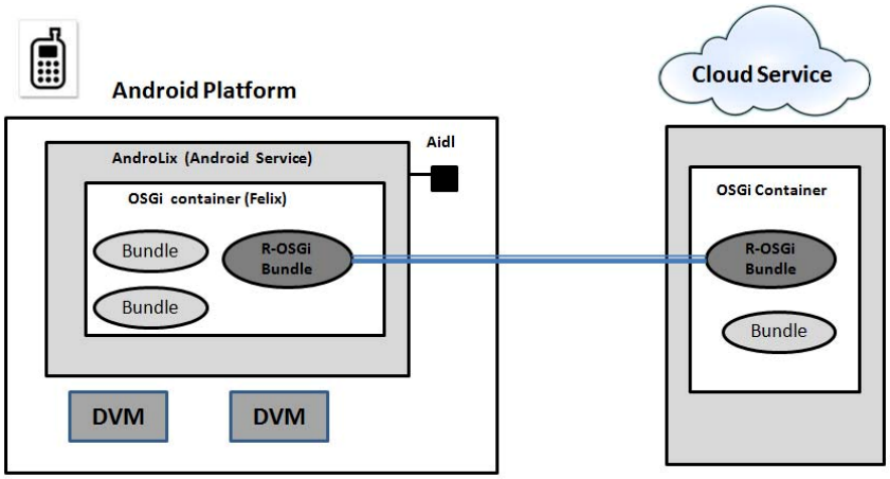


Figure 5.3. Distributed OSGi interactions

Hence, the OSGi bundle is considered as a Cloud service's deployment unit. The Cloud provider exports the services that can be managed in a distributed and autonomous fashion by the OSGi bundles management system. A service is set up as a remote service by specifying it with the following service properties:

service.exported.interfaces, and

service.exported.configs.

The first property determines which service interfaces are to be exported in a remote manner, using the following method: *props.put ("service.exported.interfaces", "*");* and the second property used as the first parameter of the following method specifies how those interfaces can be made remote through the following method:

```
props.put("service.exported.configs","org.apache.ws");
```

5.6. MCC-OSGi interaction models

Based on the fact that OSGi framework is integrated within Android mobile platforms and the inter-connection between OSGi frameworks is resolved (see Figure 5.4), we have identified three service architecture models depending on whether the service provider is the mobile platform or the Cloud.

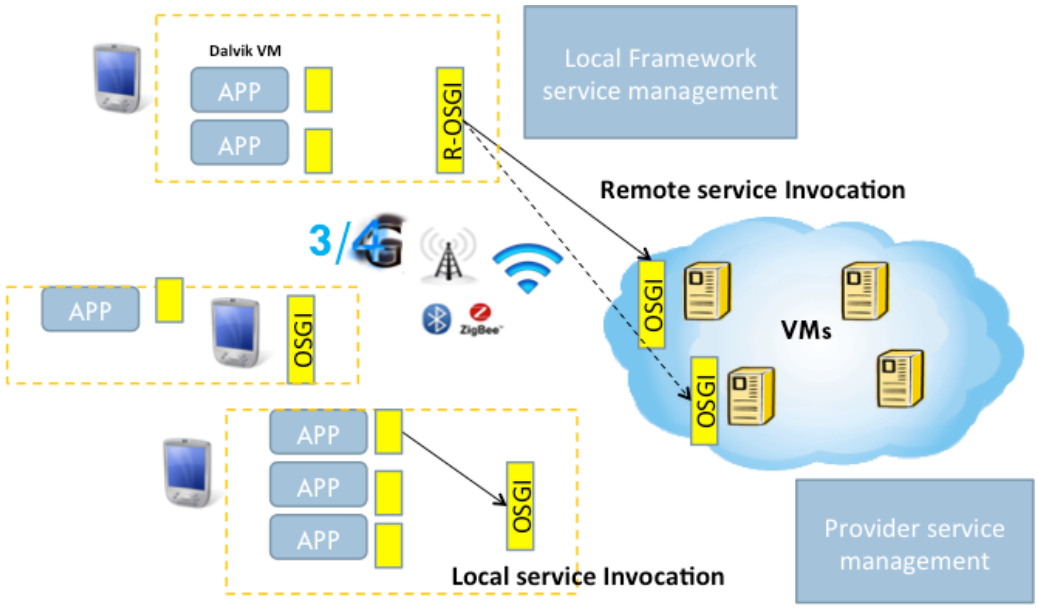


Figure 5.4. Global MCC design for OSGi based framework

5.6.1 The mobile platform as a service provider and the Cloud as a client

As illustrated in Figure 5.5, when the mobile platform acts as a Service Provider (SP) vis-à-vis the Cloud, the remote clients are executed on the Cloud. The execution bytecode on the client side does not need a DEX service adaptation that is required by the Android platform and only OSGi bundles are converted to .dex files in order to be executed in the DVM as a Dalvik code. We have then to add the Dalvik classes of the Jar file to make it executable on Android platforms. The service model defined here is suitable for mobile Cloud context where mobile devices provide sensing services to the Cloud.

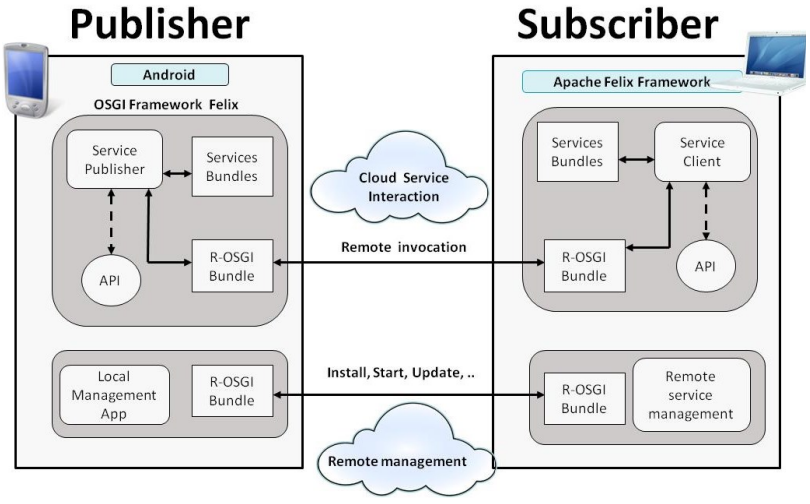


Figure 5.5. Android as a SP and the Cloud as a client

5.6.2 The Android platform as a client and the Cloud as a service provider

As shown in Figure 5.6, the second situation is a little more complex because when calling a service, the client gets the interface description of the service in the form of a bytecode. The client parses it and creates a bundle proxy bytecode that has to be run on the Android platform. As Android applications are composed of Dalvik codes, we change the original R-OSGi bundles

and add patches to transform dynamically the Java bytecode to a Dalvik bytecode based on the following approaches:

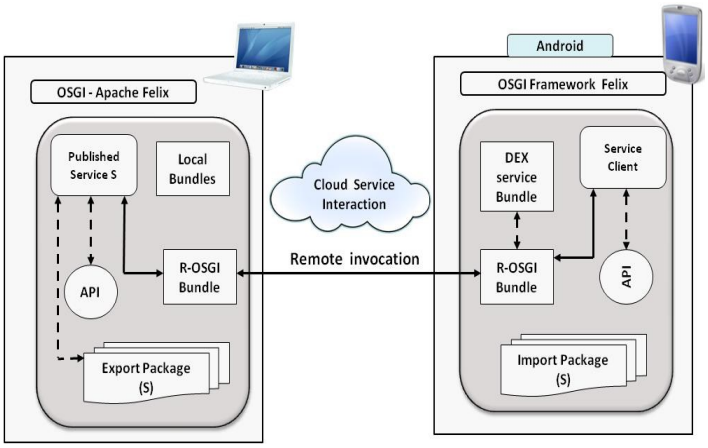


Figure 5.6. Android as a client and the Cloud as a service provider

Step1: Usually R-OSGi uses the ASM library to parse bytecode and creates the proxy object. In our case, we have added the Dalvik classes of the Jar file to make R-OSGi executable on Android platforms.

Step2: As the object created via ASM is always a bytecode, we have to convert it into a Dalvik code. We thus add a dexification service that we can integrate as a bundle in the container or directly in R-OSG. Here is the source code for dexification.

```
→ import com.android.dx.command.dexer.Main;
→ public class DexServiceImpl implements DexService {
→ public byte[] createDexedJarFromFile(String filename) {
→ byte[] dexed = Main.createDexJar(filename);
→ return dexed; }
→   }
```

Step3: Finally, we have to integrate the DexService in the R-OSGi creation process. For this purpose, we need to patch R-OSGi in order to invoke the dexification service when creating the bundle proxy. This dexification process, as in Figure 5.7, has been facilitated thanks to the development project of [65]. After all of these modifications are performed, the R-OSGi bundle can be executed on the Android mobile device to access remote services.

The service model described here allows mobile devices to outsource their computation to the Cloud, by executing services hosted on the Cloud.

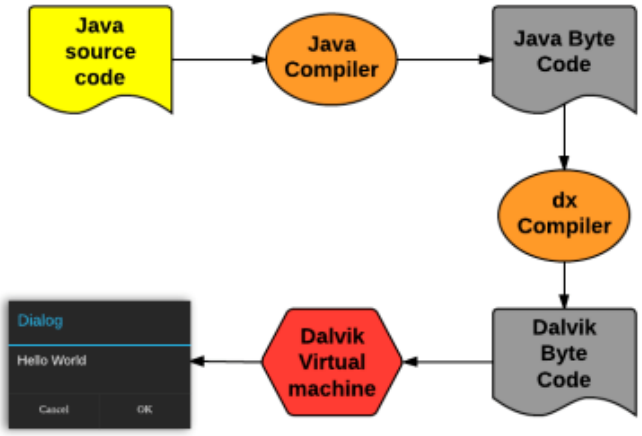


Figure 5.7. Dexification process

5.6.3 Mobile platforms as both service provider and client

As shown in Figure 5.8, the R-OSGi bundle patched with the dexification process is necessary only on the client side. This model can be used in a device-to-device architecture, where each mobile device can be a service provider as well as a service consumer. In addition, MCC-OSGi adopts an XMPP (Extensible Messaging and Presence Protocol) based solution, in which an XMPP server is used as a discovering service between different OSGi devices.

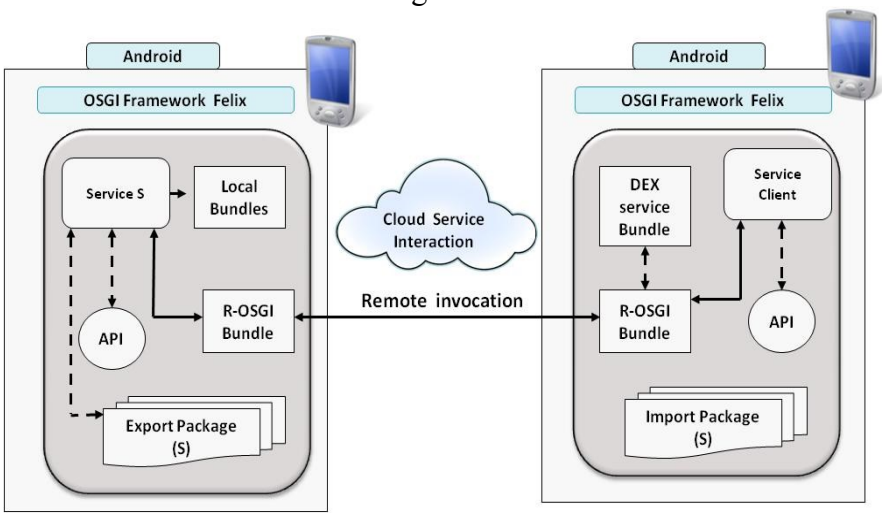


Figure 5.8. Symmetric interaction (each entity is an SP and a consumer)

5.7 Modular and lightweight SOA for Android development platforms

The previous OSGi Felix solution tries to provide modularity by incorporating frameworks as intermediary layers between the application and the operation system. In fact, in that solution, to execute the services of a local bundle, the Android application (apk) triggers a bundle proxy by running Android API, then the proxy asks the OSGi middleware that calls the requested bundle. The bundle seeks for elementary packages of the Java virtual machine, and then the mobile OS layer triggers the physical capacity. This execution process enables negative impacts in terms of end-to-end applications performances. This motivates us to make a new proposal as in [68] to incorporate OSGi-based SOA primitives directly into Android OS layer.

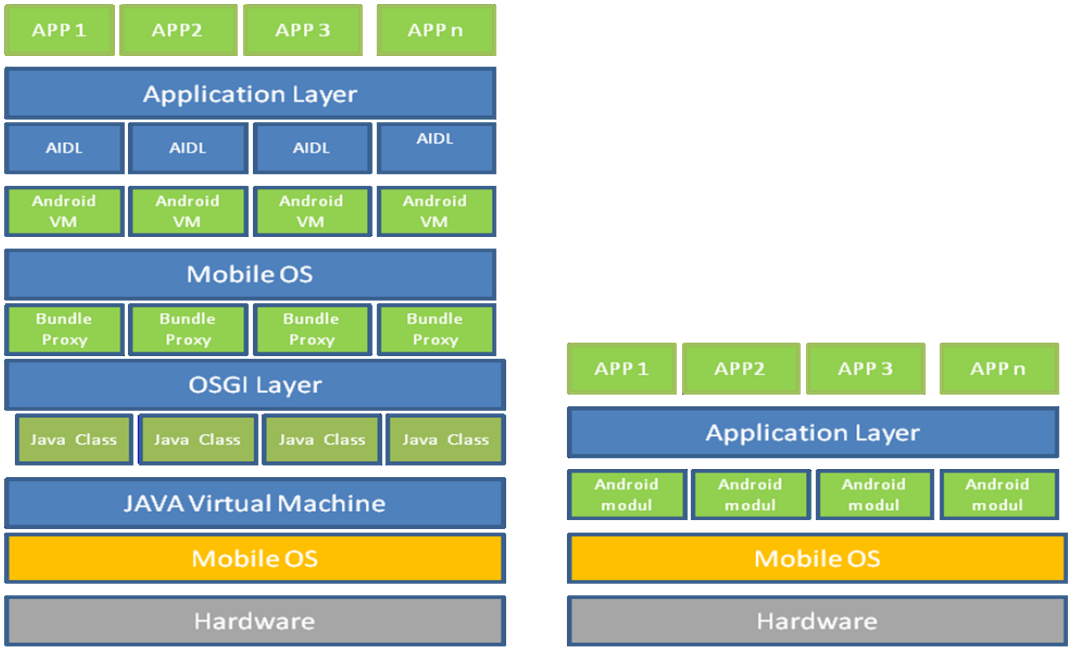
In this part, we propose a new adaptation of SOA in the mobile context by incorporating OSGi-based SOA primitives directly into Android OS layer. Our solution targets to develop a

component-based software package that implements the service-component model with modularity and reusability capabilities. Hence, modularity and dynamicity are assured without introducing multilayer bottlenecking middlewares. In fact, our methodology defines a new design on Android terminals to manage the life cycle of the services that are used by Android applications. Regarding performance issues, the proposed solution implements a model based on components called Andro-modules, that makes Android operation system layer natively service oriented.

5.7.1 Motivation for a new service based architecture

In our OSGi-based solution described in the preceding section, the framework has been integrated as a middleware on the mobile OS to bring SOA features such as modularity, reusability and dynamic updates. However, this solution shows some drawbacks as in the following:

- In the context of a mobile-to-mobile environment, the interaction of mobile devices using OSGi bundles requires that all the mobile nodes install the OSGi framework previously;
- In the context of a local mobile device, the OSGi-based Android is a multi-layer platform. To execute the services of a local bundle, the Android application (apk) triggers a bundle proxy by running Android API, then the proxy asks the OSGi middleware that calls the requested bundle. The bundle seeks for elementary packages of the Java virtual machine, and then the mobile OS layer triggers the physical capacity. These drawbacks introduce at the same time performance impacts and compatibility limitations. Since we want a SOA support for Android, we propose here to adapt the Android platform by implementing analogically to OSGi "bundles" components called Andromodules without adding an OSGi framework (such as Felix) on Android that may alter the performances (see Figure 5.9).



Different layer in Androlix **Reduced layer in Andromodul**

Figure 5.9. OSGi AndroLix versus native Andromodules

5.7.2 Andromodule based Architecture

As shown in Figure 5.10, any Andromodule A is a dynamic Android module composed of:

- a service interface providing a set of APIs;
- a service body that is an Android Dalvik Code;
- a Manifest file that is an XML file that describes the API and the services offered by the Andromodules on which A depends for its execution.

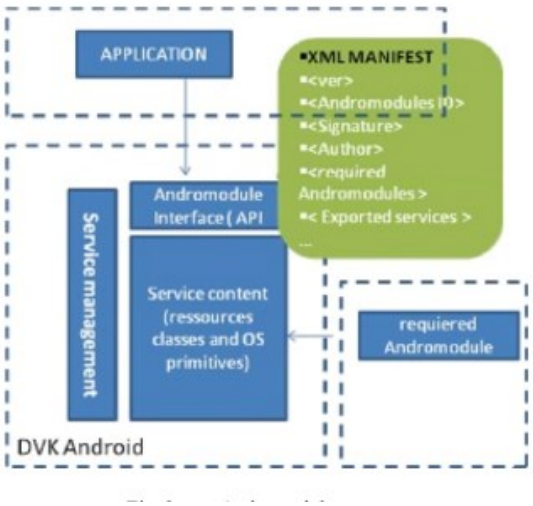


Figure 5.10. Andromodule based architecture

The Manifest file contains executive and non-executive information. The non executive information is related to the version of the Andromodule, to the author and to some other descriptions, while the executive information corresponds to the prerequisite Andromodules, to the services or API publicly available to other Andromodules and to the requirements for minimum performance execution. Note that the idea of the Andromodule is analogous to the OSGi bundle unit except that the Andromodule is executed directly as a Dalvik code, avoiding the dexification process used in the first solution with AndroLix middleware to convert Byte code to DEX code. As shown in Figure 17, the Andromodule based architecture is a lightweight platform in comparison with the OSGi-based architecture because in the former the Android applications interact directly with Andromodules for execution without going through different middleware layers. In addition, the Andromodules are part of the OS modules for which the management is launched upon launching the Android OS. As a consequence, no performance impact on application running. The service management system, implemented as an Android class, includes many methods used to manage the life cycle of the Andromodules, to register and share services across Andromodule service providers, and to decouple services consumers. The principal functions of the service management system are the following:

- ***installModule*** (location): installs a module from a specified location and assigns it an ID automatically.
- ***start*** **(Id):** checks the existence of a module corresponding to the given ID, tries to resolve this module and starts it if it succeeds to resolve it. The module-dependency resolving depends on the information that exists in the XML file. Additionally, it invokes the method *activate*().
- ***activate*****(*****Id*****):** checks the existence of a module corresponding to the given ID and registers the service in the registry.

- ***stop(Id)*:** checks the existence of a module corresponding to the given ID and stops the service if it is in a start or active state.
- ***uninstallModule(Id)*:** checks the existence of a module corresponding to the given ID and uninstalls it if it is resolved or stopped.

5.8 Conclusion

We identified, in R-OSGi platform, an excellent support for dynamic services invocation on Platform as a Service (PaaS) mobile Cloud, thanks to the transport abstraction through OSGi remote services and dependency management using the OSGi framework capabilities. The proposed solution supports OSGi bundles running on both mobile devices and Cloud-side virtual machine OS platforms, and the bundles can be transferred and run on different platforms without compatibility issues. The presented solution is achieved: 1) by incorporating OSGi into Android software development platform, 2) by setting up a remote OSGi on the Cloud and on mobile devices, and 3) by defining three service interaction models. The resource requirements of R-OSGi are by design modest. In addition, we outline through the different service models that R-OSGi does not impose role assignments (e.g., client or server). The relation between modules can be symmetric and so is the distributed application generated by R-OSGi.

In the second part of this chapter, we presented a new modular design platform for application development. This design is directly implemented on the mobile OS to provide optimal performances. In Android platforms, each application is executed on a distinct Dalvik Virtual Machine (DVM). In the opposite, in an OSGi design, all OSGi applications run in one shared Java Virtual Machine. In the first case, Android platform has the drawback of components duplication within DVMs. In the second case, with a single VM in OSGi, the classes are shared and for each bundle is associated a distinct class loader that allows selective export of internal packages and selective import of required dependencies, but can also introduce latency in application execution since it requires the launching of many layers (JVM, OSGi framework (AndroLix or Felix …)). The proposed design is an intermediary solution which offers the same performance execution as a classic Android execution since it is integrated within the Android OS without additional layers. Moreover, it offers modularity facilities for sharing services and reduces duplication in mobile devices thanks to the introduction of the Andromodule components. In fact, with the emergence of mobile applications and the high performance requirements facing a big data and remote Cloud services, we need to consider modularity and life cycle of the services directly inside the design of the mobile operating systems.

Before evaluating, in chapter 7, the performance of the proposed solutions, we extend in the next chapter our designed solution to a distributed and multi-environment context through a Cloud of interacting mobile devices.

Chapter 06: Service Architecture for multi-environment Mobile Cloud Services

6.1 Introduction

The previous chapter describes how to adapt service oriented architecture to build a dynamic framework while alleviating several problems like portability and interoperability in MCC.

In this chapter, we present a global MCC architecture solution adapted to three different designs depending on both the application domain constraints and the execution environment. The first scheme is a central mobile Cloud design that offers to mobile devices offloading capabilities into powerful Cloud-centric servers and remote services consuming. This centric design is the most applied model for Cloud-based mobile applications since it's similar to a client/server approach. However, centric approaches assume a reliable WAN connection to Cloud servers and limit the use of mobile devices to a client role.

An alternative MCC approach considers the accumulative power of swarm of mobile devices [69], despite the limitations of mobile-device resources. This swarm can be turned into a giant resourceful and ubiquitous infrastructure to provide a low-cost distributed computing, or rich sensing distributed applications. This introduces the second proposed model which is a distributed design based on Mobile-to-Mobile (M2M) cooperation. Vehicular networks, P2P music Cloud platforms, smart-cities applications, and all interest-based apps can be suitable use cases. In this approach, nearby computing devices [70] can be weak devices that might not be able to perform complex resource-intensive tasks alone but by collaborating they can provide enriched services. Furthermore, if services are voluntary and free, it gives freedom to mobile providers to terminate their services anytime.

To deal with these challenges, particularly in hostile environment, where WAN Internet connections can be absent or the mobile devices can have poor connection capabilities, or either when service-offloading providers need to be more controlled, Cloudlets model has been introduced in [71].

In a 3-tiers model, Cloudlets act as intermediary nodes between mobile devices and centric Cloud servers. Cloudlets are explored as deployable services on nearby units to be invoked with reduced latency using alternative LAN communication technologies such as one hope cellular networks and Wi-Fi that enhance performances.

In the next sections, we will highlight the three architecture models and show how to deploy OSGi framework in these environments.

6.2. Centric Cloud service oriented design

The definition of mobile Cloud computing is commonly related to a centric architecture (see Figure 6.1). This model aims to:

- Prolong the capabilities of storage computation-limited devices;
- Provide seamless access to data/application on a remote resource rich server from anywhere; and
- Provide Client/Server based services.

Mobile devices are clients that require services to the Cloud in the form of IaaS, PaaS and SaaS services.

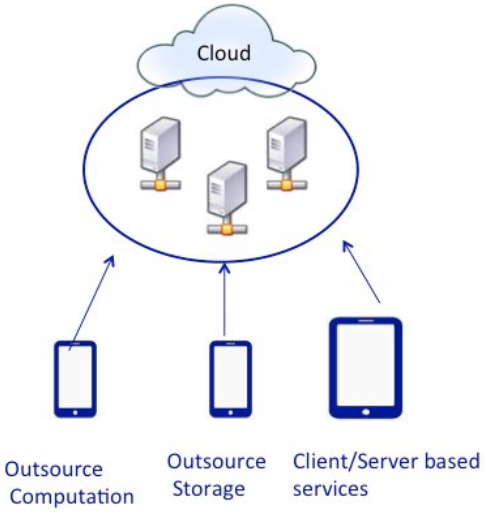


Figure 6.1. Centric Cloud design

6.2.1 OSGi integration for centric model

In this section, we will explain how a centric Cloud solution is designed and deployed using OSGi framework.

6.2.1.1 Framework interfaces

The OSGi framework architecture is described in the previous chapter with:

- A client interface on the mobile device
- A provider interface on the Cloud side.

The interaction between the client and the provider is performed following the steps bellow:

1) In the service provider side, for each remote service, a proxy bundle is generated dynamically with the exported methods;
2) In the mobile side, the client gets the interface description of the service in the form of a bytecode. Then it parses it and creates a bundle proxy byte code that has to be run on the Android platform;
3) As Android apps are composed of Dalvik codes, traditional R-OSGi bundles are patched to transform dynamically, using DEX service, the Java bytecode to a Dalvik bytecode;
4) Then, the application bundle of the Android platform uses the invoked remote bundle.

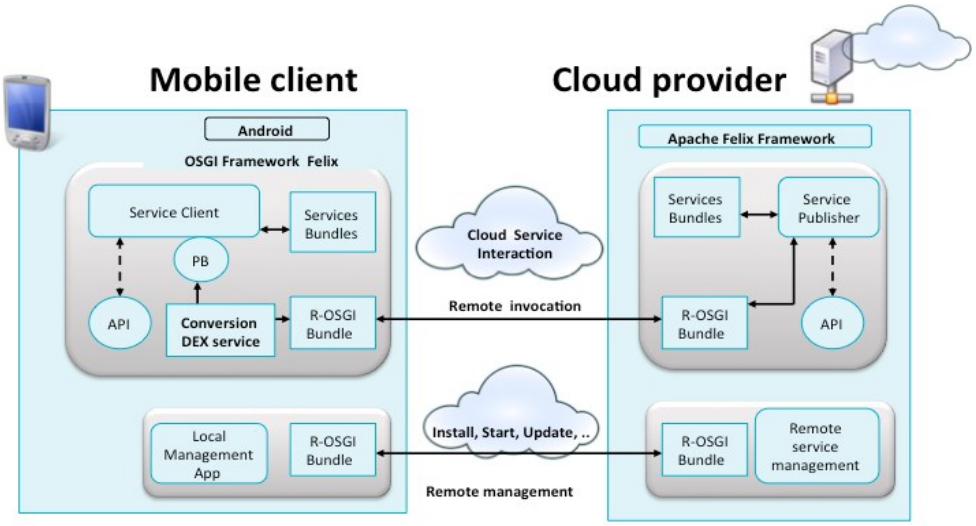


Figure 6.2. OSGi centric Cloud architecture

As explained, in the previous contribution, to ensure the interaction between OSGi frameworks distributed between the mobile device and the Cloud, we propose to use the Remote-OSGi (R-OSGi) bundle to handle OSGi remote services (see Figure 6.2). R-OSGi specification is unique in being completely transport and protocol independent [71]. The provider and consumer roles are defined with respect to the use of the service rather than the transport:

- **Role 1**: The Cloud provider framework is responsible for discovering and creating proxies that represent remote services. That's why the client doesn't need to explicitly mention the address of a remote service in the consuming bundle. All remote services have the following service property: *service.imported*
- **Role 2**: On the consumer side, there is a need to establish a local proxy endpoint and to connect it to the remote provider endpoint in order to import the remote service.

6.2.1.2 Bundle-state adaptation

In the mobile context, the communication may go down due to network disconnection at any arbitrary time. There isn't much to do in the provider side if the client is unreachable. However, we need to take in account this possibility in the client side. In the following, a bundle-state adaptation is proposed to deal with network disconnections especially in hostile and mobile environments. One of the features of OSGi is the bundle dependency resolution. Before starting the application bundle, the framework checks if the needed services (list of the import packages) are available. This minimizes application bugs due to the required-components absence. In dynamic MCC environment, since the required bundle is remote, a bundle can become unavailable after a network disconnection due to the mobility of the device for example. In OSGi framework, a bundle has different states as shown in Table 6.1.

ACTIVE	The bundle is running.
INSTALLED	The bundle is installed but not yet resolved.
RESOLVED	The bundle is resolved and is able to be started. This means that required packages and dependencies are available.
START_ACTIVATION	The bundle start operation must activate the bundle according to the bundle's declared activation policy.

START_TRANSIENT	The bundle start operation is transient and the persistent auto-start setting of the bundle is not modified.
STARTING	The bundle is in the starting process.
STOP_TRANSIENT	The bundle stop is transient and the persistent auto start setting of the bundle is not modified.
STOPPING	The bundle is in the stopping process
UNINSTALLED	The bundle is uninstalled and may not be used.

Table 6.1. Bundle states

To handle service continuity in this OSGi-based MCC framework by investigating the behavior of started and running services in case of disconnection with the remote bundles, we propose to complete the bundle state by adding a health-check process locally to the bundle, as in Figure 6.3. This process is performed as in the following: after the first bundle resolution, the access to the remote bundle is checked using timeout and frequency attributes. If the remote bundle is considered unreachable, the client bundle is set to a "frozen state" until the required bundle is reachable again. If the application timeout is elapsed, a "white flag" is returned to the client bundle that switches to stop state without bugging the application.

As introduced here, the benefit of the health-check mechanism is that if a service becomes unavailable while being in use, the composite service's internal logic can be modified without recompiling the assembly or restarting it. Once a bundle is installed on the mobile framework, the bundle dependencies are resolved by the framework. This step ensures that the local bundles are available, and starts remote dependencies health-check process. This process tests remote Cloud access using adapted attributes such as periodicity, accepted delay, etc. These check attributes can be listed in the Manifest file of the main bundle.

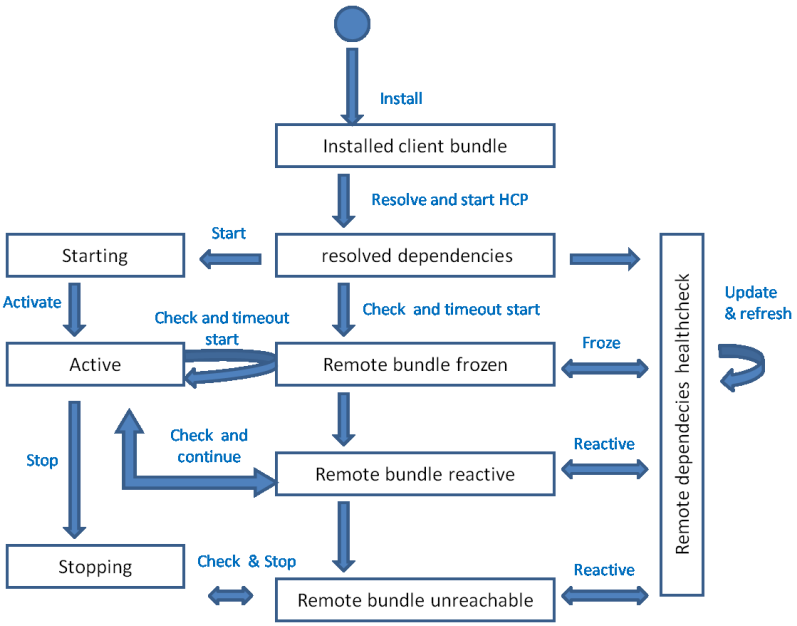


Figure 6.3. The proposed bundle life cycle

6.3. Distributed M2M Cloud services

In this section, we propose a distributed mobile-to-mobile service architecture and we show the way to overcome some issues inherent to the M2M design.

6.3.1 Application context

In this model (see Figure 6.4), mobile devices are considered not only as service consumers but also as service providers to build a sensing-based application platform. In such

a framework, each mobile device senses its surrounding information, such as wireless communication channel status, neighboring nodes information, environmental information (e.g., CO2 and pollution levels, etc.), personal information (e.g., medical and health information using bio sensors), etc.

Figure 6.4. MCC global design

With the power of mobile-to-mobile communication and distributed interaction, new MCC emerging applications depend on a distributed model where mobile nodes can be both service consumer and service provider. Many examples inspired from smart cities [72], like connected vehicular safe driving apps, urban sensing, mobile social computing, mobile healthcare, location based and community services, domotics, etc. may require interaction between devices augmented thanks to personal Clouds as well, where offloading from smart phones to more powerful tablets can also be considered. In this context where the Cloud is qualified as virtual since it is built with autonomous limited-resource devices, the service providers are mobile devices that need to cooperate in order to provide the required service.

In the following, we illustrate the use of OSGi to build a suitable Mobile-to-Mobile service-oriented architecture.

6.3.2 OSGi-based proposed solution

In the OSGi-based approach (see Figure 6.5), each mobile device is acting either as a provider, a consumer, or both. Each device needs to have an embedded OSGi mobile framework. Mobile frameworks are downloadable from the centric Cloud. Once the framework is installed, applications supporting OSGi can interact with a proxy bundle like presented in the previous centric model.

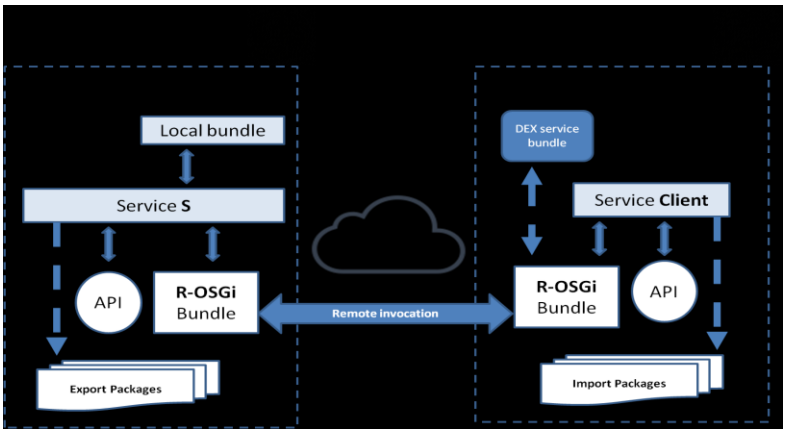


Figure 6.5. Mobile-to-Mobile OSGi interaction

We choose the Apache Felix platform to implement OSGi because of its small size compared to other OSGi implementations. To integrate Felix within an Android platform, we implement Felix as an Android remote service that extends the *android.app.Service* class. The access to Felix being made via an RPC[1] communication protocol, the framework provides to clients an AIDL[2] description containing different methods that are implemented by this service. The R-OSGi bundle patched with the dexification process is only necessary on the client interface of each mobile framework because of the incompatibility between Java VM used by OSGi and Dalvik VM used by Android.

6.3.3 Service advertisement and discovery components

In OSGi, the bundle that acts as a service provider registers the service interface, the service properties, and the service implementation into the OSGi service registry (see Figure 6.6). The bundle requestor retrieves the service interface and the service properties from the OSGi service registry.

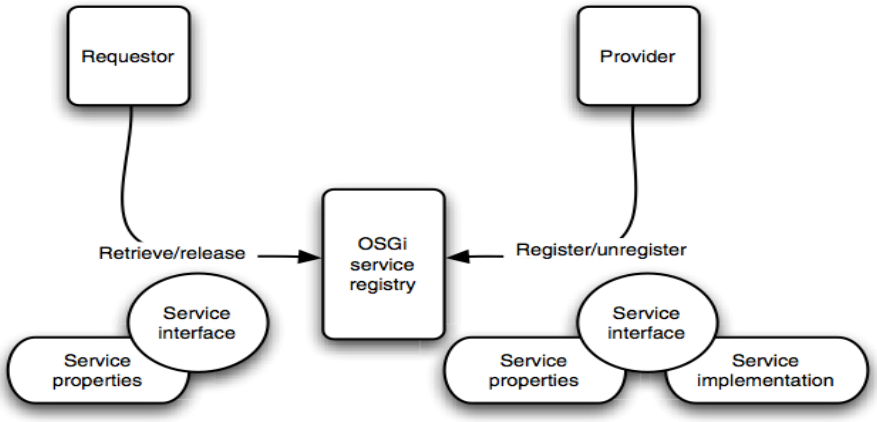


Figure 6.6. Service Registry in OSGi

[1] Remote Procedure Call
[2] Android Interface Definition Language

The OSGi service registry enables a bundle to publish objects to a shared registry, advertised via a given set of Java interfaces. It acts as the mediator, isolating the provider from the client. The registry facilitates bundles decoupling by promoting a dynamic collaborative model based on a service-oriented paradigm (publish/find/bind). However, in a full-distributed model with ad hoc mobile interaction, a dynamic discovery of the best surrounding provider is necessary to avoid centric bottleneck. To address the inter-OSGi framework communication issues, we adopt an XMPP solution [73] that is described in the following paragraphs.

6.3.3.1 XMPP presentation

XMPP (Extensible Messaging and Presence Protocol) is an open standard communication protocol for message-oriented middleware based on XML (Extensible Mark-up Language). XMPP server is used as a signaling and communication service between different OSGi frameworks.

6.3.3.2 XMPP advantages

XMPP offers various benefits as in the following:

- **Portability**: unlike most instant messaging protocols, XMPP is defined in an open standard and uses an open system approach, by which anyone may implement an XMPP service and interoperate with other organizations' implementations;
- **No license needed:** XMPP is an open protocol; implementations can be developed using any software license;
- **Decentralization:** The XMPP can be run in each mobile platform. The XMPP network uses client–server architecture. The model is decentralized - anyone can run a server. By design, there is no central authoritative server;
- **Overlay communication**: with XMPP, routing messages can be based on a logical endpoint identifier - the JID, instead of IP addresses. This presents opportunities to use XMPP as an overlay network implementation on top of different underlay networks. This is a particular opportunity in MCC where virtual machines, networks, and firewalls can present obstacles.

6.3.3.3 XMPP integration to OSGi framework

To integrate this service within Android based framework, a signaling and communication agent bundle has been developed within Felix a lightweight implementation of OSGi. To enable the communication between OSGi bundles and the Android service, we developed internal supporting bundles with the mechanism of Java reflection and Android broadcast inside OSGi framework, as well as an internal process inside the Android service. With internal supporting bundles, we introduce the following interaction between three types of bundles: XMPP bundle, advertisement bundle and discovery bundle as shown in Figure 6.7.

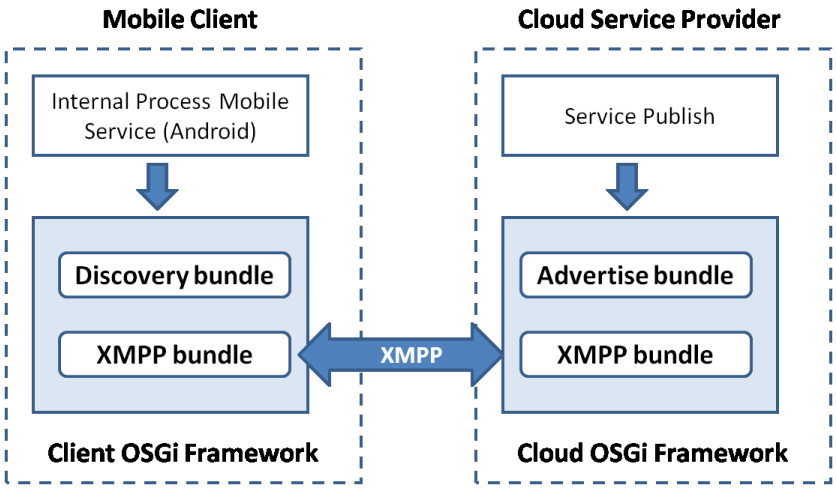


Figure. 6.7 Bundle discovery interaction

Discovery Bundle: The discovery bundle registers a broadcast listener that is used to listen to the Android broadcast sent to OSGi framework from the internal process inside Android service. Upon receiving a broadcast, the discovery bundle will dispatch the message to another service running inside OSGi framework. In our case, it will be sent to XMPP client bundle.

Advertisement Bundle: In the Cloud side, for each published bundle, a notification is sent to the advertisement bundle that adds public Manifest information to the advertisement bundle. The advertisement bundle broadcasts the availability of a new bundle or a group of related bundles. This interaction can be used to update the bundles in asynchronous way. The interaction between the *Discovery* bundle and the *Advertisement* bundle can be established based on a publish/subscribe paradigm. In the centralized model, the discovery bundle updates the local OSGi service registry with the remote bundles' information (bundle ID, location, URL, IP/Port access…) when an update is received from the Cloud publisher advertisement bundle.

Using functional items and XMPP protocol, the bundle discovery can be optimized by applying non-functional items, as proposed in [85] where service discovery optimization is performed by using an indexing of two indices, one for service outputs and the other for service inputs. The functional discovery is split into two tasks, namely index creation and service retrieval. In our model, after the bundle discovery, the remote bundle selection is performed not only based on the outputs, i.e., exported services of the bundle but also based on the inputs, i.e., imported services of the provider bundle. For example, when a client bundle discovers two provider bundles with the same required service (that is, the same exporting packages), the provider service with local imported services is preferred to a provider with remote ones. After assigning local constraints to individual services, each service client is formulated to select a service of maximum utility as the best service based on the respective service interface and the provider service local constraints. This permits to reduce the number of hops in a remote complex service provided by many providers at the same time.

6.4. Cloudlet based model

The third model that we investigated is a three-tier architecture based on the Cloudlet concept as discussed in the next paragraphs.

6.4.1 Application context

In the preceding proposed architectures based either on centric approach or distributed one, we assume implicitly that the Cloud WAN or Internet network access is reliable. However, coverage quality can be variable depending not only on the distance with the base station, but also on the variation of the available bandwidth and the speed. In a hostile environment where first responders operate in crisis situations or soldiers fight in battlefields for example, connection to the Cloud is generally impossible. To reduce interaction latency and dependency on the WAN connection, Cloudlets [74] are proposed to create a proximate Cloud to access nearby remote resources. A Cloudlet is a new architectural element that arises from the convergence of mobile computing and Cloud computing. It represents the middle tier of a 3-tier hierarchy: mobile device, Cloudlet, and Cloud as illustrated in Figure 6.8. As stated in [74], a Cloudlet can be viewed as a "data center in a box" to make the Cloud closer to the mobile device while relying on high bandwidth (e.g., one-hop Wi-Fi) and low latencies.

In the following sub-sections, we discuss three solutions that implement a Cloudlet based design: the first one is based on Virtual Machines (VMs) and is related to an existing project, while the others are our own implementations: one uses Docker containers and the other relies on OSGi framework. To compare the performances between these solutions, some measurements have been undertaken and discussed in chapter 7.

6.4.2 VM based Elijah solution

Despite their introduction in the 1970's [75], VMs have been propelled thanks to the Cloud data centers. Hypervisors like VMware, Xen or KVM brought VMs to a wide use in Cloud service platform thanks to the elasticity and isolation properties. Simanta et al., in [76], present a reference architecture based on Cloudlets as part of a project called Elijah project (see Figure 26). Based on a VM manager, the Cloudlets are viewed as code-offloading elements used to serve mobile devices in single-hop proximity.

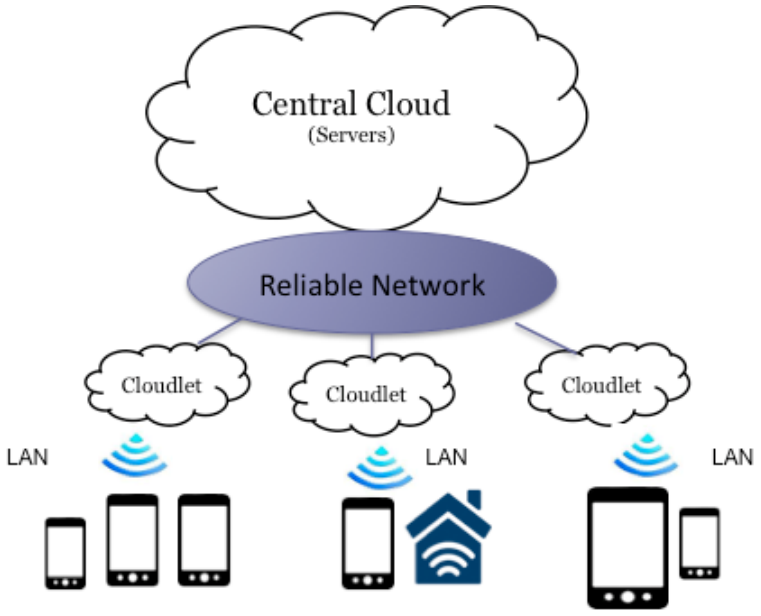


Figure 6.8. Cloudlet general design

The major components of this architecture are the Cloudlet host and the mobile client. The Cloudlet host is a physical server that hosts a discovery service that broadcasts the Cloudlet IP address and its port number allowing mobile devices to find it. The Cloudlet host contains a base-VM image that is used for VM synthesis, and a Cloudlet server that handles offloaded code in the form of application overlays. Whereas, the mobile client hosts apps that discover Cloudlets and upload application overlays to the Cloudlet.

An application overlay is created once per application (see Figure 6.9). The base VM is a VM disk-image file that is obtained from the central core and saved to a Cloudlet that runs a VM manager compatible with the base VM. The VM manager starts the base VM and the application is installed. After installation, the VM is shut down. A copy of the modified base-VM disk image is saved as the complete VM disk image. The application overlay is calculated as the binary diff (VCDIFF RFC3284) using xdelta3 [84] tool, between the complete VM disk image and the base VM disk image. The base VM is then deployed to any platform that will serve as a Cloudlet. The mobile device carrying application overlays will be able to execute these applications on any Cloudlet that has the corresponding base VM.

Figure 6.9. VM Overlay in Cloudlet architecture

The Cloudlet client is an Android app that:

1. Discovers Cloudlets through information broadcasted by the discovery service residing in the Cloudlet host;
2. Creates an HTTP connection to the Cloudlet server for overlay transmission and uploads the overlay.

In the Elijah solution, the client needs to carry the overlay (see Figure 6.10), which can be heavy because of the limited resources of mobile devices. In addition, the VM synthesis operation is time consuming. The overlay needs to be processed offline while being strongly tied to the Cloudlet VM. Overlay replication between Cloudlets needs also to be studied to offer Cloudlet roaming with full security.

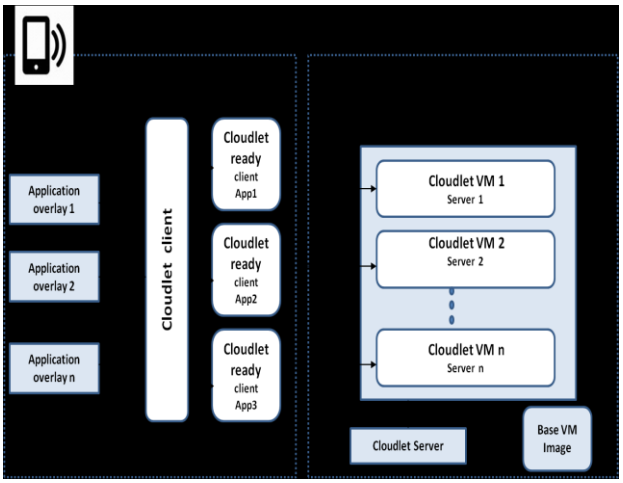


Figure 6.10. Elijah based overlay architecture

6.4.3 Container-based Cloudlet solution

In the virtualization-based approach (see Figure 28), each VM contains a full operating system with all its layers. The hypervisor scheduling operates at two levels as explained in the following. The first scheduling occurs at the hypervisor level where the hypervisor schedules VMs, and the second one at the virtual machine level where the guest operating system schedules processes. Taking that into account, it is obvious that this approach creates a significant overhead. Another virtualization approach, based on containers [77, 78], has been proposed to reduce significantly this overhead, and thus providing better performances, especially in terms of elasticity and density within a lean data center.

In the container-based virtualization, scheduling occurs only at one level. The containers contain processes, while the kernel and the libraries are shared between the containers (see Figure 6.11). Hence, the scheduler is used only at process level to schedule processes. The behavior is similar to a classical operating system; the only difference is that, in container-based virtualization, processes are affected to a container, and between each container, isolation occurs at two levels:

- Isolation between processes; and
- Isolation between processes and the hardware.

Docker container based virtualization improves performances since there is no hypervisor overhead (see Figure 6.11).

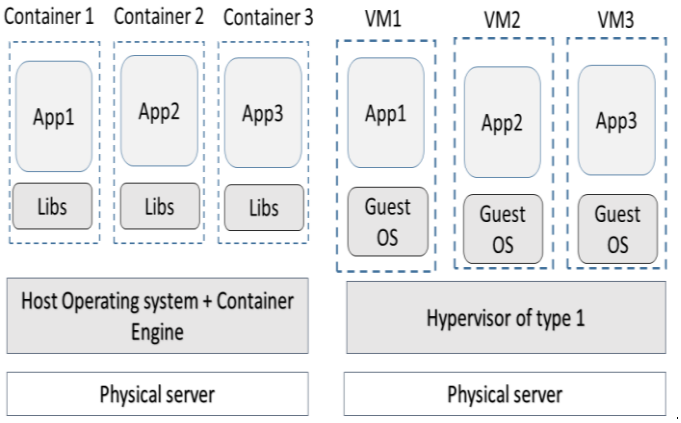


Figure 6.11. Containers vs VM layers

Using Docker containers, both of the mobile device and the Cloudlet server may use Linux-based OS. In this case, the app is isolated within a container and can be executed either within the mobile device or the Cloudlet container depending on the resources needed by the application. In this solution, there is no need to overlays like defined in Elijah project. Moreover, because the Docker containers are standard and include minimum libraries, there is no need to pre-requisite installations, besides the fact that they are less heavy (MB) than VM images (GB) and consequently can be offloaded while consuming less energy.

6.4.4 OSGi-based Cloudlet model

In the OSGi based solution, we propose to install each OSGi framework on a dedicated VM to meet scalability and elasticity needs, where distinct VMs are managed within the Cloudlet by a hypervisor as depicted in Figure 6.12.

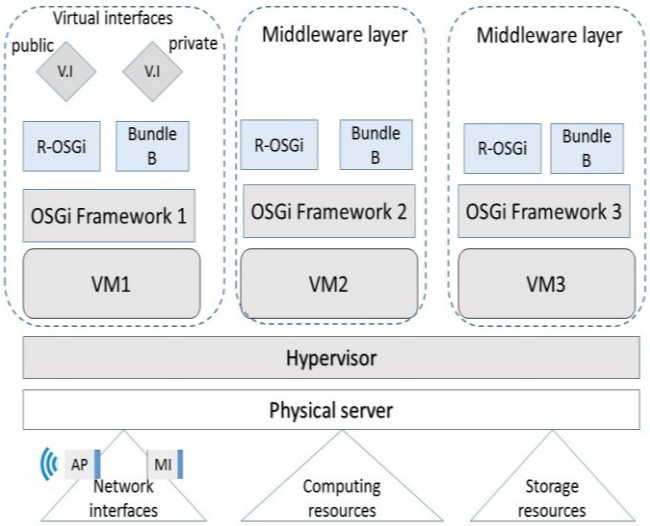


Figure 6.12 OSGi-based Cloudlet architecture

Regarding the global architecture, the Cloudlet has different physical interfaces:

- Access Point interface (AP): provides a Wi-Fi 802.11 dual band (2.4 & 5 Ghz) to connect with the mobile devices.
- Cloud service and management interface (MI) which connects the Cloudlet to the Cloud through a wired network for service communication and for supervision and administration of the Cloudlet station.

In addition to network capabilities, the Cloudlet has storage and computing capacities where Cloudlet storage can act as intermediary caching service. Because we consider a three-tier architecture with the following entities: mobile device, Cloudlet and Cloud, an OSGi framework is installed on each of the three components. Apart the mobile device, each of the Cloud and the Cloudlet hosts the OSGi framework within a dedicated VM. The VMs are created on demand and are deleted when they are not needed. The elasticity, as one of the intrinsic properties of the Cloud and the Cloudlet, is then guaranteed. In the following, we explain how OSGi is distributed through this three-tier architecture.

- **Cloud server framework**

The Cloudlet based architecture relies on the Cloud as a sandbox that can serve either by the Cloudlet to offload the OSGi VM associated with the mobile device when it is leaving this

proximate Cloudlet, or by a Cloudlet that is discovered by the mobile device and that needs to download, from the Cloud, the OSGi VM image of the mobile user to interact with it.

- **Cloudlet framework**

The Cloudlet can be shared between multiple mobile users or multiple applications that need the segregation of execution environments. The Cloudlet system is segregated into different virtual zones (VMs).The Cloudlet can download the required image from the Cloud as explained earlier, but it can also build locally the corresponding VM of the mobile device by getting initially the OSGi framework from the Cloud in order to set up the OSGi environment within the VM. This entire environment (VM enriched with OSGi) will serve to provide OSGi services to the mobile device. Using the management interface, the Cloud ensures the interaction and the management of the Cloudlet.

- **Mobile framework**

An OSGi framework is installed in the mobile device and includes a Cloudlet interface for the discovery, and service interaction with OSGi framework hosted within the corresponding VM in the Cloudlet. In this OSGi-based Cloudlet model, the mobile device makes call to the Cloudlet once the Cloudlet is discovered. After a mutual authentication is performed, the Cloudlet downloads from the Cloud a sub environment of the user, which allows the mobile OSGi framework to interact with its image on the Cloudlet.

6. 5. Conclusion

In this chapter, we propose a design classification of mobile Cloud computing in three main architecture models. First, we present the traditional model used for centric Cloud services and show how OSGi based SOA framework can fit to this model. Second, we adapt the first model to a distributed Mobile-to-Mobile Cloud computing in order to deal with dynamic mobile and connected objects interaction. In the third part, we adapt our model to 3-tier Cloudlet design, and we explain how this model can fulfil IoT network constraints and can be suitable for hostile environments.

We proposed, in this chapter, a design for different MCC-SOA architectures to show that there is no ideal model, but each model can be more suitable depending on the application context. Deploying one unique framework to deal with different contexts is a major benefit facilitating the remote Cloud services development and deployment and a very important added value in MCC services where the same user and the same device act in different application contexts at the same time. Our service-oriented framework based on OSGi was adapted to each of the three-presented contexts. Thanks to the bundle units, remote OSGi bundles and a dynamic management service, our middleware deals with multi basic Cloud needs with local storage and execution, a remote execution or a service migration. We also show how mobile Cloud computing can offer a very rich environment to provide multiple services by a dynamic and fluid collaboration between the mobile devices as a part of the Cloud. To meet the scalability requirement, we propose in this model to use advertisement and discovery components based on a publish/subscribe paradigm, and implemented it with eXtensible Messaging and Presence Protocol XMPP. Additionally, we launch the OSGi framework within VMs in the Cloud as well as in the Cloudlet. The Cloudlet based design seems to be a very interesting and promising model.

In the next chapter, we will evaluate the performances of the different solutions proposed and explored.

Chapter 07: Implementation and Performance Analysis

7.1 Introduction

The previous chapter adapts the proposed MCC framework to different architecture contexts. The first one is a traditional centric model, where mobile devices are reduced to consuming services. The second one is a distributed model where the power of mobile-to-mobile interaction offers unlimited value-services opportunities, and finally a three-tier architecture is considered with the introduction of the Cloudlet notion.

In this chapter, we conduct some experiments and analyze the performances of the three OSGi-based architecture contexts that we investigated before.

For the centric Cloud model, our OSGi implementation in MCC context has been tested either by using the mobile Cloud (called MobiCloud[3]) of the Arizona State University (USA) and a private Cloud set up on a local area network (LAN). MobiCloud is a mobile Cloud computing developed by SNAC[4] group led by Dr Dijiang Huang with whom we collaborated for this part.

Regarding M2M design, we compared the performances of our two proposed solutions: the middleware solution based on OSGi implementation and the Andromodule Android embedded implementation.

Additionally, we also explore the performance of our service-oriented framework based on the Cloudlets, and contrast it with two alternative existing solutions such as VM based solution and Docker container based approach.

7.2. Mobile client/Cloud provider performance analysis

In order to test the proposed OSGi-based service model and to evaluate the impact of the implemented middleware in terms of execution time and memory consumption, we developed a demonstrative service example established in MobiCloud, a mobile Cloud computing designed by Arizona State University [79]. MobiCloud provides VMs in a PaaS (platform as a service) model.

7.2.1 MobiCloud network connection design

In the MobiCloud, Xen servers are installed and managed with XenCenter [80]. A private Vlan is defined for the VMs and a public gateway is defined for Internet traffic. A firewall protects the access to the Cloud by allowing only our CNAM IP address as traffic source. A GRE tunnel (Generic Routing Encapsulation) [81] is established on public interfaces of each part to segregate remote traffic in Internet and to create layer 3 network extension over Internet as shown in Figure 7.1.

[3] https://www.thothlab.org/
[4] Secure Networking And Computing

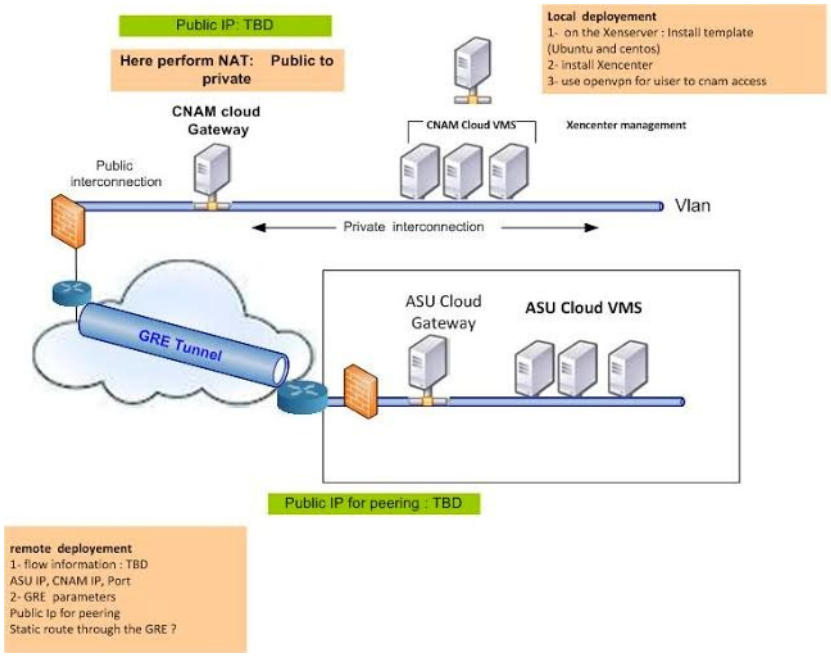


Figure 7.1 MobiCloud network connections

7.2.2 Mobile Cloud OSGi integration

In the second part, services are exposed directly on public interfaces to allow access to mobile devices from Internet access. The Cloud publisher server has a public IP address and a TCP port access. Each mobile phone has an integrated OSGi framework. Each mobile framework interacts with its dedicated VM located in MobiCloud VM pool. We set up Felix on a MobiCloud VM and installed R-OSGi modules within this VM. R-OSGi remains accessible via the public Internet URL http://MobiCloud.asu.edu with the port number 9237 as in Figure 7.2.

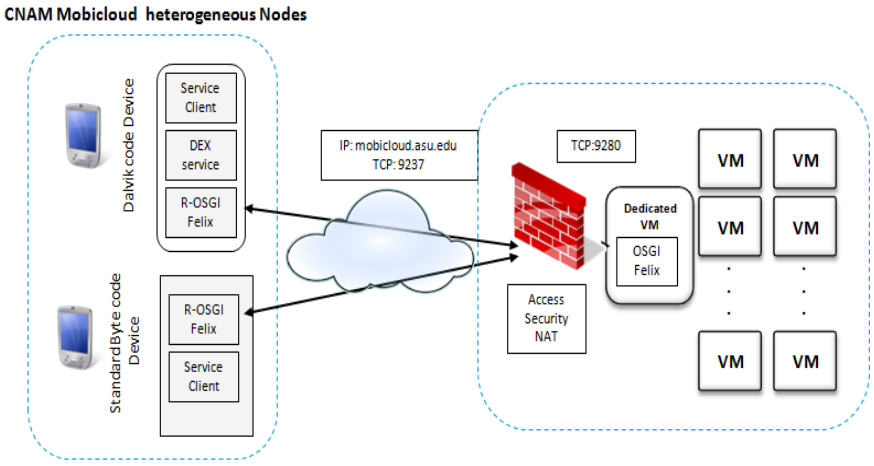


Figure 7.2. MobiCloud OSGi configuration

7.2.3 Performance evaluation

In order to show the performance impact of using OSGi bundles in remote centric Cloud services, we considered three scenarios that we tested as in the following:

a) The first scenario deals with local execution of the bundles within the mobile device without any remote operation.
b) The second scenario considers remote execution where the remote service is provided and executed on the Cloud server framework.
c) In the third scenario, the remote services provided by the Cloud are migrated to be run locally on the mobile device.

7.2.3.1 Execution time

To perform tests on the R-OSGi based Cloud, two cases are considered depending on whether the client and the service provider are hosted on the same framework or not. The goal is to compare a local service execution with remote Cloud base execution using OSGi.

Case 1: Mono framework tests

The client and the provider of the remote bundle are hosted on the same hardware, and share the same OSGi framework. Our objective in this part is to show the performance limits of the mobile devices because of their resource constraints. The measurements are undertaken on a Samsung Galaxy Tab mobile phone that uses a 2.2 version of Android, with Cortex 1.0 GHz processor.

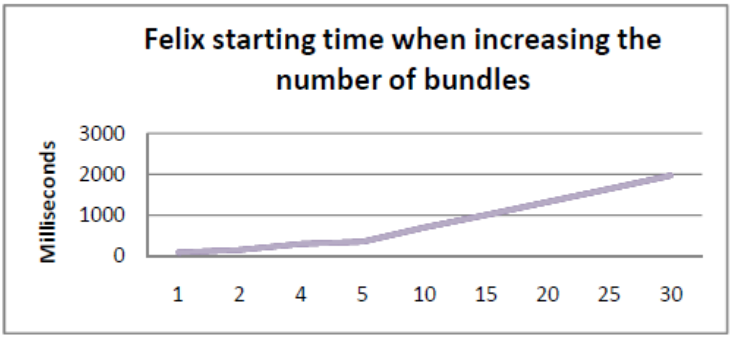


Figure 7.3. Felix starting time when increasing the number of bundles

First, we start Felix on the mobile device and we measure the starting time when increasing the number of bundles as depicted in Figure 7.3. We notice that this time may reach two seconds if Felix handles up to 30 bundles. Even if this time seems to be important, Felix is started once when starting the Android platform.

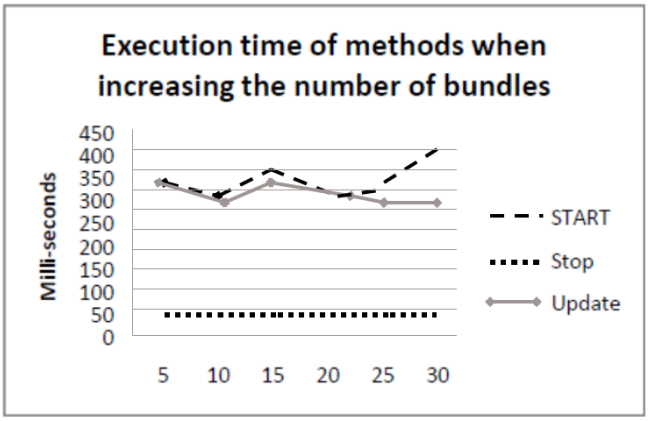


Figure 7.4. Execution times with bundles variation

We also measure the execution time of three methods: *start*(), *stop*() and *update*() to respectively start, stop and update bundles as in Figure 7.4. When increasing the number of bundles, the execution time of the three methods seems to vary independently of the number of bundles. However, when increasing the level of dependencies, the execution time of the methods *start*() and *stop*() increases linearly with the number of dependency levels (see Figure 7.5). A dependency level corresponds to a bundle that depends on another. In fact, when a bundle is started, its dependencies are resolved first. In the same manner, when a bundle is stopped, its dependencies are freed.

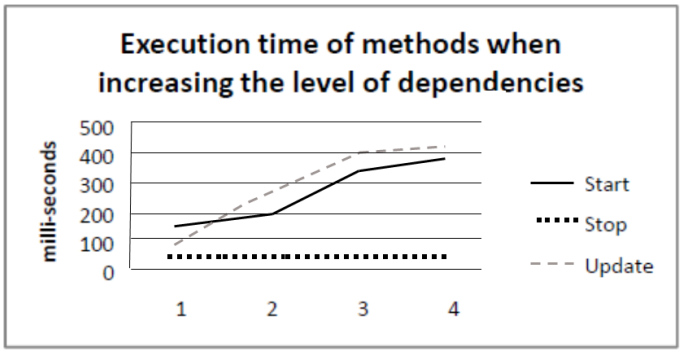


Figure 7.5 Execution times with dependencies variation

Based on the above presented performance evaluations, we show that the methods that manage the bundles life cycle depend only on the number of the bundle dependencies for each bundle.

Case 2: Multi framework case without migration

The client runs on the Tab mobile phone and the provider of the remote bundle is hosted on a Cloud, so that the communication is done through a network access (Wi-Fi and GRE over Internet).

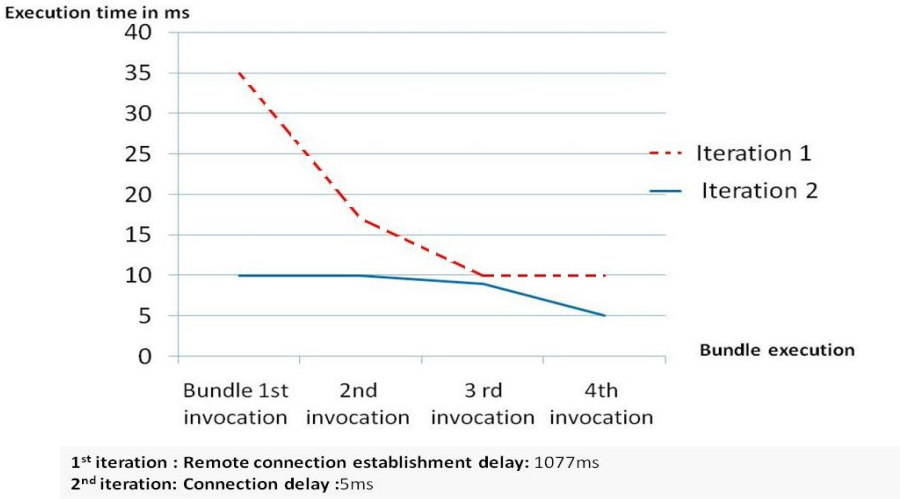


Figure 7.6 Remote bundle invocations

We observed, in this part, the connection establishment and the remote bundle invocation delays. Figure 7.6 shows a significant delay of connection, which is 1077ms. This delay is introduced because in addition to establishing the network connection and dependency

resolution, the end point R-OSGi proxy is created between nodes. By default, Felix stores all of its persistent state in the Felix-cache directory. This is shown, in the second iteration of Figure 35, where the invocation time is significantly reduced, given the parameters already cached.

Additionally, Table 7.1 compares the execution time of the same types of bundles while these bundles are called locally or remotely from the mobile device. We can conclude from this table that launching a bundle on a Cloud does not take a significant execution time.

Action	Case1: Local invocation (Without R-OSGi)	Case2: remote invocation (with R-OSGi)
Start bundle	150 ms	185 ms
Stop bundle	45 ms	57 ms
Update bundle	90 ms	95 ms

*Average bundle size 13940(bytes)

Table 7.1 Remote Bundle Invocation

Case 3: Multi framework case with migration

In addition to the local and the remote execution modes, we considered also a new scenario where a service may have to migrate from the mobile to the Cloud in order to save mobile-device energy and/or to delegate the service processing to the Cloud in case the service is resource consuming.

Each mobile phone with an integrated OSGi framework interacts with its associated VM belonging to the Cloud VM pool. Within the VM of the Cloud, we set up Felix OSGi and installed R-OSGi modules.

Two framework instances are installed:

- One OSGi framework set up within a VM of the Cloud server based on Xen server.
- The other OSGi framework is installed on a "Samsung Galaxy Tab" mobile Android device that uses a 4.1 version of Android, with a dual-core 1.4 GHz processor. Open Wi-Fi 2,4 GHz connection has been used to perform the tests.

Figure 7.7 shows the measured execution time of the bundles when varying their size according to the three execution scenarios: local, remote execution, service migration.

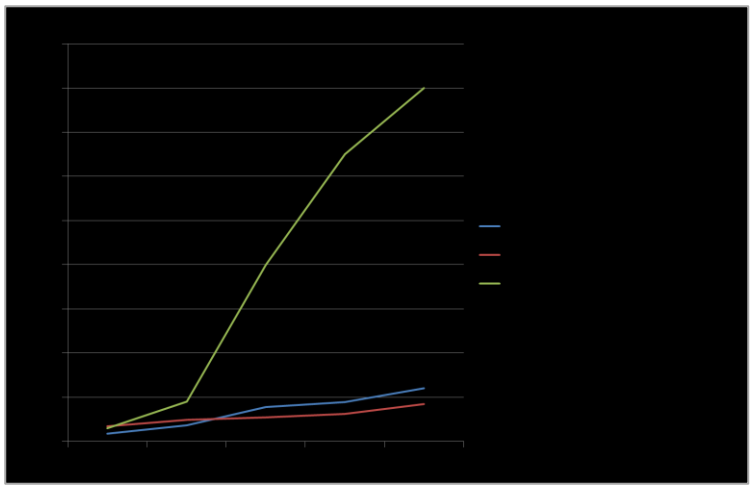


Figure 7.7. Execution-time (in ms) evolution with bundle size

According to the results of Figure 7.7, we can notice that in local-execution context, the execution time grows with the quantity of executed code but the variation of time execution remains relatively low.

In the remote-execution scenario, the bundle execution time is higher than local execution for small bundles because of the added network delays. However, for more important bundle sizes we notice that the time execution in remote access becomes lower than local execution, since the performances of the Cloud server allow a rapid execution. The inversion of the curve indicates when it becomes interesting to execute a remote service where the combination of the transmission delay and the remote execution delay begins lower than the local execution time, as specified in formula 1.

$$Delay_{remote} + Delay_{transmission} < Delay_{Local}$$
$$Delay_{transmission} = \sum (Delay_{Network}, Delay_{I/O}) \quad (1)$$

Whereas, in the migration mode, the execution time and the network delay are very important because the time of bundle offloading grows linearly with the bundle size. In fact, the migration mode requires bandwidth and calculation resources availability in mobile side to execute locally the migrated service. This mode is advantageous when the migrated service is re-used, which allows to decrease the execution delay after the migration step.

7.2.3.2 Memory consumption

The consumed memory obviously depends on the number of local bundles launched and their size. Since the execution of remote bundles is done on Cloud provider nodes, the growth of the remote bundles does not affect linearly the Android memory consumption. Input-Output performance and network communication resources are most sensitive to the growth of the number of remote bundles. Figure 7.8 shows the resource consumption in terms of memory consumed on the Galaxy tablet, with local bundles and remote ones accessed through WiFi.

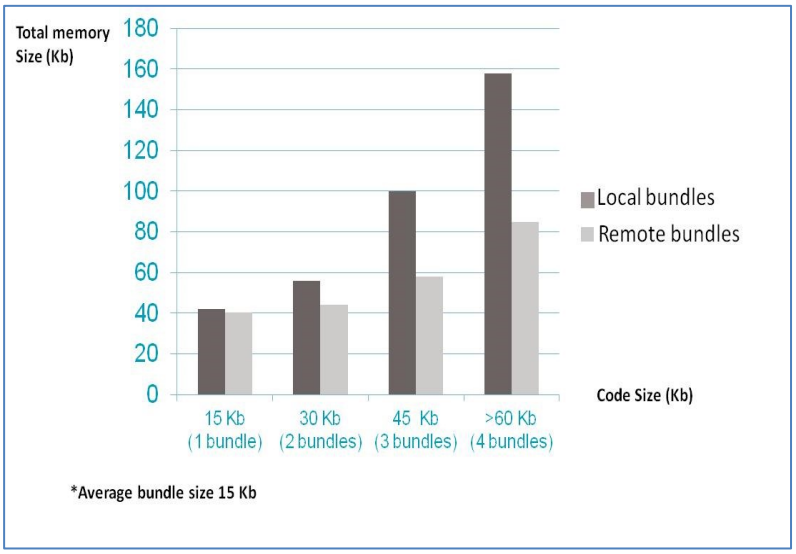


Figure 7.8. Memory consuming

In terms of memory consumption, Figure 7.9 shows how the mobile evolves when the bundle size varies. Obviously, we notice that a local execution is more memory consuming than in remote and migration modes. In fact, the consumed memory depends on the size of the service, i.e., the number of service bundles and their size. When the execution of bundles is remote (i.e., executed on the Cloud provider), the growth of the remote bundles does not affect linearly the

mobile memory consumption. The transmission delay, due to I/O and network communications, is more impacted by the growth of the number of remote bundles.

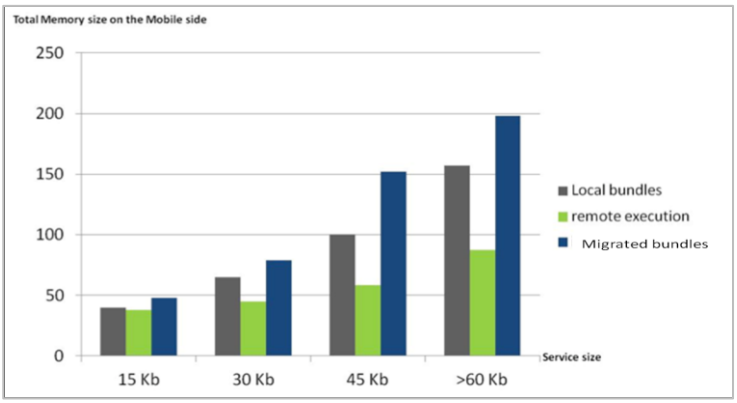


Figure 7.9 Memory consumption evolution with bundle size

We notice that the bundle migration (import and local execution) is the most consuming memory mode, since this execution mode implies an I/O network communication before launching a local execution.

7.3 M2M - OSGi versus Andromodules performance analysis

In this section, we focus on the performances of the designed SOA solutions by measuring the overhead generated by the OSGi Felix middleware and by comparing its performances with the Andromodule-based solution to highlight the benefit of the lightweight solution while keeping SOA modularity concept.

For this purpose, we choose the execution time as a measurement criterion and we use DDMS (Dalvik Debug Monitor Server) tool of Android SDK that runs in both emulators and real terminals. DDMS provides a powerful option called *TraceView* to measure execution time. Hence, in these experiments based on the *TraceView* tool, we measured the execution times of all the developed methods that interact with Felix and then on Android environment. First, we computed the necessary time to start Felix, and then we measured, according to the number of bundles and their dependencies, the execution time of these methods. The experiments undertaken in these two solutions have been tested on an emulator with the following characteristics:

- Device Nexus S (4.0", 480 * 800: hdpi));
- Android 2.2 – API level 8; Ram 343; VM Heap: 32;
- Internal Storage: 60MB
- Bundle size: 32 KB.

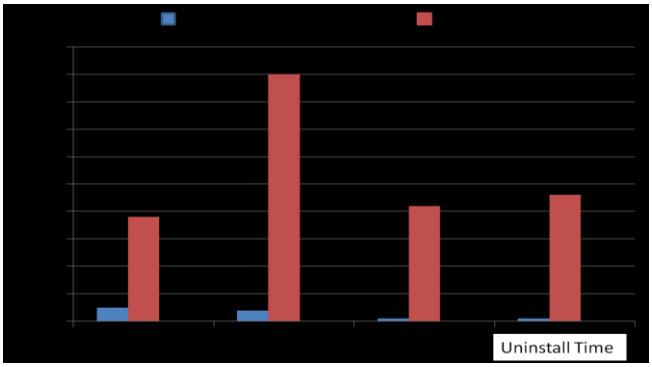


Figure 7.10 Bundle execution time (in ms)

Figure 7.10 depicts that Andromodule-based platform is so promising to solve the problems caused by the huge number of bundles' dependencies, because the operations are executed in a few milli-seconds while the same operations take few seconds with OSGi-based platform. In the OSGi model, the most important time is the bundle *start*() time, since the bundle needs to upload all required resources as Jar files and Java virtual machine during the first execution. The service bundle stays in a lazy state until one of its class files is loaded. In this context, a bundle listener (*Discovery* bundle) can receive notifications only after it has been started, which can introduce potential events missing during the significant start time. However when using Andromodules, the method *start*() is executed at the OS level and all resources are already uploaded during the OS start-up.

7.4. Cloudlet based model

In the preceding proposed architectures based either on centric approach or distributed model, we assume implicitly that the Cloud WAN or Internet network access is reliable. However, coverage quality can be variable depending not only on the distance with the base station, but also on the variation of the available bandwidth and the speed. In a hostile environment, connection to the Cloud is generally impossible. To reduce interaction latency and dependency on the WAN connection, Cloudlets [81] are proposed to create a proximate Cloud to access nearby remote resources.

We analyze, in the following sub-sections, the performances of the three solutions presented in chapter 6 to implement a Cloudlet based design: the first one is based on Virtual Machines (VMs), the second one uses Docker containers and the third one relies on our proposed OSGi framework.

7.4.1 VM based Elijah solution

For this experiment, we have used two personal computers:

- One computer with a Linux Ubuntu 14.04 LTS <64 bits> (CORE i7) as the Cloudlet server ; and
- A second computer with Windows 7 <32 bits> (DUAL CORE) as a mobile device.

After importing the base VM to our local database, we run the Cloudlet server in order to listen to incoming client requests. We assume that in the client side, we have a VM overlay containing the *geany* program and we wanted to reconstruct a custom VM on the server side. The client connects to the Cloudlet upon detecting its IP address, using the program "*synthesis_client*" and supplies the VM overlay. When the client sends the VM overlay to the Cloudlet, the Cloudlet server starts performing the VM synthesis operation. This experiment shows that the offloading technique based on VM overlays is a little bit heavy. As an example, we consider *geany* program whose size is 2.7 MB. When the overlay is generated, it reaches 9.1 MB. Hence, the VM synthesis operation takes approximately 2.4 minutes to be performed. We noticed that the largest amount of time is consumed by the upload overlay operation and depends on the overlay size. Using VM image compression allows better performances.

7.4.2 Dockers based solution

In our environment, we consider containers based on Ubuntu 13.10 everywhere, on the mobile device, on the Cloudlet and on the Cloud. In this model both the client and the Cloudlet need to use a Linux based OS.

In Docker mechanism, images are the result of a recursive mounting of different image layers. Each image layer has a parent image layer except for the root image layer. If we consider again the example of *geany* program, we will have an image containing an Ubuntu OS layer

enriched with the *geany* layer that is offloaded by the mobile device to be added to the Cloudlet image so that the new Docker image within the Cloudlet is the one given in Figure 7.11.

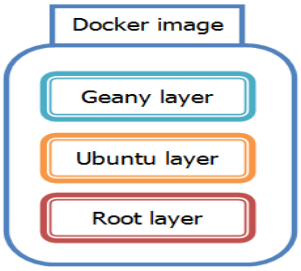


Figure 7.11 Docker image

We noticed that the Docker images can start much faster than VMs (less than 1 second compared to 11 seconds on our hardware) because unlike VMs, Docker images are lightweight implementations and the *geany* layer that is offloaded from the mobile to the Cloudlet has a smaller size than that of the overlay.

7.4.3 OSGi based solution

OSGi is natively running on Java Virtual Machine but without inter framework hypervisor. OSGi has been tested as a middleware running on the operating system. Our tests, generally, show that OSGi based solution has equivalent and even better performances than Docker-container and VM based solutions. We will illustrate this through the following example. In our tests, we took programs with the equivalent size as in the following: an OSGi bundle of 1.2 KB, a program of 1.5 KB to be run in Elijah environment and a program of 1.4 KB to be run on a Docker image, as presented in Table 7.2. The overlay computed for Elijah program reaches 1.4 MB while the Docker image containing the corresponding program has a size of 6.6 KB. Table 7.2 shows that the execution delay (9s) is high when the mobile device offloads the Docker container with its required application, which is heavy to outsource. In addition, the mobile client needs to carry with him the whole container (~600 MB), which is too much for a mobile device.

	OSGi	Elijah	Dockers
Program Size KB	1,2	1,5	1,4
Overlay/layer size		1,4MB	6,6KB
Communication	Internet WiFi	LAN WiFi	Internet WiFi
Execution delay	< 1s	6s	9s

Table 7.2. Cloudlet-based solutions

The Elijah VM overlay presents an intermediate but important execution time (6s). The overlay-based solution offloads heavy overlays even when they are compressed. The VM synthesis operation is time-consuming task. This solution is very sensitive to network variation during the overlay communication. By contrast, OSGi-based solution shows that running a remote bundle inside OSGi Cloudlet is particularly faster (<1sec) remotely on the Cloudlet. Consequently, the performances are significantly enhanced. These tests are done while the OSGi framework is already running. R-OSGi allows the required services to be executed. There is no additional layer generation during the execution. Only the JVM and the OSGi framework are required.

7.4.4 Performance comparison

To compare the difference among these discussed Cloudlet implementations, we consider calculating the execution time locally by examining four different sizes of programs as presented in Table 7.3.

We then compare these programs under OSGi, Elijah and Docker platforms to measure their execution time. In this test, communication is established via Wi-Fi to access to the Cloudlet server. Figure 7.12 presents time in seconds for downloading and launching the overlay for Elijah, OSGi bundle, and Docker layer.

	Program size		
	OSGi	**Elijah**	**Docker**
Program 1	3.8 kb	4.96 kb	4.96 kb
Program 2	1.22 Mb	1.22 Mb	1.22 Mb
Program 3	7.922 Mb	7.923 Mb	7.923 Mb
Program 4	15.843 Mb	15.844 Mb	15.844 Mb

Table 7.3 Size of the tested programs

Generally, increasing size of program leads to a rise of execution time for all models. Although transferring an overlay via high speed LAN Wi-Fi, however, Elijah takes more time in comparison with the others especially soar to double time in Program 4 which is the heaviest program. The concept of a VM overlay is similar to copy-on-write virtual disk files or VM image hierarchies [83]. It means that, the overlay solution is time-consuming task because of the heavy overlay offloading.

On the other hand, with installed Java virtual machine and OSGi framework on each node, applications supporting OSGi can interact with a proxy bundle which is generated dynamically with the exported methods without transferring any heavy code. As a result, running bundle in OSGi framework is the fastest solution in term of time-consuming. The remaining solution, Docker, has insignificant higher execution time than OSGi due to the layer transferring.

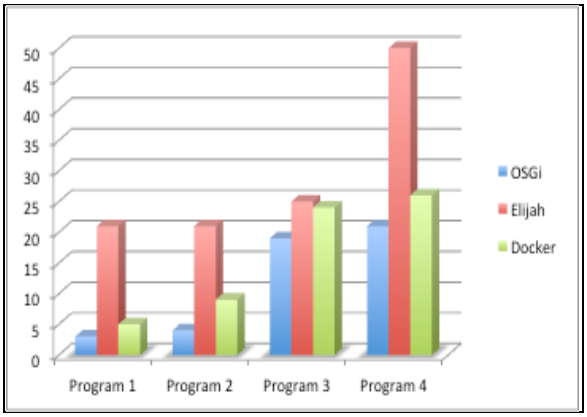


Figure 7.12 Time in seconds for downloading from the mobile device and launching the overlay, OSGi bundle, and Docker layer on the Cloudlet

7.5 Conclusion

In the two previous chapters, we proposed a solution to integrate OSGi as a framework for mobile Cloud computing, then we classified this design in three main architecture models.

First, the traditional model used for centric Cloud services is presented and we show how OSGi based SOA framework can fit to this model. Second, we adapt the first model to a

distributed Mobile-to-Mobile Cloud computing in order to deal with dynamic mobile and connected objects interaction. In the third part, we adapt our model to 3-tier Cloudlet design, and we explain how this model can fulfil IoT network constraints and can be suitable for hostile environments.

In the MobiCloud environment, performance evaluations are conducted by accessing a benchmark of the MCC-OSGi components, e.g., on the mobile devices settings, and on the networks capabilities (Wi-Fi, 3G, etc). In this chapter, we evaluated the performances of the remote service invocation regarding our new architecture models, to highlight the benefit of our proposed solution in terms of performance.

For each context, we explored the performance of our service-oriented framework, and contrasted it with alternative existing solutions. We explained, in this chapter, how OSGi framework can be a suitable choice for remote interaction in different design constraints and use cases. A comparison with well-known solutions like VM, and Docker containers demonstrated the power of OSGi SOA middleware.

Different service architecture models have been discussed and proposed until now while dismissing the security aspects regarding bundle execution and communication. The next chapter investigates the way to make OSGi based solutions more secure during the design of these architectures in respect to the concept of security by design.

Chapter 08: Enhancing MCC Framework Security

8.1. Introduction

The use of MCC frameworks optimizes the performance, compatibility, and scalability of services in mobile computing environment. However, despite the recent efforts conducted on MCC security, a number of challenges still remain to offer secure MCC frameworks while dealing with multi-context environment and with the expected adaptability and energy-efficiency.

We showed in the previous chapters how we can adapt a unique modular middleware to the different MCC use cases and contexts. SOA modularity and service reuse with OSGi bundles, offer an adaptive and efficient solution for application framework.

Although low-coupled middleware architecture can considerably improve the MCC services efficiency, the security issue remains one of the main drawbacks with extensible applications that may execute untrusted codes locally or remotely, with network security challenges, data segregation and data access control, authentication, and data integrity constraints.

The security issues need to be analyzed in order to enhance the global security without altering the dynamicity of the proposed design. Security aims to guarantee the privacy and the availability of data and user information. Indeed, even when developers don't plan to run the framework with security enhancements, the framework has to be security-context aware to guarantee applications working in security enabled environment.

Performance penalty should also be considered while defining the security of mobile framework.

In this chapter, we give an overview of the MCC middleware security challenges and the limitation of our previous SOA based design. We then present propositions for the enhancement of the MCC framework security for the different designs.

8.2 Security considerations

In this part, we provide a characterization of the security threats that may affect the MCC paradigm. We explore the particular security weakness of our previously proposed framework.

In the following we analyze the security issues from two aspects: global MCC risks related to the different MCC entities (devices, Cloud servers, networks) and specific framework risks related to the OSGi based design.

8.2.1 Global challenges

8.2.1.1 Server Cloud system

Many security challenges exist in the Cloud computing. These risks can be classified depending on the provided service level:

- In IaaS model: in addition to the security related to the hardware and the infrastructure of Cloud computing, the main challenges lie in virtualization to secure virtual machines and images, virtual local network, etc.
- With the PaaS model, the API (Application Programming Interface) needs to be enforced with secure services that are often open to public access (Internet).
- In a SaaS model, the data-security management is one of the most relevant security objectives, as well as to overcome Web Application vulnerabilities.

8.2.1.2 Network Security

There is a need for a secure communication channel between the Cloud and the mobile device. An unsecure mobile-Cloud communication channel can introduce attacks, at communication level: access control attacks, data integrity risks and availability (denial of service attacks, man in the middle, etc.). The secure routing protocols can be used to protect the communication channel between the mobile device and the Cloud.

8.2.1.3. Data Security and privacy issues

Secure MCC frameworks need to ensure the protection and the privacy of data created and manipulated by mobile devices and by the Cloud, as well as the confidentiality of data transferred through the communication-network channel. Data security includes the availability of information, the integrity of data used on remote providers, and the access control to sensitive data. The data security is closely linked to the application security.

8.2.1.4. User and mobile-device authentication

One of the most challenging aspects in MCC is to guarantee the user privacy and the provisioning of mobile-applications security for Cloud-resources use. MCC security includes user authentication services and user management, through authentication protocols 802.1X, digital certificates, and multi factor authentication. As stated in [87], authentication is proved thanks something we know (like password or PIN), something we have (like a token or a smart card), something we are (like using biometrics as fingerprints), or either location attributes, etc. Depending on the number of parameters we consider for authentication, we can have two-factor authentication, three-factor authentication, etc.

8.2.2 Framework issues

8.2.2.1 Running environment

On the Cloud servers, the segregation between execution environments is generally applied per client or per application or group of applications using virtual machines.

Mobile operating systems integrate more and more virtualization mechanisms. A good example is the Android Dalvik virtual machine, which allows running the same program on a variety of devices, regardless of their technical features. Each application runs on a Dalvik virtual machine.

However, in our previous OSGi based model, all local services and bundles of the framework are executed on the same virtual machine without considering the need of segregation between untrusted applications (see Figure 8.1). Hence, there is a need to enhance the running environment segregation between applications or groups of applications with different sensitivity or trust levels.

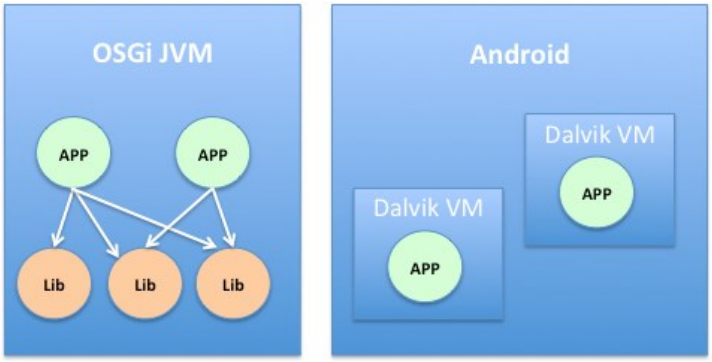


Figure. 8.1: Segregation limitation

8.2.2.2 Weak service identification

Service delivery in our proposed framework is based on low-coupled bundles interaction. We know that the bundles can be local to the mobile device or hosted on remote devices or VMs.

The identification of bundles in OSGi is based on bundle IDs (string objects), which is easily compromised since it's a simple string in the related Manifest XML file.

By compromising the existing ID, the bundle identity is usurped. Therefore, malicious bundles can either execute destructive operations or obtain access to sensitive data, which may compromise the privacy and/or the integrity of the user's data.

8.2.2.3 Software integrity

One of the most challenging points in Cloud middleware is the software integrity. Since services are executed remotely, the Cloud client (that is, the mobile device) has limited means to guarantee that the provider will execute the right expected bundle.

Data integrity is easier to guarantee in storage as a service because the client is the initiator of the data and can hold an image and can protect them using security mechanisms such as signature, MAC (message authentication code), etc.

When the provider is totally trusted, authentication between the service provider and the consumer in addition to the use of a secure communication channel, allows executing a secure remote service.

However when the provider is not trusted, the challenge becomes more complex since the service consumer holds no control on the remote environment.

8.2.2.4 Manifest advertisement

As presented in Chapter 6, the discovery and the advertisement processes are based on the bundle characteristics listed in the Manifest file. Manifest messages are plain XML files. This gives MCC entities especially publisher entities the risk to modify the advertisement. Hence, XML data transfer need to be secured.

8.3 Objectives

In this chapter, we particularly focus on the following objectives:

Local software security:

How to guarantee the execution of a secure code on local mobiles where potential malicious applications are running? How to offer the environment segregation when needed, while keeping the advantages of OSGi service reuse?

Remote execution:

How to ensure that the advertised bundles are the imported ones? How to control the integrity of a bundle executed on a remote device?

Keeping design modularity:

Security must be intrinsic to the framework code and not integrated as an additional layer. In fact, internal mobile security must prevent malicious external code from executed undesired actions.

In order to bring answers to these questions, the following section is organized in three main parts:

Firstly, we consider the traditional centric model where the mobile device acts as a service consumer. We focus in this case, on local risks in the mobile side and we show how to enhance local OSGi framework security by adapting the framework and the bundle structure, and by securing the execution environment.

Secondly, we consider the additional security risks introduced in an M2M context where the mobile device acts as a service consumer/provider, to address the risk of unknown node that can execute malicious codes, where bundle and service integrity is crucial in view of a remote context.

Finally, we focus on the Cloudlet three-tier model, and we explain the additional risks introduced by the Cloudlet. We explain how the Cloudlet can bring a real efficient solution for security-as-a-service scheme with a lower cost.

8.4 Centric MCC framework security

The service consumer and the service provider are the main entities of this model. These two entities only share knowledge about the service interface and the protocols they use to communicate. Security at PaaS level in this model is related to three main points:

- **Network**: Secure communication channel establishment after mutual authentication process between the provider and the client;
- **Service provider**: Cloud service provider platform.
- **Service consumer**: Security on the mobile framework.

Existing communications protocols can guarantee security properties like authentication using SSL, TLS, etc. However, messages can carry complex data and executable code that need to be secured once hosted in the mobile device.

We focus in this part on the mobile framework security and we propose the following contributions:

1. Enhancing the bundle structure;
2. Adding an OSGi module permission control and a life cycle permission control; and
3. Improving the bundle life cycle to take into account sensitive bundles (framework, admin bundle, R-OSGi bundle, etc.)

8.4.1 Bundle-structure Enhancement

As shown in the previous chapters, the bundle is considered as the deployment unit in our framework design, since it is a self-contained unit which explicitly defines its dependencies with other modules and services.

In addition to basic bundles presented in the framework, we introduce a new bundle category called secure bundles, as in the following. Generally, a standard bundle is composed of:

1. The Manifest file,
2. The bundle name,
3. The bundle version,
4. The bundle-symbolic name,
5. The *Bundle Activator:* defines an optional activator class which implements the *Bundle Activator* interface,
6. The *Bundle-Required Execution Environment*: that specifies which Java version is required to run the bundle, the *Bundle-Class Path*, and finally the dependencies related to the imported and exported packages.

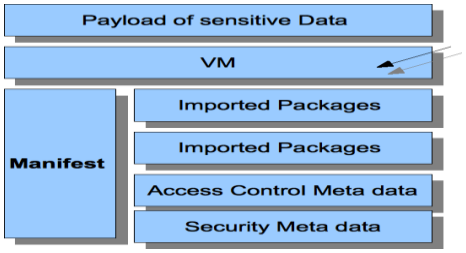


Figure 8.2. Enhanced secure bundle

As depicted in Figure 8.2, in our solution, we propose to enhance the data of the bundle scheme with the following items:

1. **Crypto ID:** is the security bundle ID encrypted with the framework key;
2. **Provenance bundle**: is the digital signature of the service bundle provider;
3. **Integrity check information**: is based on Message Authentication Code;
4. **Access control metadata**: determines the required access level of the client bundle to invoke a remote bundle provider; and
5. **VM**: A dedicated virtual machine that controls the resource code of the bundle.

A secure bundle is a bundle that has access to sensitive data or execute critical operations (admin, framework launch, etc.).

8.4.2 Local security policy: module and lifecycle permissions control

As seen in the OSGi chapter, Java security architecture offers permission mechanisms (*java.security.Permission*) to allow access to specific and sensitive operations, such as file systems or network socket access.

To map the domain-based security to the right code, Java uses a special concept called a protection domain, which is represented by the *java.security.ProtectionDomain* class, to encapsulate the security characteristics of the domain. In OSGi, since a domain is mapped one-to-one with a bundle, a domain protection is simply a bundle protection domain.

We propose in this part to enhance the control of the framework code to deal with malicious codes executed in the local mobile framework or in different frameworks and applications running in the mobile environment.

As shown in Figure 8.3, all classes originating from a given bundle are members of the same bundle protection domain.

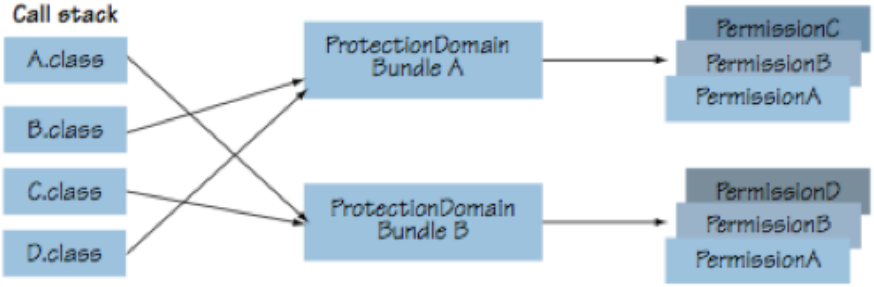


Figure 8.3. Bundle domain protections

In respect to OSGi bundles, we identify two risks related to bundle interaction as in Table 8.1.

Layer	Risk
Malicious import /export	A malicious bundle can import a malicious code in a specific package.
Wrong dependencies	A bundle can use the dependencies listed in the manifest file to change the dependency tree of a bundle and add a dependency with a malicious bundle.

Table 8.1. Bundle risks

To overcome the risks listed in Table 8.1, we integrate in the framework architecture two permissions control levels as in Table 8.2.

Layer	Permissions
Module Intra Bundle	Package Permission: controls which package a bundle is allowed to import and/or to export.
Module Inter bundle	Bundle Permission: controls which bundle is allowed to be required according to the dependencies.

Table 8.2. Module permissions

The proposed bundle permission has the structure described in Figure 8.4 including three actions: *getActions*, *involvePermission* and *newPermissionCollection*.

```
java.security.BasicPermission
org.osgi.framework.BundlePermission
```
BundlePermission(authority to provide and/or require, actions)

Methods

- ***getActions*** *: Returns the representation of the BundlePermission actions.*
- ***involvePermission****: Determines if the specified permission is involved by this object.*
- ***newPermissionCollection*** *: Collects new permissions.*

Figure 8.4. Local permission implementation

The actions that a bundle is allowed to perform are based on the permissions it has. The bundle must have the appropriate permission for each action as:

- Invoking a service;
- Importing or exporting a package; or

- Reading and/or writing a file.

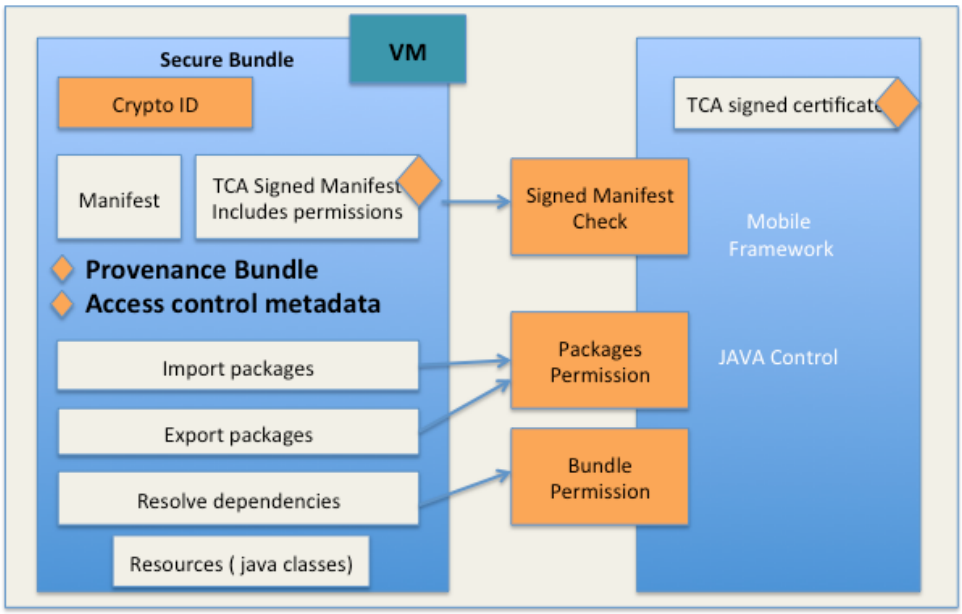


Figure 8.5. Bundle packages and dependencies control

We propose to include in the bundle its self-permissions by securing the Manifest file. The guarantee of the authenticity of the Manifest file allows checking the packages and the dependencies as depicted in Figure 8.5. The Manifest file includes the permissions at package and dependencies levels, and the Manifest file is signed by the Cloud Trusted Authority (CTA) during the publication step.

8.4.4 Bundle lifecycle evolution

Secure bundles have a unique identity, that is, Crypto ID which is the security bundle ID encrypted with the framework key. This identity is kept during the life cycle of the bundle, unless it is uninstalled and reinstalled. In addition to the classical bundle states (installed, resolved, starting, active, stop, and uninstall) as in Figure 8.6, we enrich the life cycle with four other states (see Figure 8.7) as in the following.

1- *Published* state: is the state of the bundle in the publisher side before the CTA signed it;
2- *Approved* state: the framework authenticates the bundle;
3- *Authenticated* state: means the bundle is valid;
4- *Rejected* state: if the bundle signature is not valid or doesn't exist.

The framework performs a bundle signature control before the bundles are installed. If the signature of a given bundle is invalid, the bundle is rejected and the installation is aborted. When the bundle is accessible remotely, the authentication is done before remote execution.

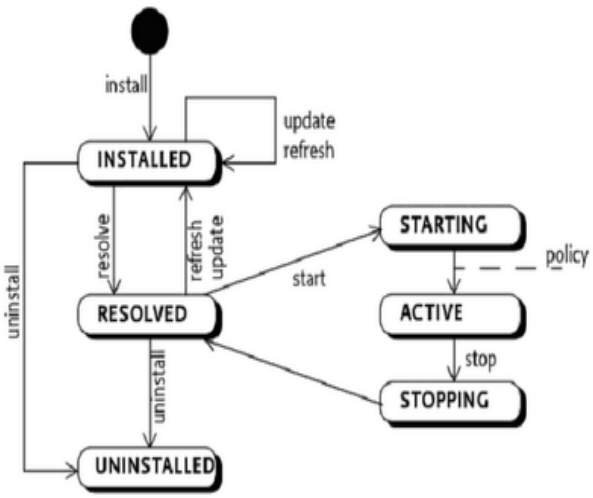


Figure 8.6: Classical Bundle life cycle

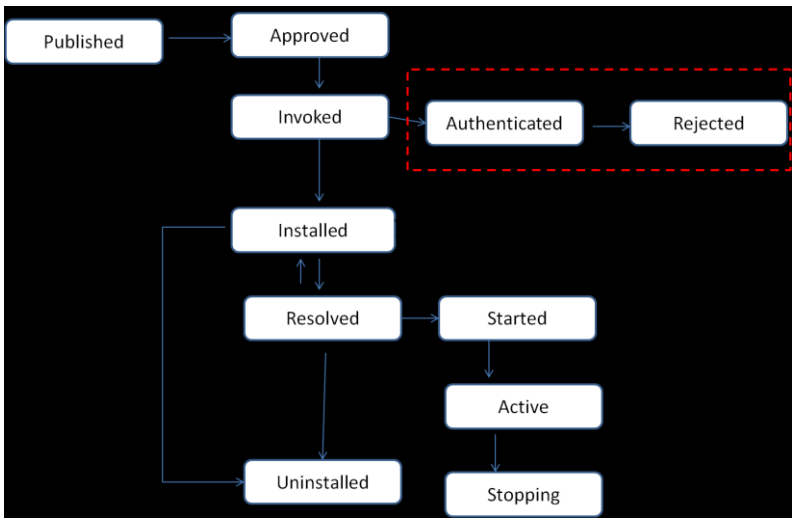


Figure 8.7. Enhanced bundle life cycle

The life cycle of a bundle includes critical operations that can either stop/uninstall a mandatory bundle or launch/install an undesired bundle. The control of the life cycle of the mobile bundles is important to avoid losing admin control. Hence, we attach to the framework Admin permissions to allow only particular signed secure bundles with a crypto ID and a certificate performing sensitive lifecycle operations (Install, Uninstall, Stop, etc.). The filter gives access to the following parameters:

- **Provenance Bundle** - The location of a bundle; and
- **Crypto ID** - The bundle ID of the bundle.

Admin permission P gives a bundle's authority to perform specific privileged administrative operations or get sensitive information about this bundle.

8.5 Mobile to Mobile framework security

In a mobile to mobile scheme, each mobile device can be a provider and a consumer. A mobile device may publish and subscribe, join or leave the network. However, mobiles nodes

may behave maliciously and thus compromise the MCC service. In fact, there are two main weaknesses added in the previous secure scheme. The service provider can be any mobile node, and the invoked bundles are remote. Unlike the centric model, the provider is not trusted in M2M distributed design. Hence, there is a need to control the authenticity of the advertisement process to guarantee that the executed bundles are really the discovered ones.

Moreover, the sensed information such as location, health, personal information, should be processed and stored in a secure fashion to protect user's privacy in the Cloud.

The previous permissions mechanism, presented in the centric model, provides a powerful mechanism to control access inside the bundle and between the bundles. However, this solution is limited to the bundles that are local to the framework. Besides, the OSGi framework has no control on remote bundles.

In mobile to mobile distributed model (see Figure 8.8), we need to secure the interaction between frameworks and their attached bundles. Thus, the objective here is to secure a bundle execution without knowing the provider host. With the mobile Cloud computing, a service may reside anywhere, which is an issue that needs to be addressed.

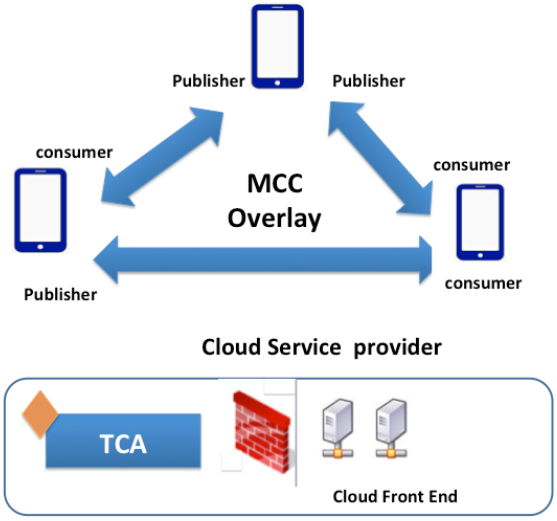


Figure 8.8. Mobile to mobile interaction

The authenticity of the framework is necessary to perform bundle and actions control locally. To manage permissions in a remote mobile Cloud context, we need to secure interaction by:

1. Authenticating the frameworks: to authenticate the provider and to ensure that the bundle is compliant with the system; and by
2. Delegating the control of the remote bundle to its authenticated framework.

8.5.1 Framework authentication model

In this part, we propose a securing approach that can be used on unknown hosts. The objective is to create a delegation model (see Figure 8.9) in which mutual authentication between signed frameworks allow a secure exchange.

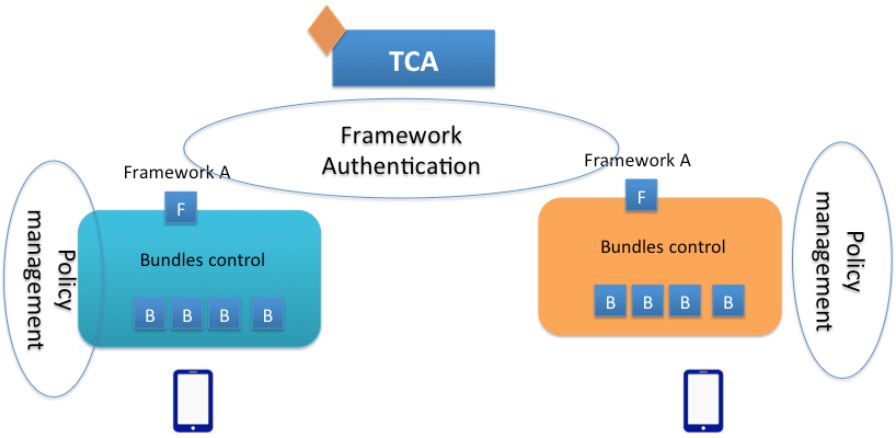


Figure 8.9. Mobile to mobile delegation security model

Cloud Trusted Authority: We introduce the Cloud Trusted Authority (CTA), in the global MCC scheme. The CTA is installed on the Cloud infrastructure or can be a third trusted part.

Before downloading the framework on mobile devices, each framework is signed by the Cloud Trusted Authority (CTA). The framework signature involves a Public key PB_F and a private key PV_F. The public key is shared with the mobile framework in the form of a certificate, while the private key is kept secret in the mobile side to authenticate the sent data. This checks that the sender has access to the private key and that the data has not been modified. With this signature mechanism, the client framework authenticates that the provider framework is signed by the TCA, and that this provider framework respects the *local security policy* for its attached bundles.

The **local security policy** is the set of permissions installed on the framework to guarantee the secure execution of the different bundles as shown in the centric model.

The TCA configures each framework to allow restricted set of permissions to mobile bundles, after which the signer can create bundles that use those permissions or some subsets, without any additional intervention or communication with the central Cloud TCA service.

8.6 Cloudlet model and integrity check

The Cloudlet service provider can be seen as a platform as a service (PaaS) on which dedicated VMs are pre-installed. Once a Cloudlet is discovered, a three-part authentication is done involving the mobile device, the Cloudlet and the Cloud where the Cloud is the trusted authority. For example, in [86], Bouzefrane et al. showed how to perform authentication in the Cloudlet architecture context while focusing on NFC applications.

We propose in this part to focus on the secure bundle integrity check using Cloudlets. The Message Authentication Code (MAC) is used to authenticate a message in order to confirm its origin and to ensure its integrity, that is, the message has not been changed in transit. The MAC algorithm uses as input a secret key and an arbitrary-length message to be authenticated, and outputs a MAC tag. The MAC value protects both a message's data integrity as well as its authenticity, by allowing verifiers who possess the secret key to detect any changes to the message content by comparison.

We propose to use this protocol to authenticate remote imported bundles following three main steps (see Figure 8.10):

- The TCA generates a random key for each framework provider;

- The provider uses a signing algorithm and returns a tag given the framework key and the bundle;
- A verifying algorithm efficiently checks the authenticity of the message given the key and the MAC.

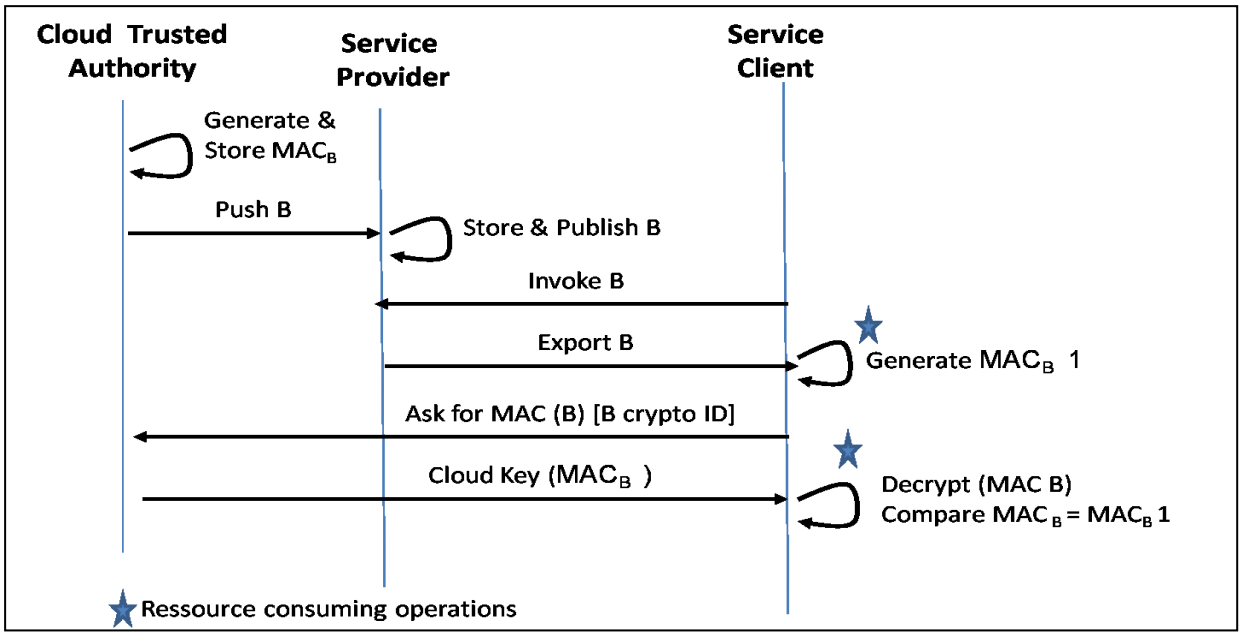


Figure 8.10. MAC–based integrity control for distributed model (without Cloudlet)

For each published bundle, the CTA generates a MAC and stores it. The bundles are then installed on the different providers. When a service consumer invokes the bundle, it is exported. The client performs than a MAC calculation. The client asks the TCA for the corresponding MAC and then compares the calculated value with the received one to check integrity.

The main limitation with this model is that the client performs resource consuming calculation. We propose in the Cloudlet model to offload the MAC calculation on the Cloudlet to reduce energy consuming on the mobile side, as mentioned in Figure 8.11.

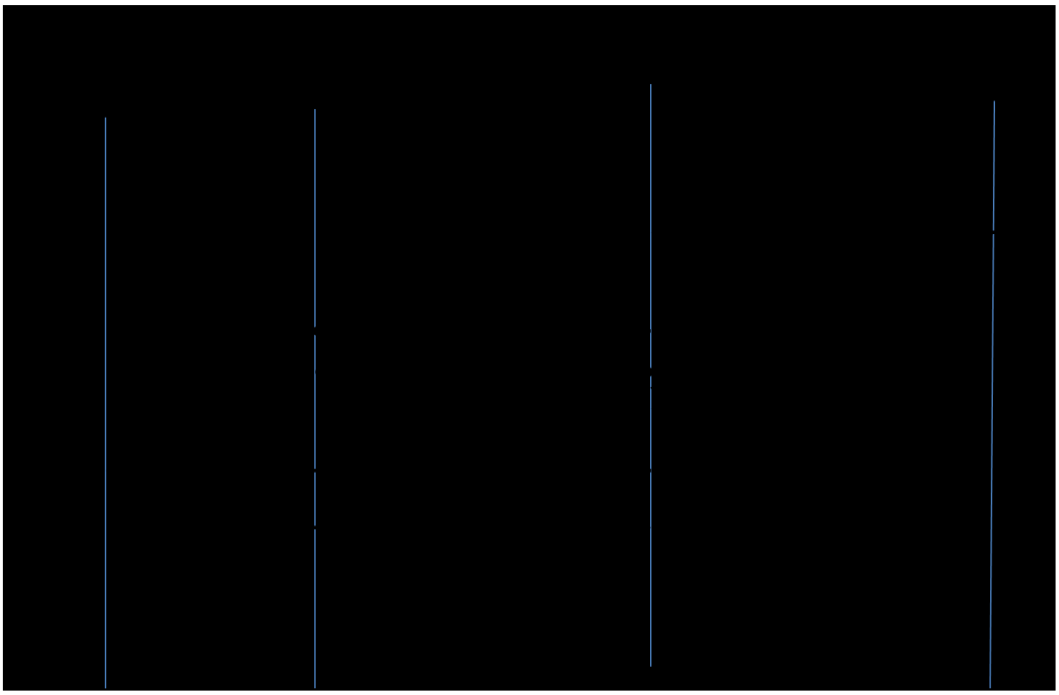


Figure 8.11 MAC –based integrity control using Cloudlet model

8.8 Conclusion

With the growth in the popularity of mobile Cloud computing, privacy and security have become a critical concern. Because of resources limitation, the security solutions proposed for the Cloud computing environment cannot be directly run on a mobile device. Indeed, there is a strong need for an efficient and effective privacy-preserving system based on a lightweight approach.

We show, in this chapter, how to combine OSGi mechanisms and authentication and integrity security protocols to provide the infrastructure to deploy and manage services in secure environments. First, we enhance the local framework security based on the provided Java security mechanisms. Then, we introduce the security in a modular way by adapting the structure of the bundle itself and its life cycle and by integrating OSGi permissions to express security policies for bundles.

To keep the model secure but simple, we use framework digital signature to assign permissions to bundles based on who signed them. In this case, a simple way to enforce the permissions needed by a bundle is the specification of local permissions inside of the bundle itself.

Additionally, we see how in distributed mode, the challenge is more complex since the provider is not trusted any more, and how we need to check the authenticity of the provider before using its exported bundles and how a delegation model can help to deal with unknown provider issues.

Finally, we focus on the Cloudlet model to show the additional issues brought by the Cloudlet entity, and the way to overcome the issues, before giving some propositions to go a step further and enhance security thanks to the Cloudlet security and integrity controller.

In the future work, we aim to establish performance tests to evaluate the cost of security enhancement in the MCC framework according to the different designs.

Chapter 09: Conclusion

We are convinced that dealing with the growing complexity of software, the emergence of mobile applications and the high performance requirements facing a big data and remote Cloud services context, need to consider modularity, security and life cycle of the services directly inside the design of the mobile framework and OS layers considerably improve the Cloud services by optimizing the remote services invocation. Our modest contribution presented in this dissertation aims to consolidate this firm conviction.

The growth of connected devices, mostly due to the large number of Internet of things deployments and the emergence of mobile Cloud services (MCC), introduce new challenges for the next mobile Cloud computing service architectures.

We explain in this PhD thesis how MCC needs dynamic frameworks to provide elasticity for better distribution and more scalability while dealing with limited environment resources and variable mobile context (Web applications, real-time, enterprise services, mobile to mobile, hostile environments, etc.). We focus on MCC comprehension to understand the high service potential of mobile devices and the power of the great and various number of mobiles connected objects.

We show, in the first part of this work, how service oriented architecture can be a key solution to provide distributed mobile Cloud services and how OSGi platform particularly can be an adaptive and efficient framework to provide such implementation thanks to the bundle structure, the integrated management and portability offered with Java Virtual Machine. We aimed at using facilities offered by OSGi in Android platforms. We proposed a solution to integrate Felix, a lightweight implementation of OSGi specification. OSGi solutions allow Android applications to take advantage of the facilities offered by OSGi such as to update bundles dynamically without restarting the involved applications, to manage the versioning, to share services between distinct applications, and to check the dependencies before the application launching avoiding errors during execution. Then, we identified in Remote OSGi platform an excellent support for dynamic services invocation on Platform as a Service (PaaS) mobile Cloud, thanks to the transport abstraction through OSGi remote services and dependency management using the OSGi framework capabilities. We illustrate the use of remote OSGi in Cloud context within Android platforms, and we explain how OSGi can considerably improve the Cloud services by optimizing the remote services invocation.

In the second part, we adapt the proposed MCC framework to different architecture contexts: the first is a traditional centric model where mobile devices are only consuming services, the second is a distributed model where the power of mobiles to mobiles interaction offers unlimited value-services opportunities, and the third is a three-tier architecture based on the Cloudlet notion. For each context, we explore the performance of our service oriented framework and contrast it with alternative existing solutions.

In the third part, we investigate the security issues in two main axes:

- Securing the mobile OSGi clients by controlling the remote service invocations in a distributed Cloud context, by adapting the framework structure by design.
- Securing the Cloud providers from malicious service invokers by performing common policies negotiation mechanisms between the provider and the client of the remote services.

Regarding future works, we aim to extend the OSGi based mobile Cloud to multiple distributed Cloud service sites to perform scalable and performing evaluation of our distributed service interaction mechanisms.

In addition to improve the security of the framework, we aim to focus on the following remaining challenges:

- Communication channels are considered to be secure in our model, by using secure protocols, however in our context we need to address the efficiency and the security issues for mobile Cloud networking and communication, since each mobile device establishes a secure VPN proxy service through the mobile Cloud computing. Many to Many security protocols can help to define distributed security approaches that are more suitable for MCC context.

- In addition, the performance comparison needs to be conducted to evaluate the cost of the security approach, and by comparing the framework performance and scalability before and after enhancing its security.

Finally, we believe that Service Oriented Architecture is an ideal paradigm for mobile Cloud services, and the integration of its modularity in the low layers of mobile operating systems as seen in our experimentation with Android to help highly improve performances.

As future work, we can investigate on developing additional components using a service modular structure (like the Andro-modules) into the mobile operating system like discovery, authentication, and security protocols.

Chapter 10: Publications

This research work has led to various publications that are given here:

International Journal:

1. F. **Houacine**, S. Bouzefrane, A. Adjaz. "Service Architectures for multi-environment Mobile Cloud Services", International Journal of High Performance Computing and Networking, vol. 9(4), pp. 342-355, 2016.

International Conferences

1. T. Le Vinh, R. Pallavali, F. **Houacine**, S. Bouzefrane. "Energy efficiency in Mobile Cloud Computing Architectures", The IEEE 4th International Conference on Future Internet of Things and Cloud, August 2016, pp.to appear, Austria.
2. S. Bouzefrane, A. Benkara Mostefa, F. **Houacine**, H. Cagnon. "Cloudlets Authentication in NFC-based Mobile Computing", 2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, April 2014, pp.267 – 272, Oxford, GB, 2013.
3. M. Zneika, H. Loulou, F. **Houacine**, S. Bouzefrane. "Towards a Modular and Lightweight Model for Android Development Platforms", 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, August 2013, pp.2129-2132.
4. F. **Houacine**, S. Bouzefrane, D. Huang, Li Li. "MCC-OSGi: An OSGi-based Mobile Cloud Service Model", The IEEE ISADS (Eleventh International Symposium on Autonomous Decentralized Systems), March 2013, pp.37-44, Mexico.

Industrial Conference

A. Adjaz, S. Bouzefrane, F. **Houacine**, P. Paradinas. "An OSGi Based Service Architecture in the Context of Tag-to-MobiCloud Computing", Chip-to-Cloud Security Forum, Nice, France.

References

[1]. Andre Bottaro and Fred Rivard, “OSGi ME - An OSGi Profile for Embedded Devices”, Eclipse Summit, Germany, 2009.

[2].[The NIST Definition of Cloud Computing. National Institute of Standards and Technology. Retrieved 24 July 2011.

[3]. Ajey Singh, Maneesh Shrivastava, “Overview of security issues in Cloud Computing” , International Journal of Advanced Computer Research (IJACR), Volume-2, Issue-3, March-2012, pp.41-45.

[4]. Chris Matthews, “Virtualized Recomposition: Cloudy or Clear?”, Yvonne Coady University of Victoria, to complete.

[5]. Abdul Nasir Khana, M. L. Mat Kiah, Samee U. Khanb & Sajjad A. Madanic. (2012). “Towards secure mobile cloud computing": A survey. Future Generation Computer Systems, 08-003, pp. 1-5.

[6]. D. Huang, X. Zhang, M. Kang, and J. Luo, “MobiCloud: Building Secure Mobile Cloud Framework for Mobile Computing and Communication”. In Proceedings of the 5th IEEE International Symposium on Service-Oriented System Engineering (SOSE), pp. 27-34, 2010.

[7]. X. Li, H. Zhang and Y. Zhang, Deploying Mobile Computation in Cloud Service, Lecture Notes in Computer Science, Vol. 5931, Cloud Computing, pp. 301-311, 2009.

[8]. Amir Taherkordi and Frank Eliassen, “Towards Independent in-Cloud Evolution of Cyber-Physical Systems”, Department of Informatics University of Oslo, Norway Email: famirhost,frankg@ifi.uio.no

[9]. Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing (Draft). January, 2011.

[10]. Satyanarayanan, “Mobile Information Access”. IEEE Personal Communications, Vol. 3, No. 1, February 1996.

[11]. http://www.nist.gov/

[12]. OSGi in Depth paperback – December 21, 2011. Alexandre de Castro Alves

[13]. Reference Model for Service Oriented Architecture 1.0, Committee Specification 1, 2 August 2006line 863 (https://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf)

[14]. Hashimi, S., “Service-Oriented Architecture Explained”, O’Reilly ONDotnet.com, August 2003.

[15]. Kishore Channabasavaiah, Edward Tuggle and Kerrie Holley, “Migrating to a service-oriented architecture”, Dec. 16, 2003, IBM DeveloperWorks (http://www.ibm.com/developerworks/library/ws-migratesoa/)

[16]. N. Bussière, D. Cheung-Foo-Wo, V. Hourdin, S. Lavirotte, M. Riveill, and J.-Y. Tigli. “Optimized contextual discovery of web services for devices”. In IEEE Int. Workshop on Context Modeling and Management for Smart Environments, Volume 2, pp. 707–712, October 2007.

[17]. J. Coutaz and G. Rey. “Foundations for a theory of contextors”. In book entitled “Computer-Aided Design of User Interfaces III”, 2002, pp. 13–33, Springer Netherlands Editor.

[18]. Johnneth Fonseca, Zair Abdelouahab, Denivaldo Lopes and Sofiane Labidi, “A security framework for SOA applications in mobile environment”, International Journal of Network Security and Its Applications (IJNSA), Vol.1, No.3, pp. 90-107, October 2009.

[19]. Decker, M, and Bulander, R. "A Platform for Mobile Service Provisioning Based on SOA Integration", In Communications in Computer and Information Science, Vol 23, 2009, pp 72-84, Springer Berlin Heidelberg.

[20]. Natchetoi, Y. Kaufman, V. and Shapiro, A. "Service-oriented architecture for mobile applications", Proceedings of the 1st international workshop on Software architectures and mobility / International Conference on Software Engineering, pp 27-32, 2008.

[21]. Christian Vecchiola, Xingchen Chu, and Rajkumar Buyya, “Aneka: A Software Platform for .NET-based Cloud Computing”, High Speed and Large Scale Scientific Computing, pp. 267-295, W. Gentzsch, L. Grandinetti, G. Joubert (Eds.), ISBN: 978-1-60750-073-5, IOS Press, Amsterdam, Netherlands, 2009.

[22]. Ying Huang et al., "A Framework for Building a Low Cost, Scalable and Secured Platform for Web-Delivered Business Services," IBM Systems Journal, November 2009.

[23]. Daniel Szepielak, “REST-Based Service Oriented Architecture for Dynamically Integrated Information” IBM research report, pp. 7-12, Nov. 2006:

[24]. (http://domino.watson.ibm.com/library/CyberDig.nsf/papers/6038BDC6C6A3F941852572360065D911/$File/rc24118.pdf)

[25]. [14] Anuraj Ennai and Siddhartha Bose "MobileSOA: A Service Oriented Web 2.0 Framework for Context-Aware, Lightweight and Flexible Mobile Applications", In the 12th Enterprise Distributed Object Computing Conference Workshops, pp. 345 – 352, Munich 2008.
[26]. M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing", IEEE Pervasive Computing, vol. 8, no. 4, pp. 14–23, Oct. 2009.
[27]. OSGi: http://www.OSGi.org/Specifications/HomePage
[28]. http://felix.apache.org/
[29]. http://developer.android.com/guide/components/aidl.html
[30]. Jean-Yves Tigli, Stéphane Lavirotte, Gaëtan Rey, Vincent Hourdin and Michel Riveill "Lightweight Service Oriented Architecture for Pervasive Computing", IJCSI International Journal of Computer Science Issues, Vol. 4, No. 1, pp. 1-9, 2009.

[31]. M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," Pervasive Computing, vol. 8, no. 4, pp. 14–23, 2009
[32]. M. Shiraz, A. Gani, R. Hafeez, and R. Buyya, "A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing," IEEE Commun. Surveys & Tutorials, vol. Accepted, November 2012.
[33]. [Computational Offloading or Data Binding? Bridging the Cloud Infrastructure to the Proximity of the Mobile UserHuber Flores. S. Narayana R. Buyya†. 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering
[34]. Energy efficiency of mobile clients in cloud computing. P. Miettinen, K. Nurminen. Nokia Research Center. Proceeding HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. Pages 4-4
[35]. Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges Z. Sanaei, S. Abolfazli, A.Gani, and R. Buyya, IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 16, NO. 1, FIRST QUARTER 2014 369
[36]. Privacy & Security of Mobile Cloud Computing (MCC) Manmohan Chaturvedi Principal Advisor Research & Technology Development Beyond Evolution Tech Solutions Pvt. Ltd.
[37]. https://wiki.eclipse.org/Tutorial:_Building_your_first_OSGi_Remote_Service
[38]. Building Reflective Mobile Middleware Framework on Top of the OSGi Platform G´abor Paller Nokia Research Center, K¨oztelek str. 6, Budapest 1092, Hungary gabor.paller@nokia.com
[39]. A.Adjaz, S. Bouzefrane , D. Huang , P. Paradinas - An OSGi-based Service Oriented Architecture for Android Software Development Platforms, International Conference on Software & Systems Engineering and their Applications , November 2011, pp.1-10, Paris, France.
[40]. https://tools.ietf.org/html/rfc6120
[41]. Mobile Cloud Computing Dijiang Huang, Arizona State University, Arizona, USA dijiang@asu.edu
[42]. M. Tian, et al.,Performance considerations for mobile web services, Computer Communications, Vol.27, No.11, 2004, pp.1097-1105.
[43]. C. Werner and C. Buschmann, Compressing SOAP messages by using differential encoding, Proc.ICWS,IEEE, 2004, pp.540-547.
[44]. L. Richardson and S. Ruby, RESTful Web Services, O'Reilly Media, 2007.

[45]. MOMCC: Market-Oriented Architecture for Mobile Cloud Computing Based on Service Oriented Architecture Saeid Abolfazli1 , Zohreh Sanaei2 , Abdullah Gani3 , Muhammad Shiraz4 Mobile Cloud Computing Research Lab Faculty of Computer Science and Information Technology University of Malaya, Kuala Lumpur, Malaysia Email: abolfazli1 , sanaei2 ,abdullahgani3@ieee.org, muh shiraz@siswa.um.edu.my4

[46]. https://www.osgi.org/
[47]. http://www.eclipse.org/equinox/
[48]. http://www.knopflerfish.org/documentation.html
[49]. http://felix.apache.org/
[50]. The world's smallest OSGi solution. Prosyst. http://www.prosyst.com/ Technical White Paper.
[51]. OSGi in Depth, Alexandre de Castro Alves, Manning Publications Editor, Dec. 2011, 325 pages, ISBN 193518217X, 9781935182177.
[52]. I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso,"Calling the Cloud: Enabling Mobile Phones as Interfaces to Cloud Applications," J.M. Bacon and B.F. Cooper (Eds.), Middleware 2009, LNCS 5896, pp. 83-102, 2009.

[53]. Alexander Pokahr & Lars Braubach, “Towards Elastic Component-based Cloud Applications”. Volume 570 of the series Studies in Computational Intelligence, pp 161-171, Springer International Publisher.
[54]. https://www.osgi.org/developer/design/
[55]. H. Schmidt, J. Elsholz, V. Nikolov, F. J. Hauck, R. Kapitza, OSGi4C: enabling OSGi for the Cloud”, In Proceedings of the Fourth International Conference on COMmunication System softWAre and middleware (COMSWARE’09, pp. 15:1 - 15:12, Dublin, June 2009.
[56]. J. S. Rellermeyer, G. Alonso, and T. Roscoe. R-OSGi: Distributed Applications Through Software Modularization. In R. Cerqueira and R. H. Campbell (eds.), Middleware ’07, LNCS Vol. 4834, pp.1-20, Springer, Heidelberg, 2007.
[57]. J. S. Rellermeyer, G. Alonso, T. Roscoe: Building, Deploying, and Monitoring Distributed Applications with Eclipse and ROSGi. In: Fifth Eclipse Technology eXchange (ETX) Workshop (in conjunction with OOPSLA 2007), Montreal, Canada, pp. 50- 54, Octerber, 2007.
[58]. S. Mohammed, D. Servos, J. Fiaidhi, Developing a secure distributed OSGi Cloud computing infrastructure for sharing health records , In Autonomous and Intelligent Systems Lecture Notes in Computer Science, Vol. 6752/2011, pp.241-252, 2011.
[59]. C. Hang and C. Can, Research and Application of Distributed OSGi for Cloud Computing, In Proceedings of International Conference on Computational Intelligence and Software Engineering (CiSE), pp.1-5, Wuhan, China, dec. 2010.
[60]. Huijun Wu, Dijiang Huang, Yan Zhu, “Establishing A Personal On-Demand Execution Environment for Mobile Cloud Applications”, Journal of Mobile Networks and Applications, Vol. 20, Issue 3, pp. 297-307, June 2015.
[61]. EZdroit project, 2011 : http://www.ezdroid.com/
[62]. S. Bouzefrane, D. Huang and P. Paradinas, An OSGi-based Service Oriented Architecture for Android Software Development Platforms, In Proceedings of 23rd International Conf. on Systems and Software Engineering and their Applications (ICSSEA’2011), pp. 1-10, Paris, Nov. 2011.
[63]. http://felix.apache.org/site/index.html
[64]. http://developer.android.com/guide/components/aidl.html
[65]. http://www.edc4it.com/blog/mobile/what-do-you-know-about-android-runtime.html
[66]. http://people.inf.ethz.ch/troscoe/pubs/middleware07-rosgi.pdf
[67]. F. Houacine, S. Bouzefrane ; L. Li ; D. Huang “MCC-OSGi: An OSGi-based mobile cloud service model” Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium. 6-8 March 2013 Mexico City, Mexico. pp 1 - 8
[68]. M. Zneika, H. Loulou, F. Houacine, S. Bouzefrane, “Towards a Modular and Lightweight Model for Android Development Platforms”, 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, August 2013, pp.2129-2132,
[69]. Saeid Abolfazli, Zohreh Sanaei, Abdullah Gani, Muhammad Shiraz, “MOMCC: Market-Oriented Architecture for Mobile Cloud Computing Based on Service Oriented Architecture”, Mobile Cloud Computing Research Lab Faculty of Computer Science and Information Technology University of Malaya, Kuala Lumpur, Malaysia (http://arxiv.org/pdf/1206.6209.pdf), June 2012.
[70]. M. Satyanarayanan, “Pervasive computing: vision and challenges,” Personal Communications, IEEE, vol. 8, no. 4, pp. 10–17, 2001.
[71]. https://wiki.eclipse.org/Tutorial:_Building_your_first_OSGi_Remote_Service
[72]. O. Kotevska, A. Lbath & S. Bouzefrane, « Toward a real-time framework in cloudlet-based architecture » Tsinghua Science and Technology, Vol. 21, n°1, pp. 80-88, 2016.
[73]. M. Zneika, H. Loulou, F. Houacine, S. Bouzefrane, “Towards a Modular and Lightweight Model for Android Development Platforms”, 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, August 2013, pp.2129-2132,
[74]. M. Satyanarayanan, P. Bahl, R. Caceres, & N. Davies, “The case for vm-based cloudlets in mobile computing,” Pervasive Computing, vol. 8, no. 4, pp. 14–23, 2009
[75]. R. J. Creasy. The origin of the VM/370 time-sharing system. IBM Journal of Research and Development, 25(5):483–490, Sep 1981.
[76]. Simanta, Soumya; Lewis, Grace; Morris, Ed; Ha, Kiryong; and Satyanarayanan, Mahadev. “A Reference Architecture for Mobile Code Offload in Hostile Environments”. Proc. of the Joint 10th Working IEEE/IFIP

Conference on Software Architecture & 6th European Conference on Software Architecture (WICSA/ECSA 2012). August 2012. http://www.cs.cmu.edu/~satya/docdir/simanta-mobicase2012.pdf.
[77]. https://www.docker.com/
[78]. Stephen Soltesz, Herbert Potzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson, "Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors", ACM SIGOPS Operating Systems Review - EuroSys'07 Conference Proceedings, Vol. 41 Issue 3, pp. 275-287, June 2007.
[79]. MobiCloud: Building Secure Cloud Framework for Mobile Computing And Communication. Dijiang Huang, Xinwen Zhang, Myong Kang, Jim Luo 2010 Fifth IEEE International Symposium on Service Oriented System Engineering
[80]. http://xenserver.org/
[81]. https://tools.ietf.org/html/rfc2784
[82]. Simanta, Soumya; Lewis, Grace; Morris, Ed; Ha, Kiryong; and Satyanarayanan, Mahadev. "A Reference Architecture for Mobile Code Offload in Hostile Environments". Proc. of the Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture (WICSA/ECSA 2012). August 2012. http://www.cs.cmu.edu/~satya/docdir/simanta-mobicase2012.pdf.
[83]. Ha, K., Pilai,P., Richer, W., Abe, Y., & Satyanarayanan, M. "Just-in-time provisioning for cyber foraging", pp. 153-166, Mobisys 2013.
[84]. Web site www.Xdelta.org
[85]. Surianarayanan, C., Ganapathy, G. and Ramasamy, M.S. "Towards quicker discovery and selection of web services considering required degree of match through indexing and decomposition of non-functional constraints", Int. J. Computational Science and Engineering, Vol. 10, Nos. 1/2, pp.45-69, 2015.
[86]. S. Bouzefrane, A. Benkara Mostefa, F. Houacine, H. Cagnon, "Cloudlets Authentication in NFC-based Mobile Computing", 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, April 2014, pp.267 – 272, Oxford, U.K.
[87]. Yusuf Bhaiji, "Network Security Technologies and Solutions (CCIE Professional Development Series)", Pearson Education, March 2008, 840 pages.
[88]. Dinh Thai, Hoang & Lee, Chonho & Niyato, Dusit & Wang, Ping. (2013). A survey of mobile cloud computing: Architecture, applications, and approaches. Wireless Communications and Mobile Computing. 13. . 10.1002/wcm.1203.

Fatiha HOUACINE
Service-Oriented Architecture for the Mobile Cloud Computing

Résumé

La croissance des appareils connectés, principalement due au grand nombre de déploiements de l'internet des objets et à l'émergence des services de cloud mobile, introduit de nouveaux défis pour la conception d'architectures de services dans le Cloud Computing Mobile (CCM) du cloud computing mobile. Nous montrons dans cette thèse comment l'architecture orientée services SOA peut être une solution clé pour fournir des services cloud mobiles distribués et comment la plate-forme OSGi peut être un cadre adaptatif et efficace pour fournir une telle implémentation. Nous adaptons le cadre CCM proposé à différents contextes d'architecture. Le premier est un modèle centré traditionnel, où les appareils mobiles sont réduits à consommer des services. Le second est un modèle distribué où la puissance de l'interaction de mobile à mobile offre des opportunités illimitées de services de valeur, et enfin, l'architecture à trois niveaux est considérée avec l'introduction de la notion de cloudlet. Pour chaque contexte, nous explorons la performance de notre cadre axé sur le service et le comparons à d'autres solutions existantes.

Mots clés: Cloud Computing, OSGi, Mobile Cloud, Application Framework, Mobile design

Résumé en anglais

The growth of connected devices, mostly due to the large number of internet of things IoT deployments and the emergence of mobile cloud services, introduces new challenges for the design of service architectures in mobile cloud computing MCC. An MCC framework should provide elasticity and scalability in a distributed and dynamic way while dealing with limited environment resources and variable mobile contexts web applications, real-time, enterprise services, mobile to mobile, hostile environment, etc. that may include additional constraints impacting the design foundation of cloud services. We show in this thesis how service-oriented architecture SOA can be a key solution to provide distributed mobile cloud services and how OSGi platform can be an adaptive and efficient framework to provide such implementation. We adapt the proposed MCC framework to different architecture contexts. The first one is a traditional centric model, where mobile devices are reduced to consuming services. The second one is a distributed model where the power of mobile-to-mobile interaction offers unlimited value-services opportunities, and finally, three-tier architecture is considered with the introduction of the cloudlet notion. For each context, we explore the performance of our service-oriented framework, and contrast it with alternative existing solutions

Key words: Cloud Computing, OSGi, Mobile Cloud, Application Framework, Mobile design