MCC-OSGi: An OSGi-based Mobile Cloud Service Model

Fatiha Houacine, Samia Bouzefrane
Conservatoire National des Arts et Métiers - CNAM
Paris, France
houcin_f@auditeur.cnam.fr, samia.bouzefrane@cnam.fr

Li Li
Wuhan University
Wuhan, China
lli@whu.edu.cn

Dijiang Huang
Arizona State University
Phoenix, USA
dijiang@asu.edu

***Abstract*—In this article, a new mobile Cloud service model is presented. It offers a dynamic and efficient remote access to information services and resources for mobile devices. Mobile Cloud computing has been evolved as a distributed service model, where individual mobile users are Cloud service providers. Compared to traditional Internet-centric Cloud service models, the complexity of mobile service management in a dynamic and distributed service environment is increased dramatically. To address this challenge, we propose to establish an OSGi-based mobile Cloud service model – MCC-OSGi – that uses OSGi Bundles as the basic mobile Cloud service building components. 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 architecture models. The presented solution is validated through a demonstrative application with relevant performance measurements.**

***Keywords*— OSGi, Service Oriented Architecture, Mobile Cloud Computing, Android.**

I. INTRODUCTION

In recent developments of mobile Cloud computing [1] [2], a new service-oriented framework is presented to involve mobile devices 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. Running mobile applications on a traditional Internet-based Cloud framework, i.e., running executable binary codes in both mobile devices and Cloud, the application development approach is rigid in that the sensing capability and Cloud offloading operations cannot be easily reused and reprogrammed by various Cloud-based services. Moreover, the application development overhead of traditional approaches is higher than the service oriented approaches.

For example, a mobile device can provide various sensing services that can be used as a building block for various purposes without a pre-configuration in a traditional client-server type of application framework. The provided sensing services can be called by any services through a service management framework without even knowing for what the sensing services will be used. This feature can greatly improve the reusability of the sensing service and reduce the application developments overhead tremendously incurred by the traditional client-server approaches.

To achieve the described mobile application-running scenarios, service oriented approach is a natural choice to support mobile Cloud software development and service provisioning. In this context, Service-oriented Architecture (SOA) [5, 9] is viewed as a suitable paradigm to guarantee more control, re-use and reliability of the software. It has proved to offer flexibility, scalability and reusability, due to the independency, interoperability and loose coupling of the services. Most of existing SOA solutions have been implemented in the enterprise environment. SOA for mobile applications is still in its infant stage.

In this research, we propose a Mobile Cloud Computing – OSGi (MCC-OSGi), where each mobile device is considered as a mobile service provider in the Cloud. MCC-OSGi is established based on OSGi (Open Services Gateway initiative [4]) framework and demonstrated on the Android platform that can offload or compose remote services on the Cloud. The presented approach provides dynamic OSGi-based class loading, versioning management, and dynamic bundle reconfiguration without restarting Android applications. The main research goal of this paper 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. The main contributions of this research are:

- The establishment of an 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 performance evaluation shows that the presented solution is lightweight and suitable for mobile Cloud services.

In the following sections, the paper is organized as follows: Section 2 discusses the related work; Section 3 describes the overall MCC-OSGi architecture and the models established on Android platform. In Section 4, three possible MCC-OSGi service models are presented to deal with heterogeneous interactions between bytecode-based devices and Dalvik code devices. We underline distinct service architectures that are suitable respectively for sensing mobiles devices, Cloud offloading, or device-to-device communication. To demonstrate the feasibility of the OSGi based Cloud architecture, Section 5 presents an application example based on OSGi and the corresponding performance measurements. Finally, Section 6 concludes this paper.

II. Related Work

Mobile Cloud Computing [1][3][6][8] has become a novel computing model that integrates Cloud computing, wireless communication infrastructures, mobile devices, etc. In a mobile Cloud computing environment, on one hand, mobile applications and services on mobile devices have been augmented by accessing to unlimited computing power and storage space; while on the other hand, lightweight computing devices extend Cloud services into the mobile sensing domain. Previous work has been done on either side, but does not consider both sides in a comprehensive approach.

Chun et al. [6] proposed an architecture that can distribute components of an application automatically and dynamically between a mobile and the Cloud, where cloned replica of the smartphone's software is running. Instantiating devices' replica in the Cloud is determined based on the cost policies trying to optimize execution time.

Zhang et al. [7] developed a reference framework for partitioning a single application into elastic components with dynamic configuration of execution. The components are called weblets that can be launched on the mobile or in the Cloud, depending on the configuration of the application and the device hardware capabilities. The weblets are implemented using C# and are accessed using HTTP protocol.

Giurgui et al. [15] 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.

Huerta-Canepa and Lee [10] presented a framework to create virtual mobile Cloud computing providers. This framework mimics a traditional Cloud provider using nearby mobile devices and allows avoiding a connection to an infrastructure-based Cloud provider while maintaining the main benefits of offloading. Mobile devices in this ad-hoc computing Cloud serve as a Cloud computing provider by exposing their computing resources to other devices so that they may execute tasks collaboratively. The Hyrax platform [13] derived from Hadoop supports Cloud computing on Android smartphones. Hyrax allows client applications to conveniently utilize data and execute computing jobs on networks of smartphones and heterogeneous networks of phones and servers.

In the context of Cloud computing, the OSGi Alliance in [19] investigates the way to use OSGi for Cloud-based applications. Schmidt et al. [20] 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 [14, 16] 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 [17], 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 in Cloud systems. Research in [18] 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.

Our previous work presented in [22] implemented a middleware solution that incorporates OSGi into Android software development platform that builds the service model for Android-based mobile applications. Compared to the related works, in this article, we extend the work in [22] and provide three service architectures between mobile devices and the Cloud. Unlike the work in [15] in which experimentations are implemented on Nokia phones that have a compatible Java, Android platforms used in this paper provide a Dalvik VM that is incompatible with OSGi Java VM. Hence, one of the issues addressed in this paper is to incorporate OSGi within Android mobiles platforms.

III. Description of MCC-OSGi

In this section, we first introduce the OSGi Framework, before presenting the global view of the proposed OSGi-based architecture for mobile Cloud computing, where mobiles are Android platforms. Then, we discuss how to integrate OSGi within Android platforms and set up R-OSGi in order to establish the presented three MCC-OSGi service models.

A. OSGi Framework

OSGi is the acronym of "Open Service Gateway initiative", which is a Java development and running platform, based on modular decoupled components and pluggable dynamic service models [11]. As OSGi was originally designed for embedded systems, the framework provides a lightweight and scalable solution. Based on Java, the OSGi platform offers a portable and secure execution environment.

OSGi platform consists of an OSGi framework attached to standard services. The framework offers a service facility, where a service can be added, removed or updated at any time.

It manages the bundle dependencies and their versions using a class loader associated with each bundle. Considered as a deployment unit, a bundle is a Jar file that contains a compiled code, resources as well as meta-data. The meta-data is stored in a configuration file called *Manifest.mf*, which contains all the information related to the bundle such as the bundle name, the version, the provider, the required dependencies, the packages, etc. In particular, the Export-Package entry gives the list of the services provided by a bundle, i.e., the services that can be shared with other bundles. Similarly, the Import-Package entry lists the required bundles. The OSGi framework also provides a service management system that can be used to register and share services across the bundles and decouple service providers from service consumers.

In addition, to allow remote access, R-OSGi is a standard OSGi bundle proposed in [14] to facilitate distribution of any OSGi implementation (e.g., Equinox Eclipse, Knopflerfish, Apache Felix, Concierge, etc.). Using R-OSGi, each service provider has to register the service provided as a remote service. The service is accessed through a service caller in a transparent manner. For each remote service, a proxy bundle is generated dynamically on the client side (e.g., *proxyGenerator* [25]) that propagates the method invocation of the service to the remote platform.

The research issue to be addressed in this paper is how to establish a component-based middleware using OSGi in a mobile Cloud environment while guaranteeing SOA features such as, modularity and reusability capabilities, versioning management and dynamic bundle configuration where remote access is transparent for mobile users.

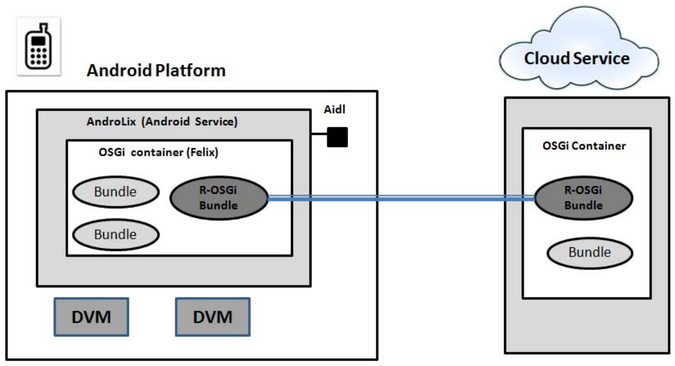

Figure 1. MCC-OSGi in MobiCloud.

To address this issue, we propose a distributed service architecture based on OSGi. We implement it in MobiCloud [1] system that is a Cloud computing that provides Virtual Machines (VMs) serving for dedicated mobile devices. Each mobile device interacts with its dedicated VM located in MobiCloud VM-pool, where the OSGi framework and their communication is supported by the signaling and communication service. To address the inter-OSGi framework signaling and communication issues, MCC-OSGi adopts R-OSGi based solution that is used as a communication service between different OSGi frameworks. We aim, through the MCC-OSGi architecture, to design a distributed service interaction while allowing a mobile device sensing for Cloud purposes, Cloud offloading for mobile devices, or device-to-device service communication. To meet our objective, we incorporate OSGi within Android application platform that supports bundle offloading and mobile sensing service composition.

B. An OSGi-based SOA for Android Platforms

The presented solutions use a component-based software design concept to implement the service-component model with SOA features such as dynamic class loading, version management, and dynamic bundle configuration avoiding the Android platform to restart when changing service compositions. In fact, we have developed a middleware to launch Felix [23] (a compact implementation of OSGi on Android devices) and to manage the life cycle of bundles that are used by Android applications. Unlike EZdroid project [21], our solution does not require the recompilation of the application during bundle updates and provides administrative functions using the OSGi platform. Compared to the solution provided by Wu et al. in [12], where an OSGi-based architecture for a smart-home environment is established through common web services, our solution is purely based on OSGi bundles.

As shown in Figure 1, the access to Felix is based on our developed middleware called AndroLix [22] that provides inter-OSGi communication through the Android Interface Definition Language (AIDL) [24] that provides descriptions containing different methods that are implemented by this set of OSGi bundles. The proposed AndroLix middleware allows the management of bundles' life cycle, and calls bundles through the AndroLix AIDL, as shown in Figure 2. The AndroLix methods that have been implemented are presented as follows:

- **install():** installs a bundle from a specified location and returns a unique bundle ID.
- **uninstall ():** uninstalls a bundle with a specific ID. If the Bundle is in a ***Resolved*** or ***Installed*** state, it is directly uninstalled. If it is in ***Active*** state, it is stopped before being uninstalled.
- **getBundleId():** returns the bundle ID.
- **startBundle():** starts a bundle after checking the availability of the bundle ID and the bundle status. The bundle is started if it is in ***Installed*** or ***Resolved*** state (i.e., the bundle's code dependencies are resolved).
- **stopBundle():** stops a bundle after checking the availability of the bundle ID and the bundle status. The bundle is stopped if it is in ***Active*** state.
- **getBundlesContainer():** retrieves the symbolic name, the bundle ID and the status (*Installed, Resolved, Active*) of all the bundles present in the bundle container.
- **startFelixFramework():** defines a persistent set of properties (Felix Framework cache directory, Felix Framework and Android packages to be loaded, etc.) and

starts the Felix Framework. This method is called when creating ***AndroLix Service***.

- **install_uninstall():** checks a package dependency of a bundle at a start-up or a stop of the bundle.
- **Call**() : allows a dynamic class-loading of classes to deal with different services associated with distinct contracts.

By calling the methods of the AndroLix middleware implemented as an Android service, any Android application is able to interact with Felix platform and its bundles.

AndoLix allows Android applications to take advantage of the facilities offered by OSGi such as dynamic bundles update and version management, but it focuses only on local running context. In a distributed and highly dynamic environment such as MobiCloud, the service providers and the consumers can be physically separated and the services can be remotely invoked. This introduces additional challenges as those related to the remote interactions and network limitations due to the movements of mobile devices. MobiCloud addresses this issue by providing on demand resources and services in a scalable way, in which OSGi bundles in the Cloud VM pool can be instantly deployed and removed within a large set of distributed nodes.

In the following sub-section, we will describe how to extend the use of OSGi bundles from the local execution environment to a distributed MobiCloud environment so that the mobile Cloud architecture is based entirely on OSGi service model.

C. A Remote OSGi-based SOA for Mobile Cloud services

To allow interactions between Android platform bundles and the Cloud services, we set up R-OSGi on the Cloud side and we integrated within Android platforms R-OSGi as a proxy bundle service. 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 method:

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

As shown in Figure 2, 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.

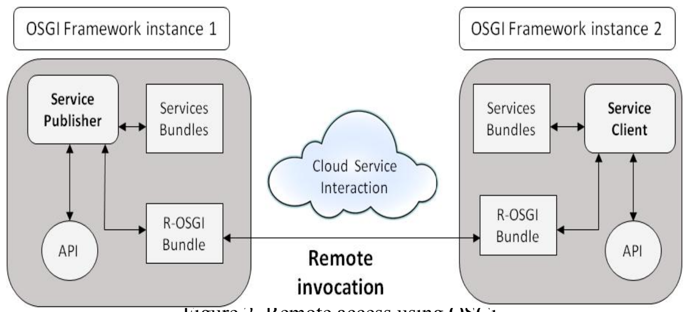

Figure 2. Remote access using OSGi.

IV. MCC-OSGI ARCHITECTURE MODELS

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

A. The Android platform as a Service Provider and the Cloud as a Client.

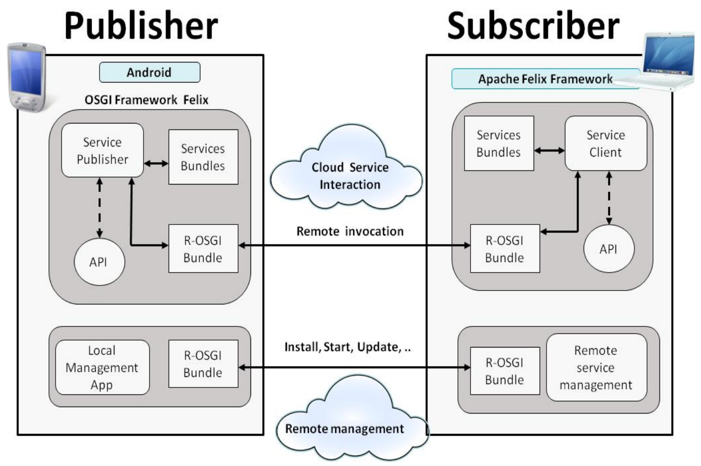

Figure 3. Android as a SP and the Cloud as a Client.

As illustrated in Figure 3, 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.

B. The Android platform as a Client and the Cloud as a Service Provider

As shown in Figure 4, 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:

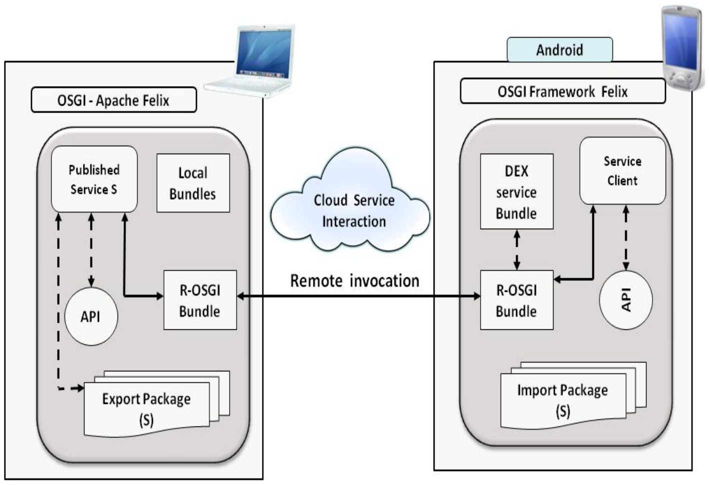

Figure 4. Android as a Client and the Cloud as a Service Provider.

a) *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.

b) *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-OSGi (see Figure 4).

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; }
}
```

c) *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 has been facilitated thanks to the development project of [25]. After all 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.

C. Android platforms as both Service Provider and Client.

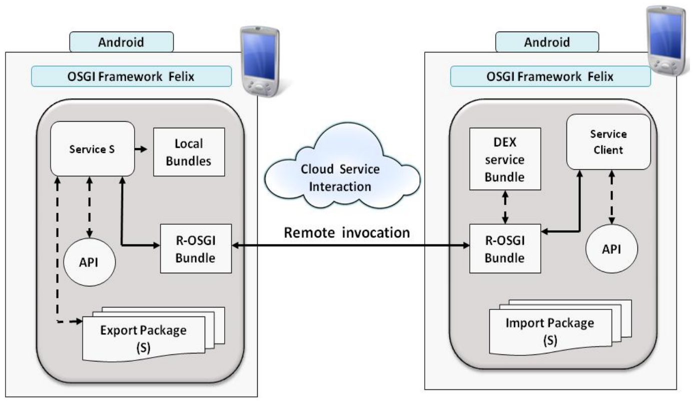

Figure 5. Each entity is an SP and a Client.

As shown in Figure 5, 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.

V. PERFORMANCE EVALUATION

In order to test the presented OSGi-based service models 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 [1], and we conducted performance measurements, as described in the following sections.

A. MCC-OSGi Service example

To illustrate the use of MCC-OSGi remote services hosted on a Cloud, we developed a service example that runs on Near Field Communication (NFC) cell phones running Android Nexus S with respect to the first service model presented in Section 4. The client side service that reads a word from an RFID tag must find this word in a dictionary service through a call to a local bundle. If this dictionary does not exist locally on the Android cell phone, a remote OSGi service in the Cloud is called to perform the treatment, allowing an outsourcing computation.

MobiCloud system provides VMs serving for dedicated mobile phone users. Each mobile phone with integrated OSGi 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 6.

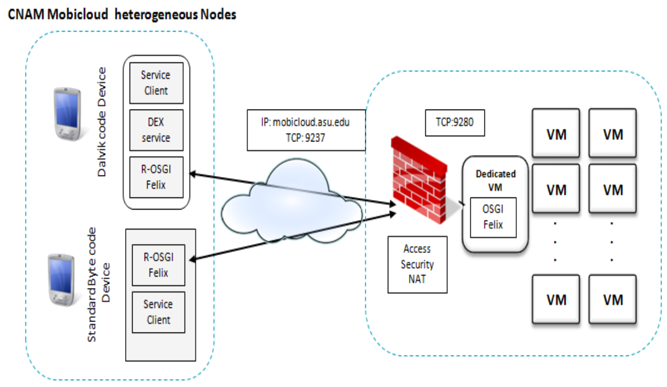

Figure 6. MobiCloud configuration.

As depicted in Figure 7, the mobile side service begins reading the information from the RFID tag (Sector 1, bloc 0), then retrieves the word, displays it on the phone screen and tries to establish a connection with the local OSGi container service to access to the dictionary. The local container itself transfers the call to the remote container that processes the information on the Cloud and returns a result to the mobile phone.

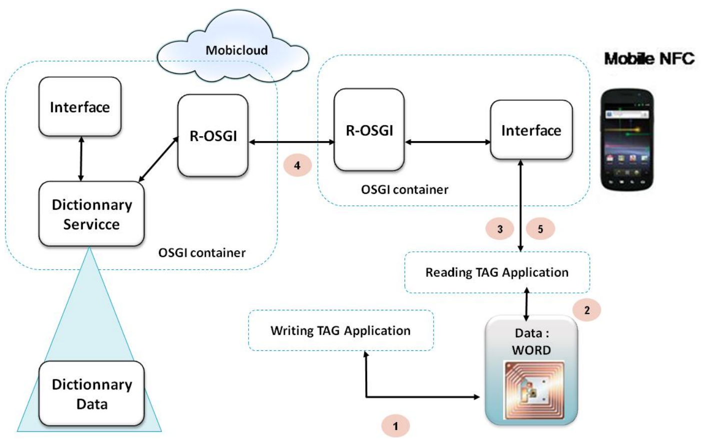

Figure 7. The application example.

The MCC-OSGi service example that is described here is used as a basis for the evaluations described in the following sections.

B. Performance Analysis

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 section we evaluate the performances of the remote service invocation regarding our new architecture models. To highlight the benefit of our proposed solution in terms of performance, we developed a test case using the previously described service example and run a set of experiments on it.

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.

a) Case 1: 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.

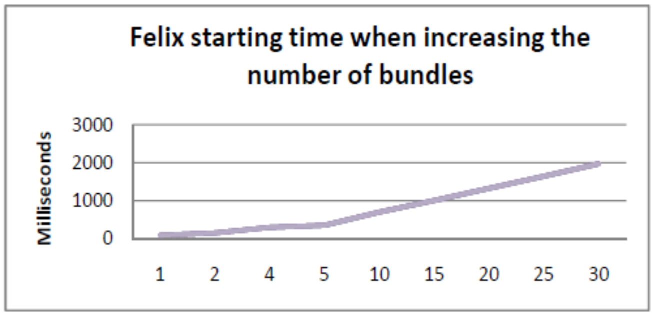

Figure 8. Felix starting time when increasing the number of bundles.

First, we start Felix on the mobile device and we measure the starting time when increasing the number of bundles as depicted in Figure 8. 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.

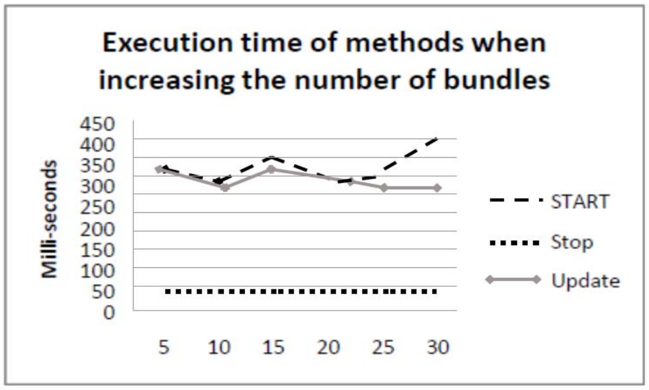

Figure 9. Execution times with bundles variation.

We also measure the execution time of three methods: *Start*(), *Stop*() and *Update*() to respectively start, stop and update bundles as in Figure 9. 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 10). 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.

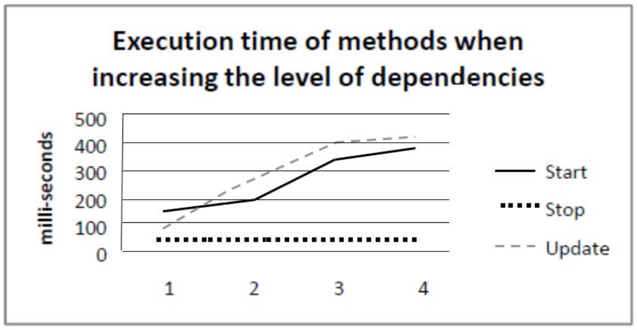

Figure 10. Execution times with dependencies variation.

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.

b) Case 2: 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).

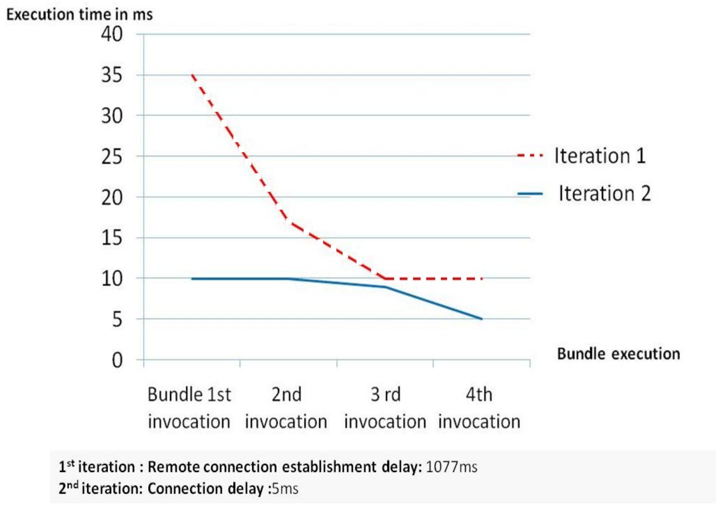

Figure 11. Remote bundle invocation.

We observed in this part, the connection establishment and the remote bundle invocation delays. Figure 11 shows a significant delay of connection, which is about 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 11, where the invocation time is significantly reduced, given the parameters already cached.

Additionally, Table 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.

TABLE I. REMOTE BUNDLE INVOCATION.

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)

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 12 shows the resource consumption in terms of memory consumed on the Galaxy tablet, with local bundles and remote ones accessed through WiFi.

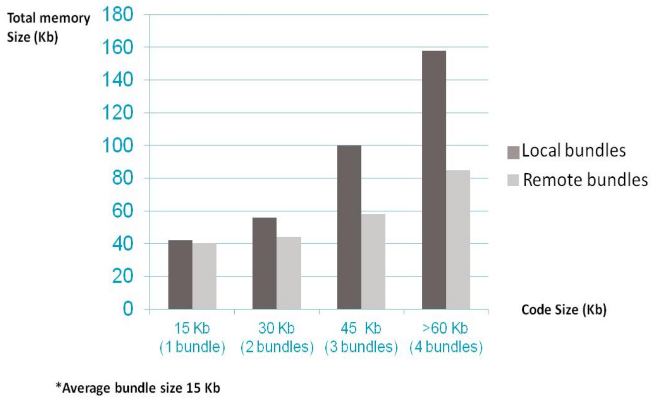

Figure 12. Memory consuming.

VI. CONCLUSION

In this article, 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.

We illustrate the use of remote OSGi in Cloud context within Android platform. Then we explain how OSGi can considerably improve the Cloud services by optimizing the remote services invocation. We have shown the performance evaluations of our approach through an application example used as a proof-of-concept in heterogeneous support environment. For future work, we will investigate the security issues in three main axes:

- Securing the mobile OSGi clients: by controlling the remote service invocations in a distributed Cloud context. Research issues such as service authentication, authorization, validation, and revocation will be studied.

- Securing the Cloud providers: from malicious service invokers, by performing common policies negotiation mechanisms between the Provider and the Client of the remote services, (e.g., encryption parameters, MobiCloud service authentication, etc.).
- We have only mentioned the signaling and communication service based on XMPP service briefly in this article. We will follow SOA features for MobiCloud such as service publishing, discovery, dynamic composition, and revocation in future work.

We also aim to extend the OSGi based MobiCloud to multiple distributed Cloud service sites to perform scalable and performing evaluation of our distributed service interaction mechanisms.

ACKNOWLEDGMENT

The research of Dijiang Huang of this research is sponsored by ONR YIP award and NSF grant DUE-0942453, CNS-1029546, and HP Innovation Research Program (IRP).

REFERENCES

[1] 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.

[2] 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.

[3] M. Satyanarayanan, P. Bahl, R. C´aceres, and N. Davies, The Case for VM-Based Cloudlets in Mobile Computing, IEEE Pervasive Computing, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[4] OSGi: http://www.OSGi.org/Specifications/HomePage

[5] D. Thanh, I. Jorstad, A Service-Oriented Architecture Framework for Mobile Services, In Proceedings of the Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop, pp. 65 – 70, 2005.

[6] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. CloneCloud: Elastic execution between mobile device and Cloud. In Proceedings of the sixth conference on Computer systems, pp. 301–314. ACM, 2011.

[7] X. Zhang, S. Jeong, A. Kunjithapatham, and Simon Gibbs, Towards an Elastic Application Model for Augmenting Computing Capabilities of Mobile Platforms, In Proceedings of The Third International ICST Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications, Chicago, IL, USA, pp. 161-174, 2010.

[8] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, MAUI: Making Smartphones Last Longer with Code Offload, in Proceedings of the 8th international conference on Mobile systems, applications, and services (ACM MobiSys '10). San Francisco, CA, USA: ACM, pp. 49–62, 2010.

[9] Y. Natchetoi, V. Kaufman, and A. Shapiro, Service-Oriented Architecture for Mobile Applications, In Proceedings of the 1st international workshop on Software architectures and mobility (SAM '08), pp. 27-32, 2008.

[10] G. Huerta-Canepa and D. Lee, A Virtual Cloud Computing Provider for Mobile Devices, In Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services Social Networks and Beyond (MCS '10). San Francisco, CA, USA: ACM, pp. 1–5, 2010.

[11] OSGi in Depth, Alexandre de Castro Alves, Manning Publications Editor, Dec. 2011, 325 pages, ISBN 193518217X, 9781935182177.

[12] C. Wu, C. Liao, and L. Fu, Service-oriented smart-home architecture based on OSGi and mobile-agent technology, IEEE Transactions on Systems, Man, and Cybernetics, Part C, Applications and Reviews, vol. 37, no. 2, pp. 193–205, Mar. 2007.

[13] E. E. Marinelli, Hyrax: Cloud Computing on Mobile Devices using MapReduce, Master Thesis, Carnegie Mellon University, 2009.

[14] 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.

[15] 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.

[16] J. S. Rellermeyer, G. Alonso, T. Roscoe: Building, Deploying, and Monitoring Distributed Applications with Eclipse and R-OSGi. In: Fifth Eclipse Technology eXchange (ETX) Workshop (in conjunction with OOPSLA 2007), Montreal, Canada, pp. 50-54, Octerber, 2007.

[17] 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.

[18] 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.

[19] http://www.OSGi.org/wiki/uploads/Design/rfp-0133-Cloud_Computing.pdf

[20] 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.

[21] EZdroit project : http://www.ezdroid.com/.

[22] 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.

[23] http://felix.apache.org/site/index.html

[24] http://developer.android.com/guide/components/aidl.html

[25] Temat project http://OSGi-at-android.wikidot.com/start