Service Architecture for multi-environment Mobile Cloud Services

Fatiha Houacine, Samia Bouzefrane, Aghiles Adjaz

Service Architecture for multi-environment Mobile Cloud Services

Fatiha Houacine[1], Samia Bouzefrane [2], Aghiles Adjaz

CEDRIC Lab, Conservatoire National des Arts et Métiers – CNAM Paris, France

[1]houcin_f@auditeur.cnam.fr, [2]samia.bouzefrane@lecnam.net

Abstract

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 work 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: Mobile Cloud Computing, Service Oriented Architecture, OSGi platform.

1. Introduction

Gartner[1] predicts the use of 25 "Billion Connected Things" by 2020. The popularity of mobile devices such as smartphones and connected objects are offering ubiquitous communication and information services, creating such a dependency in all domains from our daily life to enterprise services, and even in critical environments such as industry, health and military.

Mobile devices accelerate the emergence of various real-time mobile applications that require a high level of responsiveness and in return need intensive computing resources. Thanks to the Cloud computing paradigm, these mobile applications move to the Cloud instead of being installed and run directly on the mobile devices. Hence, apps are accessed and executed remotely on the Cloud through mobile interfaces.

Mobile Cloud Computing (MCC) aims to extend computing capabilities of mobile devices, storage capacity, and enhance data safety to improve the computing experience of mobile users in a transparent way. However, mobile devices challenges (limited resources and battery, mobility constraints, network constraints, variable hardware and software context) increase the complexity of the applications in embedded systems and impact the traditional application development model. New methodologies and service frameworks are required to reduce the complexity of the software and to supply a support facilitating software re-use.

To deal with the various challenges of MCC in multi context environment, we propose to adapt Service Oriented Architecture (SOA) to build a dynamic framework while alleviating several problems like portability and interoperability in MCC.
We present MCC architecture in 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.
An alternative MCC approach considers the accumulative power of swarm of mobile devices [1], 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, music Cloud platforms, smart-cities applications, and all interest-based apps can be suitable use cases. In this approach, nearby computing devices [2] can be weak devices that

1 http://www.gartner.com/newsroom/id/2905717

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 [3].
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 this paper, 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[2] 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 this context, we propose and compare two solutions: a middleware solution based on OSGi implementation and a native OS Android embedded implementation.
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 then compare the performances of an OSGi-based Cloudlet solution with two main solutions based respectively on Virtual Machines (VMs) and Docker containers.

The rest of this paper is organized as follows. Section II describes the main MCC challenges impacting the application framework design. Sections III to V, describe the different discussed models and related performance analysis. Finally, Section VI concludes the paper.

2. The main mobile Cloud challenges

To design evolving and scalable MCC application frameworks, many challenges need to be considered. These challenges can be classified into two main aspects: challenges related to mobility and communication environment and challenges related to computing aspect.

2.1 Mobile communication constraints

With the device mobility, network coverage quality can be variable. In fact, mobile-service unavailability and interruption may prolong execution time, increase monitoring overhead, and deplete smartphones' local resources, especially battery. In addition, as stated in [4], the increasing market of smartphones and IoT creates diversity and heterogeneity regarding different dimensions, such as hardware, OS, brand, capabilities, etc. This heterogeneity must be hidden for the mobile apps that have to focus on the services provided to the user.

2.2 Computing constraints

As stated in [5], to have the vision of unrestricted mobile capabilities, we augment the mobile devices with Cloud resources, even if this may impede overhead due to VM creation and migration, and outsourcing.
One of the most important aims of MCC is to conserve mobile resources and especially battery power. For this purpose, resource-intensive tasks are offloaded from mobile devices to the Cloud. However, offloading is not always synonym of energy and resources saving due to the wireless I/O required resources. Many recent works propose a decision making system before offloading, to determine whether offloading computation can save energy or not [6, 7, 8]. In addition, local execution is not always possible, either because of resources limitations (local processing and memory) or due to the absence of an equivalent service locally.

Considering the previous environment constraints, developing cross-platform components (i.e. Cloud, mobile, and hybrid) for Cloud-mobile applications become a challenging task, since mobile codes are not easily movable to various smartphones [9] while Cloud components must be portable to all Cloud infrastructures.
Regarding the number of Cloud-based applications and the several service providers, an efficient MCC framework should support a multi-tenancy architecture in addition to portability. Through multi-tenancy, a single software instance can serve multiple tenants without developing Cloud-based applications separately. Multi-tenancy is considered as the most fundamentally used technology to share computing and IT resources in Cloud computing [10, 11].
MCC includes several other challenging tasks like resource provisioning and software management without service interruption, and high service availability that needs to be achieved to guarantee service continuity.
In the next sections, we will investigate the different MCC based SOA designs that we propose.

3. Centric Cloud service oriented design

The definition of Mobile Cloud Computing is commonly related to a centric architecture.
In [12] and [13], the authors describe MCC as a service paradigm where the data processing and

2 Open Services Gateway initiative

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 [1, 14, 15, 16, 17, 18] 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 [19], "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 such as in [20, 21, 22, 23]. However, in previous works, the service notion is synonym of Web services generally based on SOAP protocol.

Hence, Longo et *al*. [23] propose to support Service-Oriented Computing (SOC) as a new suitable paradigm for Cloud computing, while Wu et *al*., in [10], 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.

Wu et *al*., in [24], 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.

Pokahr and Braubach, in [25], 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.

Unlike these cited research works, in our paper, we define a Mobile Cloud Computing – SOA (MCC-SOA) using "OSGi" framework. Compared to the work of Zhang et al. in [17, 18] who proposed a component migration mechanism from the mobile devices to the Cloud using OSGi, our approach is more complete and proposes a model that can either offload or compose remote services on the centralized Cloud. In the next paragraph, we recall the principal features of OSGi before presenting our Cloud-centric approach based on SOA/OSGi.

OSGi[3] - "Open Service Gateway initiative"- is a Java-based SOA platform that uses modular decoupled components and pluggable dynamic service models. A framework that implements the OSGi standard provides an environment for the modularization of applications into smaller bundles. A bundle is a deployment unit that can import and export packages and resources. Each bundle is a tightly coupled, dynamically loadable collection of classes, jars, and configuration files that explicitly declare their external dependencies (if any). Bundles reuse a single Java object registered and executed in a Java VM collaborative service model. And a service-management system, as depicted in Figure 1, is used to register and share services across bundles and decouple service providers from service consumers.

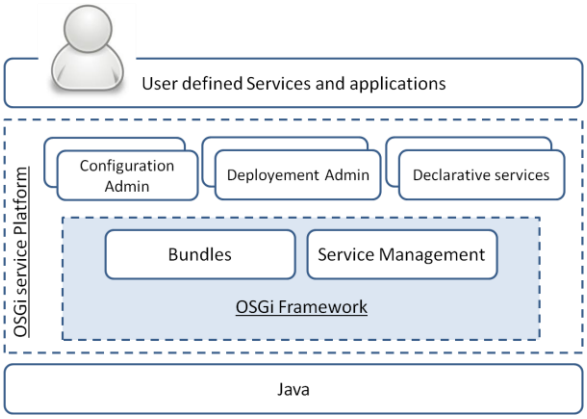

Fig 1. OSGi framework layers

OSGi bundles separate the service interface from its implementation. For remote services, such separation and low coupling is particularly important, since the host implementation is in a separate process from the service consumer. This makes it highly desirable to minimize the coupling and clear interface between distributed components[4]. As OSGi was originally designed for embedded systems, the framework provides a lightweight and scalable solution. In addition, the Java VM offers to OSGi portability and secure execution environment.

Houacine et *al*. [26] implemented a middleware solution that incorporates OSGi into Android software development platform, which builds the

3 https://en.wikipedia.org/wiki/OSGi

4 https://wiki.eclipse.org/Tutorial:_Building_your_first_OSGi_Remote_Service

initial prototype of a service-architecture for local Android-based mobile applications. This solution uses the following OSGi-based model, in which we distinguish two roles:

a- The service provider: which is a framework, installed on the Cloud server, that implements and exports services.

b- The service consumer: which is a framework running on the mobile device, used to handle interaction with mobile apps that import and consume remote services offered by the Cloud servers.

In this paper, we augment the solution of [26] proposed initially to adapt OSGi model to a mobile environment, to make a more complete contribution based on two new additional points:

i- *Service advertisement* and *discovery* components to offer means that facilitate bundle discovery in mobile Cloud context ; and

ii- *Bundle-state adaptation* to deal with network disconnections especially in hostile environments.

In the next sub-sections, are presented the different parts of the first contribution of this paper based mainly on handling remote Cloud services.

3.1 Adaptation of OSGi SOA model to mobile Cloud environment

To allow 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. R-OSGi specification is unique in being completely transport and protocol independent [27]. The provider and consumer roles are defined with respect to the use of the service rather than the transport.

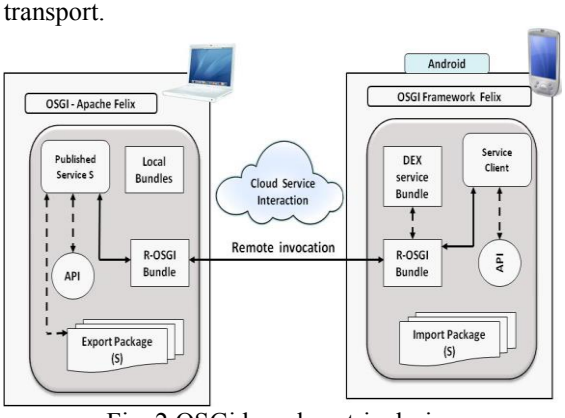

Fig. 2 OSGi based centric design

As shown in Figure 2, the interaction between the Cloud service provider and the Android mobile service consumer is performed following these main steps:

a- In the service provider side, for each remote service, a proxy bundle is generated dynamically with the exported methods.

b- 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 bytecode that has to be run on the Android platform.

c- 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.

d- Then, the application bundle of the Android platform uses the invoked remote bundle.

3.2 Service advertisement and discovery components

In OSGi, a registry is used to find required services. To address the inter-OSGi framework communication issues, we adopt an XMPP (Extensible Messaging and Presence Protocol) based solution. XMPP [28] is an open standard communication protocol for message-oriented middleware based on XML (Extensible Mark-up Language). XMPP server is used as a signalling and communication service between different OSGi frameworks. To integrate this service within Android based framework, a signalling 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 3.

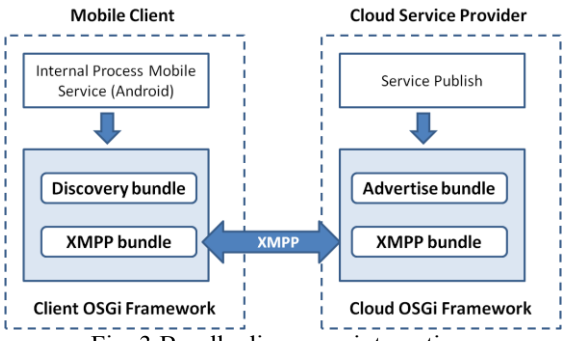

Fig. 3 Bundle discovery interaction

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 Published/Subscribe paradigm.
In the centralised 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.

3.3 Bundle state adaptation to deal with network disconnections

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 1. In this paper, we propose to handle service continuity in our OSGi-based MCC framework by investigating the behaviour of started and running services in case of disconnection with the remote bundles.

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.

Table1. Bundle states

To handle disconnection situations, we propose to complete the bundle state by adding a health-check process locally to the bundle, as in Figure 4. 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 whilst 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, its 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.

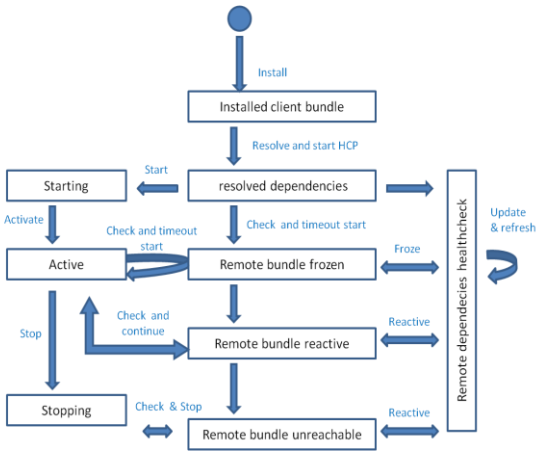

Fig. 4. The proposed bundle life cycle

3.4 Performance analysis

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.

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 within a VM of the Cloud server based on Xenserver.
- The other OSGi framework 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 5 shows the measured execution time of the bundles when varying their size according to the three execution scenarios (local, remote execution, service migration).

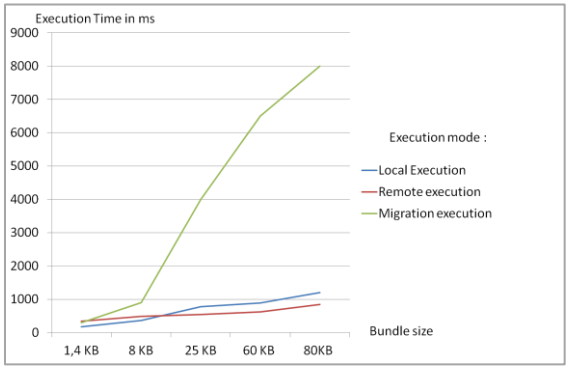

Fig. 5. Execution-time (in ms) evolution with bundle size

According to the results of Figure 5, 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 downloading 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.

In terms of memory consumption, Figure 6 shows how the mobile evolves when the bundle size varies.
Obviously, we notice that a local execution is more memory consuming. 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.

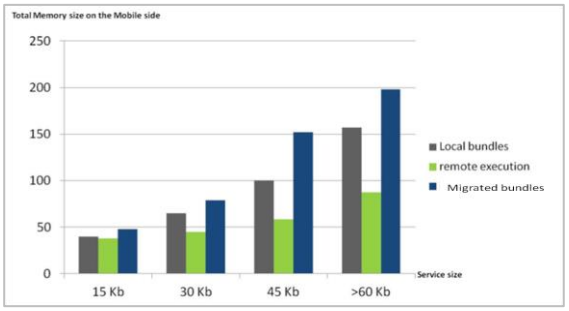

Fig. 6 Memory consumption evolution with bundle size

4. Distributed M2M Cloud services

In this model, 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, neighbouring nodes information, environmental information (e.g., CO2 and pollution levels, etc.), personal information (e.g., medical and health information using bio sensors), etc.
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 [29], 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 define two alternative solutions to the centric approach of Section 3. We first illustrate the use of OSGi to build a suitable Mobile-to-Mobile service-oriented architecture. And then, we propose a second solution that is designed as a lightweight platform by incorporating the SOA paradigm directly in the mobile Android operating system.

4.1 OSGi-based proposed solution

In the OSGi-based approach (see Figure 7), 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.

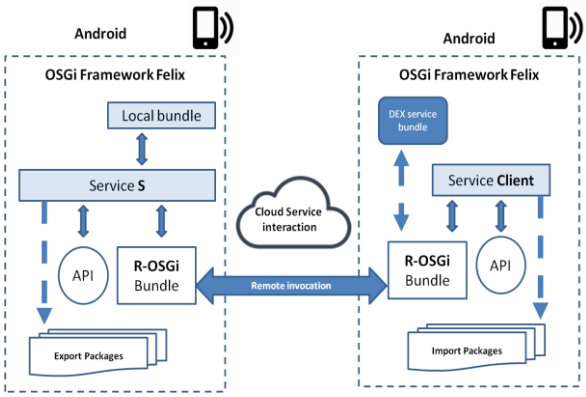

Fig. 7. Mobile-to-Mobile OSGi interaction

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[5] communication protocol, the framework provides to clients an AIDL [6] 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.

4.2 Android based service architecture

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 this 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 [30] to incorporate 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. The Android platform is organized around various layers. An application layer supplies standard apps such as an SMS application, a browser, a calendar, etc. Every application runs on a distinct Dalvik Virtual Machine to avoid altering the functioning of the other applications.

In our proposed solution, we introduce the notion of "Andromodule" which is defined as a dynamic Android module composed of:

- A service interface that provides a set of APIs
- A service body that is an Android Dalvik code
- And a manifest file which is an XML file that describes the API and the services offered by the Andromodules for dependences-resolution purposes.

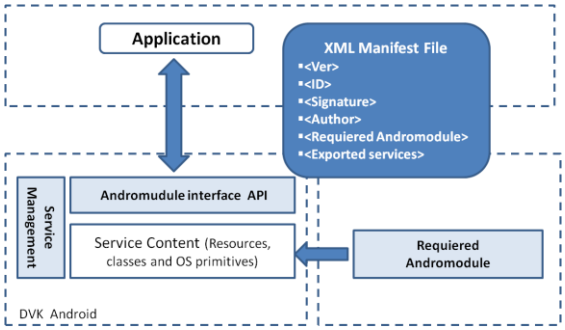

Fig.8 Andromodule components

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 Felix middleware to convert byte code to dex code.
The service management system, implemented as an Android class, includes many methods used to manage the different states of the module (see Table 2). A global overview of the Android-based architecture is given in Figure 8.

5 Remote Procedure Call

6 Android Interface Definition Language

Method	Function
InstallModule (location)	Installs a module from a specified location and assigns it an ID automatically
Start(ID)	verifies the existence of a module corresponding to the given ID, tries to resolve this module and starts it if it succeeds to resolve its dependencies
Activate (ID)	It verifies the existence of a module corresponding to the given ID and registers the service in the registry.
Stop (ID)	It verifies 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)	It verifies the existence of a module corresponding to the given ID and uninstalls it if it is resolved or stopped.

Table 2. The methods of Andromodules

In the following sub-section, we investigate the two architecture designs by comparing their performances.

4.3 Performance analysis

In this sub-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.

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.

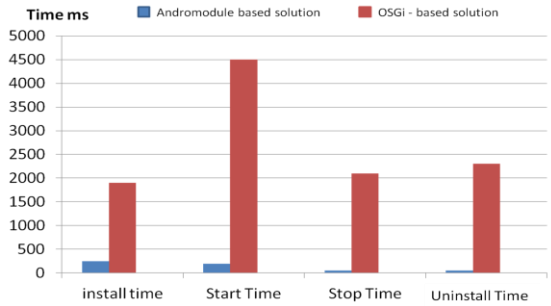

Fig.9 bundle execution time (in ms)

Figure 9 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.

5. 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 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* [3] 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 10. As stated in [3], 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.

We propose to discuss in the following sub-sections three solutions 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 OSGi framework. Finally, some measurements have been done to compare the performances between these solutions.

5.1 VM based Elijah solution

Despite their introduction in the 1970's [31], 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 [32], present a reference architecture based on Cloudlets as part of a project called Elijah project (see Figure 11). Based on a VM manager, the Cloudlets are viewed as code-offloading elements used to serve mobile devices in single-hop proximity.

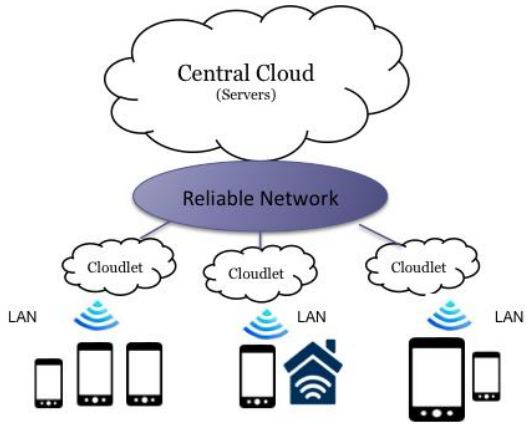

Fig. 10. Cloudlet general design

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. 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 [33] 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.

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 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.

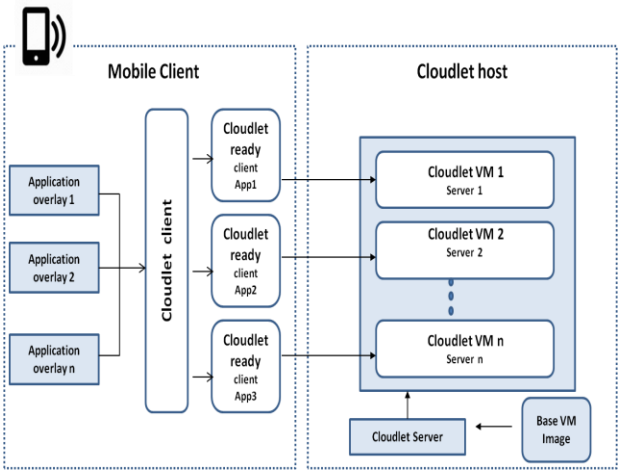

Fig.11 Elijah based Overlay architecture

5.2 Container-based Cloudlet Solution

In the virtualization-based approach (see Figure 12), 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 [34, 35], 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. Hence, the scheduler is used only at process level to schedule processes. The behaviour 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, an 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.

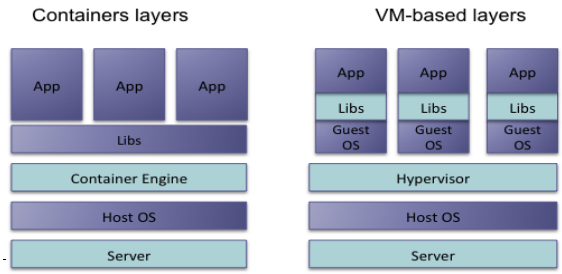

Fig.12 Containers vs VM layers

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.

5.3 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 13.

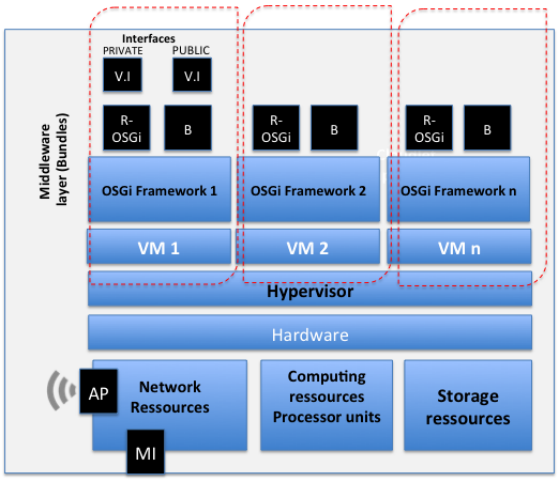

Fig.13 OSGi-based Cloudlet architecture

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.

5.4 Performances Analysis

In this section, we performed some tests related to the Cloudlet basic scheme. We measured the execution time generated by the offloading process, in the three discussed implementations: Overlay VM based solution, Docker-containers solution and OSGi middleware solution.

5.4.1 VM based Elijah solution

For this experiment, we have used two personal computers:

- One with a Linux Ubuntu 14.04 LTS <64 bits> (CORE i7) as the Cloudlet server
- And a second with Windows 7 <32 bits> (DUAL CORE) as the 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.

5.4.2 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 14.

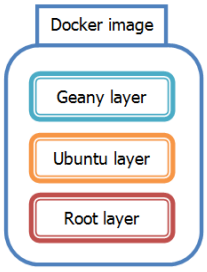

Fig. 14. Docker image

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.

5.4.3 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.

5.4.4 Discussion

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 1,4KB to be run on a Docker image, as presented in Table 3.

The overlay computed for Elijah program reaches 1,4 MB while the Docker image containing the corresponding program has a size of 6,6KB.

Table 3 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 3. 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.

6. Conclusion

We proposed, in this paper, a design and a performance analysis of 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. From the performance point of view, the first measurements highlight that OSGi is a lightweight solution suitable for mobile Cloud environment regardless of the context.

In our future works, we aim to analyse and enhance the security of our proposed middleware, on two levels: The first one will deal with the software security as a service level. The second one will be related to environment security (devices, user authentication and communication security). Finally, we plan to conduct deeper measurements to analyse the impact of the security solution on the mobile Cloud performances.

References

[1]. 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.

[2]. M. Satyanarayanan, "Pervasive computing: vision and challenges," Personal Communications, IEEE, vol. 8, no. 4, pp. 10–17, 2001.

[3]. 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

[4]. Z. Sanaei, S. Abolfazli, A.Gani, & R. Buyya, "Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges", IEEE Communications Surveys & Tutorials, pp. 369 - 392, Vol. 16, no 1, Feb. 2014.

[5]. M. Shiraz, A. Gani, R. Hafeez, & R. Buyya, "A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing," IEEE Commun. Surveys & Tutorials, pp. 1294 – 1313, Vol. 15, no 3, July 2013.

[6]. Huber Flores, S. Narayana & R. Buyya, "Computational Offloading or Data Binding? Bridging the Cloud Infrastructure to the Proximity of the Mobile User". 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, pp. 10-18, Oxford, April 2014.

[7]. P. Miettinen & K. Nurminen, "Energy efficiency of mobile clients in cloud computing". Nokia Research Center. HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (available online), 2010.

[8]. Weishan Zhang, Shouchao Tan & Klaus Marius Hansen "A Short Survey on Decision Making for Task Migrations in Mobile Cloud Environments", 2014 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI), pp. 64-67, 2014.

[9]. Zohreh Sanaei, Saeid Abolfazli, Abdullah Gani, & Rajkumar Buyya, "Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges", IEEE Communications Surveys & Tutorials, pp. 369 – 392, Vol.16, n° 1, May 2013.

[10]. D. Wu et al. "Cloud-based design and manufacturing: A new paradigm in digital manufacturing and design innovation", Computer-Aided Design, pp. 1–14, Vol. 59, Feb. 2015.

[11]. Numecent 2014. Available from http://gfxspeak.com/2013/07/19/numecent-launches-native-as-a-service-cloudpaging-platform/

[12]. White Paper, "Mobile Cloud Computing Solution Brief," AEPONA, November 2010.

[13]. Saeid Abolfazli, Zohreh Sanaei & Abdullah Gani, "Mobile Cloud Computing: A Review on Smartphone Augmentation Approaches", 1st International Conference on Computing, Information Systems, and Communications (CISCO'12), May 2012, Singapore.

[14]. Wei-Tek Tsai, "Service-Oriented Cloud Computing Architecture", The Seventh International Conference on Information Technology: New Generations (ITNG), April 2010, pp. 684 – 689.

[15]. David S. Linthicum, "Cl trustworthy oud Computing and SOA Convergence in your Enterprise: A Step-by-Step

Guide", Addison Wesley editor, sept. 2009, 264 pages, ISBN-13: 978-0136009221.
[16]. Lizhe Wang, Gregor von Laszewski & Andrew Younge, "Cloud Computing: a Perspective Study", New Generation Computing, Vol. 28, Issue 2, pp 137-146, April 2010.
[17]. Weishan Zhang et al. "Towards an OSGi Based Pervasive Cloud Infrastructure", The 2013 IEEE and Internet of Things, pp. 418 – 425, Aug. 2013.
[18]. Weishan Zhang, Licheng Chen, Xin Liu, Qinguua Lu, Peiying Zhang & Su Yang, "An OSGi-based flexible and adaptive pervasive cloud infrastructure", Science China Information Sciences, Vol. 57, no 3, pp. 1-11, Science China Press, June 2014.
[19]. Cavalcanti, José Carlos, "Effects of IT on Enterprise Architecture, Governance, and Growth", ISBN-13: 978-1466664692, IGI Global edition, Sept. 2014, 307 pages.
[20]. Johnneth Fonseca, Zair Abdelouahab, Denivaldo Lopes & 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.
[21]. Decker M., & Bulander R. "A Platform for Mobile Service Provisioning Based on SOA Integration", In Communications in Computer and Information Science, Vol 23, 2009, Springer Berlin Heidelberg, pp 72-84.
[22]. Natchetoi, Y. Kaufman, V. & 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.
[23]. Francesco Longo, Dario Bruneo, Massimo Villari, Antonio Puliafito, Eliot Salant & Yaron Wolfsthal, "Towards the future internet: the RESERVOIR, VISION Cloud, and CloudWave experiences", International Journal of High Performance Computing and Networking (IJHPCN), Vol. 8, No. 3 pp. 235-247, 2015.
[24]. 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.
[25]. Alexander Pokahr & Lars Braubach, "Towards Elastic Component-Based Cloud Applications", Proc. of the 8th International Symposium on Intelligent Distributed Computing (IDC 2014), pp. 161-171, Sept. 2014, Spain.
[26]. N. 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.
[27]. https://wiki.eclipse.org/Tutorial:_Building_your_first_OSGi_Remote_Service
[28]. https://tools.ietf.org/html/rfc6120
[29]. 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.
[30]. 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,
[31]. R. J. Creasy. The origin of the VM/370 time-sharing system. IBM Journal of Research and Development, 25(5):483–490, Sep 1981.
[32]. 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.
[33]. Xdelta.org, xdelta.org
[34]. https://www.docker.com/
[35]. 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.