µCloud: Towards a New Paradigm of Rich Mobile Applications

Verdi March, Yan Gu, Erwin Leonardi, George Goh, Markus Kirchberg, Bu Sung Lee

HP Laboratories
HPL-2011-55R1

Keyword(s):

mobile cloud; component; composition

Abstract:

Rich mobile applications are characterized by rich functionality, offline usability and portability. However, it is not trivial to simultaneously satisfy all the three criteria. Existing approaches such stand-alone applications and the thin-client architecture satisfy only a subset of these criteria. In this paper, we show that rich mobile applications can be achieved through the convergence of mobile and cloud computing. We address two main issues in cloud-enabled mobile applications, namely complexity of application development and offline usability. We then propose µCloud framework which models a rich mobile application as a graph of components distributed onto mobile devices and the cloud. Lastly, we discuss µCloud's major research issues, i.e., workflow language for interactive applications, offline usability, secure and scalable multi-tenancy, portability and energy optimization.

External Posting Date: June 21, 2011 [Fulltext] Approved for External Publication
Internal Posting Date: June 21, 2011 [Fulltext]

To be published in the 8th International Conference on Mobile Web Information Systems (MobiWIS).

Accepted in the 8th International Conference on Mobile Web Information Systems (MobiWIS)

μCloud: Towards a New Paradigm of Rich Mobile Applications

Verdi March[a,*], Yan Gu[a], Erwin Leonardi[a], George Goh[a], Markus Kirchberg[a], Bu Sung Lee[a,*]

[a]*Services Platform Lab, Hewlett-Packard Laboratories Singapore*
1 Fusionopolis Way, 14th Floor (South Tower), Singapore 138632
Emails: verdi.march@hp.com, francis.lee@hp.com

Abstract

Rich mobile applications are characterized by rich functionality, offline usability and portability. However, it is not trivial to simultaneously satisfy all the three criteria. Existing approaches such stand-alone applications and the thin-client architecture satisfy only a subset of these criteria. In this paper, we show that rich mobile applications can be achieved through the convergence of mobile and cloud computing. We address two main issues in cloud-enabled mobile applications, namely complexity of application development and offline usability. We then propose μCloud framework which models a rich mobile application as a graph of components distributed onto mobile devices and the cloud. Lastly, we discuss μCloud's major research issues, i.e., workflow language for interactive applications, offline usability, secure and scalable multi-tenancy, portability and energy optimization.

Keywords: mobile cloud; component; composition

1. Introduction

Mobile computing [1] has been undergoing a rapid shift of focus from hardware to applications. This transition is driven mainly by the critical mass adoption of mobile devices to the point that mobile devices will become as ubiquitous as desktops/laptops or even to surpass them. A recent study from Gartner indicates that a total of 297 million smartphones were sold in 2010 [2]. Furthermore, the study shows that the ratio between new smartphones and desktop/laptops increased from 0.6:1 in year 2009 to 0.85:1 in 2010, whereas data from Advertising Age[1] shows that the penetration of mobile devices is 37–59% higher than desktops/laptops in Brazil, China, India and Indonesia.

Ideally, mobile applications satisfy three desirable properties. First, in spite of the inherent limitations of the underlying hardware, mobile applications should still provide rich and non-trivial functionalities that are similar to PC applications. Such limitations are well documented, which are mainly the computational resources (i.e., CPU, memory and storage) and battery life [1]. Second, the application should be reliable under various network conditions. Not only it must stay responsive under severe network degradation, but also usable when the network is disconnected (i.e., offline usability). Third, the application should be available on various types of devices.

Stand-alone mobile applications, which are currently predominant, run locally on a device. As such, they are generally usable regardless of network connectivity. However, the applications are relatively limited in functionality

*Corresponding authors. *E-mail addresses*: verdi.march@hp.com, francis.lee@hp.com

[1] `http://adage.com/article/global-news/10-trends-shaping-global-media-consumption/147470`

due to limited resources and energy supply available on mobile devices. In addition, application portability is further complicated by the fragmented mobile device market, e.g., the presence of various versions of mobile operating systems from Apple, Google, Microsoft, Nokia, etc.

The gap between current mobile applications and rich mobile applications can be bridged through the convergence of the mobile and the cloud. We expect a new generation of mobile applications that provides much richer functionalities than standalone mobile applications. This will come as a result of offloading complex, non-trivial tasks to the cloud, thus breaking through the constraints of limited resources and energy. Furthermore, applications can be delivered faster because the cloud centralizes software implementations and promotes legacy software reuse.

However, in realizing cloud-enabled mobile applications, a number of significant issues arises. First of all, to reduce the complexity of developing a distributed application, the integration between cloud and mobile devices should be seamless and transparent. Furthermore, network conditions such as jitters, congestion and connectivity impose another challenge in ensuring the reliability of cloud-enabled mobile applications. Finally, to optimize the energy consumption on mobile devices, the overhead of on-device computations and network communication need to be considered.

To address the above mentioned challenges, we propose μCloud, a framework for rich mobile applications with five design objectives.

1. *Modular Composition* — Software composition empowers layman users to create rich mobile applications by mashing up modular components. Achieving such a non-programmatic application development requires a visual composition language that can cleanly and elegantly capture user-initiated interactions among components. Furthermore, the mash up can be done on the cloud so that applications can be develop without a local software development environment.
2. *Portability* — Tasks that do not require device-specific capabilities can be implemented on the cloud, potentially by reusing existing software.
3. *Separation of Concern* — To benefit from good software engineering practices, the framework inherently distributes responsibilities to various actors, each with their independent development cycle. Layman users assemble applications from components, or even customize applications to suit their specific need. Skilled programmers design and implement components. Lastly, cloud providers operate and maintain the mobile cloud platform.
4. *Overcome Resource and Energy Constraints* — As resource-intensive tasks are offloaded to the cloud, mobile devices can perform non-trivial information processing with minimum reduction in battery life.
5. *Offline Usability* — Not only applications must be reliable when the network is severely degraded, but also be usable when the device is disconnected from the network.

Existing approaches such as CloneCloud [3] and MAUI [4] focus on reducing energy consumption by decomposing an application, written for mobile devices, such that parts of the application can be moved to the cloud. Reusability of functionalities across applications and loose coupling are not considered; hence, software composition is not addressed. In addition, it is not easy for layman users to customize application functionality as the application needs to be programmatically modified.

To meet all the above mentioned design objectives, we propose a mash-up approach whereby a rich mobile application is composed from components. Such a composition approach promotes modular, flexible and configurable applications, and reuse of independent software components. Formally, an application is represented as a directed graph of components. We assume that components can be clearly identified and are independently developed by skilled programmers. Each component can be mobile, cloud or hybrid; it provides a high-level functionality such as face detection or face recognition. Components are self contained, i.e., decoupled from each other such that they never communicate directly with each other. Instead, components only need to specify their input, output and configuration. Then, the underlying runtime platform routes data across components.

The main contributions of this position paper are as follows. We present our approach that meets the five design objectives (Section 3), and show a prototype of our proposed model using face recognition as the use case (Section 4). Then, we highlight the major research issues that need to be addressed (Section 5).

2. Related Work

We first discuss composition-based software development approaches [5, 6, 7], followed by energy optimization approaches for mobile applications based on software decomposition [3, 4].

Similar to our proposed μCloud, the MIT's Click modular router [5], Microsoft Dryad [6] and IBM SPL [7] enable an application to be composed by connecting independent components. Cycles are permitted in Click and SPL, but not in Dryad. However, these schemes do not directly support user-initiated control flows among components. As such, there is a lack of clean and elegant constructs to represent user-initiated interactions among components.

CloneCloud [3] is a system that automatically and seamlessly off-loads part of the execution of mobile applications from mobile devices onto the device clones in a computational cloud. Hence, it optimizes the performance and energy use of the applications. The right partition of a mobile application to be migrated to the cloud is determined by performing off-line static and dynamic profiling. However, they need source code of the application for the profiling and prediction, while we provide online black-box profiling for the component-based applications.

MAUI [4] presents a platform to minimize power consumption of mobile devices by offloading tasks to the cloud. It analyzes the code of application and potentially annotates the methods for remote executions. In runtime, the profiler monitoring the resource predicts whether the method should be offloaded to the cloud. In the end, the platform reintegrates the results back into the mobile device. In addition, our platform facilitates the feedback workflow and handles user interactions in applications.

The webOS 2.0[2] provides the Node.js[3] runtime environment which is also available on the server-side. With Node.js, developers are able to write event-based network services using Javascript that can run on both server-side and on webOS 2.0 devices.

3. μCloud Framework

In this section, we present the application and execution model of μCloud.

3.1. Application Model

We propose that each application is a composition of heterogeneous components. Formally, an application is defined as a directed graph, where nodes and edges denote components and data flow between components, respectively. In our current model, data flow implies control flow. This preliminary model does not support cycles yet, though this is being addressed in our ongoing works. Each component provides a high-level functionality such as face detection or face recognition. Components are decoupled from each other; hence, they never communicate with or reference to each other directly. Each component has a set of well-defined identifier, input/output parameters and configuration. Components are further classified by their *location*: cloud, mobile or hybrid. A *cloud component* runs exclusively on the cloud, while a *mobile component* runs exclusively on a mobile device. The third category, *hybrid component*, can run on either the cloud or mobile devices. Each hybrid component consists of either multiple implementations, or a single implementation that requires a specific middleware such as webOS.

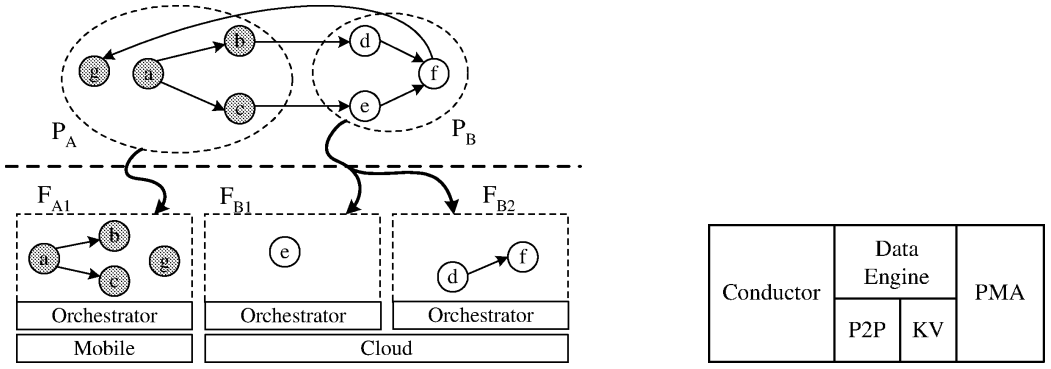
Each component has its private memory, which is isolated from all other components, and specifies zero or more input/output parameters and configuration. When an application graph is executed, the output of each component is injected to its subsequent components. We adopt the *pass-by-value* semantic for data flows between components for two reasons. First, this semantic ensures that a component can never directly refer to the internal data stored in its predecessors' memory space. Thus, the data-flow semantic will be consistent even if multiple components are mapped onto the same address space (i.e., virtual machine or process). Second, pass-by-value prevents shared data across components, which eliminates data race.

3.2. Execution Model

Figure 1a illustrates the runtime scheme of μCloud platform. An application graph is partitioned into disjointed sets or *partitions*, e.g., P_A and P_B. Each partition represents components with homogeneous resource requirements.

[2] http://www.palm.com/us/products/software/webos2

[3] http://www.nodejs.org



(a) Execution Model (b) Architecture of Orchestrator

Figure 1: μCloud Framework

As an example, the application in Figure 1a is partitioned into a partition of native mobile components (P_A) and a partition of cloud components (P_B). To execute partitions, each partition is further sub-partitioned into one or more *fragments*, e.g,. F_{A1}, F_{B1} and F_{B2}.

Each fragment is executed by an *orchestrator*. The relationship between applications and orchestrators is many-to-many. That is, an application can be fragmented onto many orchestrators. Conversely, an orchestrator can manage multiple applications. Figure 1b illustrates the architecture of an orchestrator. Each orchestrator consists of three main elements:

- *Conductor* — A conductor orchestrates the execution flow as specified in the application graph. For components directly managed by the conductor, the native interface will be used for orchestrations. To pass the execution control to a component managed by another conductor, we will fall back to web service interfaces. As an example, a mobile conductor passes the control of execution to the cloud by invoking the RESTful operations of the cloud conductor.

- *Data Engine* — The data engine facilitates and optimizes data flow from one components to another. Data may be routed *point-to-point* (P2P) or through a *key-value data store* (KV). The engine is also responsible to handle serialization. As an example, when data crosses from a mobile device to the cloud, the mobile conductor serializes primitive values into JSON format, while more complex types can be converted into strings to be interpreted and handled by the cloud conductor. Then, the mobile conductor transfers the data by invoking RESTful interface of the cloud conductor.

- *Performance Monitoring Agent (PMA)* — This element continuously monitors the performance of an application. The information serves as a feedback to trigger dynamic adaptation operations.

4. Proof-of-Concept Implementation

In this section, we describe the current μCloud prototype and a use case of face recognition.

4.1. μCloud Prototype

The current μCloud prototype consists of (i) *component and application portal*, (ii) *component SDK*, (iii) *cloud orchestrator* and (iv) *mobile orchestrator*.

The web portal for managing components and applications consists of the following modules:

- *Component Repository* is the entry point for developers to publish and maintain individual components.

- *Application Builder* is a mash-up-style user interface that enables application developers to create, publish, modify and republish applications based on existing components. This application builder generates an application graph, which is later used to compile and orchestrate the application.

- *Application Marketplace* is a place for users to browse and buy/download μCloud applications.

The component SDK provides a building block to ease the development of Java-based components. It defines component life cycle and input/output API, and guarantees that components are re-entrant. The SDK also transparently decorates components with a RESTful interface to reduce the development effort.

The cloud orchestrator is implemented in Java and supports the execution of Java components. The prototype currently executes each partition on one orchestrator only. Fragmenting partitions for execution on multiple orchestrators is part of our ongoing work. The prototype data engine currently implements P2P (i.e., point-to-point) transport. Data are passed within and across orchestrators using a native interface and a RESTful interface, respectively.

The mobile orchestrator and mobile components are based on the Android platform. Mobile components accept inputs and emits outputs via *Intents*[4]. The mobile conductor is implemented in Python using the SL4A[5] (Scripting Languages For Android) environment. The mobile conductor supports the mobile-to-cloud data flow transition using the HTTP multipart/form-data method (RFC1867) to send primitive and complex data types to the cloud conductor. For the cloud-to-mobile data flow transition, the mobile conductor retrieves a JSON-encoded data structure from the cloud conductor, and transposes the encapsulated data to use as inputs for the subsequent components.

4.2. Use Case

Face Recognition enables a user to automatically retrieve the profile of a face taken with an Android device.

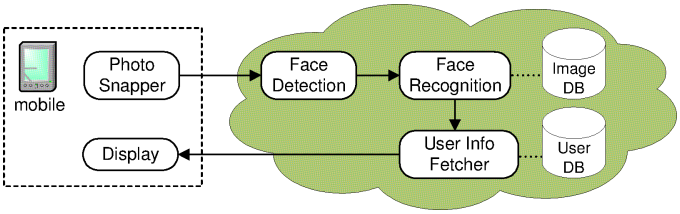


Figure 2: *Face Recognition* Application

The *Face Recognition* application is partitioned into several components, namely, the *Photo Snapper* component, the *Face Detection* component, the *Face Recognition* component, the *User Info Fetcher* component, and the *Display* component. The *Photo Snapper* and *Display* components are deployed to the mobile phone, while other components are deployed in the cloud (Figure 2).

The *Photo Snapper* snaps photos by engaging the embedded camera on the mobile phone. The *Face Detection* component utilizes the Viola-Jones method implemented in OpenCV library [8]. It detects faces from the input photo and extracts them. For each detected face, the *Face Recognition* component compares it to the image database, and returns top three of most similar faces and their corresponding profile. This component is a Java wrapper to a Python implementation[6] of the Eigenfaces face recognition algorithm [9]. Once the face is found in the image database, the corresponding user particulars such as name, designation and email information are retrieved from the user database by the *User Info Fetcher*. Note that the face images in the image database and user information are collected by the *Face Data Acquisition* application. The *Display* component displays the top three matched faces and the individual user information with the mobile phone.

5. Research Challenges

We discuss five major research issues, namely (i) *workflow language for interactive applications*, (ii) *secure and scalable multi-tenancy*, (iii) *offline usability*, (iv) *portability* and (v) *energy optimization*.

[4] `http://developer.android.com/guide/topics/intents/intents-filters.html`
[5] `http://code.google.com/p/android-scripting`
[6] `http://code.google.com/p/pyfaces`

Workflow Language for Interactive Applications — Enabling layman users to quickly and easily create μCloud applications poses a number of challenges including (i) a clean, elegant and intuitive mash-up language that supports user-initiated interactions among components, and (ii) an abstraction layer that hides domain knowledge, programming abstractions, and cloud and mobile communication aspects.

While component is an abstraction that is generally comprehensible by most users, defining a mash-up-like user interface has been the focus of research for many decades. Many of the proposed user interfaces adopt the notion of a flow as the main construct to interlink components; flows are either control flows (e.g., BPEL [10]) or data flows (e.g., Yahoo! Pipes[7]). In the business process domain, control flows are usually specified using control constructs like a control flow graph, while the data flow is specified implicitly using bindings that link two processes. Thus, the control flow and the data flow are described on two different levels. This differentiation may introduce conflicts when the two levels are put together at runtime [11]. Specifically, a process may obtain its input from any arbitrary processes that are not its predecessors. Therefore, it would be desirable to have a unified mash-up or modeling approach which integrates data and control flow specifications, yet remains intuitive to layman users.

Offline Usability — To deal with unstable network connection, mobile cloud applications should transparently support disconnected operations. At the very least, the applications should not terminate abruptly when a mobile component fails to transition to a cloud component, and vice versa. Using the face recognition application as the example, even as the network connection is lost when transitioning from the mobile device (i.e., the photo snapper component) to the cloud (i.e., the face detection component), the output of the photo snapper should be queued at the device instead of discarded. When the connection is restored, this data can then be transmitted to the face detection component running on the cloud. Similarly, should connectivity is disrupted when transitioning from the cloud to the device, the recognition result should be cached by the cloud until delivered to the device when connectivity is restored. Supporting disconnected operations requires a multi-pronged approach encompassing various aspects in distributed computing, including reliable messaging between cloud and mobile components, caching and synchronization.

In addition to disconnected operations, the distribution of large data sets needs to be optimized. Different protocols are needed for exchanging data (i) between mobile and the cloud, (ii) within a cloud and (iii) among clouds (i.e., federated cloud). The data exchange also needs to consider components that can be migrated during their execution.

Secure and Scalable Multi-Tenancy — The runtime platform supports multi-tenancy yet must be secure and scalable. The platform must minimize the risk of leakage of users' data in the presence of shared components and applications. By design, a component can be used by multiple applications. However, these applications must be isolated from each other. As such, the platform must guarantee that a component instance, bound to a particular application, has its exclusive name space.

Towards this, we are investigating two scheduling schemes: *SHM* and *DM*. The SHM (shared memory) scheme allows one orchestrator to manage instances of multiple applications via their native interfaces. This scheme reduces the runtime overhead of trusted components or components requiring the same runtime environment (e.g., Java). However, the orchestrator must guarantee the pass-by-value semantic, and that applications are isolated from each other. The DM (distributed memory) scheme allows an instance of an application to be mapped onto multiple orchestrators. This is necessary for applications composed of components with heterogeneous resource requirements.

Portability — Hybrid components run on heterogeneous environment and facilitate dynamic workload migration. An emerging approach is by building applications using web-standard technologies, such as Titanium Appcelerator[8] and PhoneGap[9]. HP has taken this a step further by building the upcoming webOS (Enyo) platform to be fully based on web standards. With the support of Node.js in webOS 2.0 onwards, we would like to investigate ways of building services using Javascript and Node.js which allow the conductors to move components seamlessly between mobile devices and the cloud. In this approach, all components are written in Javascript such that the same component can be run on a hosted environment in the cloud, or in mobile devices.

Another potential direction is mobile virtualization. With the advance of mobile hardware technologies, it may be possible in the future to run lightweight virtual machines on mobile devices with minimum overhead.

[7] `http://pipes.yahoo.com`
[8] `http://www.appcelerator.com`
[9] `http://www.phonegap.com`

Energy Analysis and Optimization — With such component-based application platform, we are seeking the dynamic deployment scheme so that the components could be hosted across the cloud and the mobile phone in order to minimize the system-level energy consumption for mobile devices. The idea is to offload expensive computational tasks from thin, mobile devices to powered, powerful devices in the cloud so that we could prolong battery life for mobile clients. Referring to our *Social Face Recognition* use case, the *Face Recognition* component can be run on the mobile so that less data is transferred between the phone and the cloud, to reduce the overhead of transmitting large images.

6. Conclusion

Rich mobile applications can be achieved through the convergence of mobile and cloud computing. However, new challenges arise, particularly the seamless integration between cloud and mobile, offline usability and energy efficiency. To address these challenges, we proposed μCloud, a framework for rich mobile applications based on software composition. We described the composition model, execution model and architectural design, followed by a social face recognition application. We also presented a number of major research issues in achieving the vision and design objectives of μCloud, which are being addressed in our ongoing work.

Acknowledgment

We thank Tong Zhang from HP Labs Palo Alto, Ke-Yan Liu and Lei Wang from HP Labs China for the face recognition application. We also thank Jerome Basa and Jesus Alconcher Domingo from HP Labs Singapore for their contribution in developing the μCloud prototype.

References

[1] M. Satyanarayanan, Fundamental challenges in mobile computing, in: Proc. of PODC, 1996.
[2] C. Pettey, L. Goasduff, Gartner says worldwide mobile device sales to end users reached 1.6 billion units in 2010; smartphone sales grew 72 percent in 2010 (Feb. 2011).
URL http://www.gartner.com/it/page.jsp?id=1543014
[3] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, CloneCloud: Elastic execution between mobile device and cloud, in: Proc. of EuroSys, 2011.
[4] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Stefan, R. Chandra, P. Bahl, MAUI: making smartphones last longer with code offload, in: Proc. of MobiSys, 2010.
[5] E. Kohler, R. Morris, B. Chen, J. Jannotti, M. F. Kaashoek, The Click modular router, ACM Transactions on Computer Systems 18 (3) (2000) 263–297.
[6] M. Isard, M. Budiu, Y. Yu, A. Birrell, D. Fetterly, Dryad: Distributed data-parallel programs from sequential building blocks, in: Proc. of EuroSys, 2007.
[7] M. Hirzel, H. Andrade, B. Gedik, V. Kumar, G. Losa, M. M. H. Nasgaard, R. Soule, K.-L. Wu, SPL stream processing language specification, Technical Report RC24897 (W0907-066), IBM Research Division (Nov. 2009).
URL http://domino.watson.ibm.com/library/cyberdig.nsf/papers/DCF1CFDB512167AD85257677005DC3FB
[8] G. R. Bradski, The OpenCV library, Dr. Dobb's Journal of Software Tools.
[9] M. A. Turk, A. P. Pentland, Face recognition using Eigenfaces, in: Proc. of CVPR, 1991.
[10] OASIS Standard Committee, Web services business process execution language version 2.0, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html (2007).
[11] A. Ankolekar, M. Paolucci, K. P. Sycara, Towards a formal verification of OWL-S process models, in: Proc. of ISWC, 2005.