

# MobiCloud - Making Clouds Reachable: A Toolkit for Easy and Efficient Development of Customized Cloud Mobile Hybrid Application

Ashwin Manjunatha, Ajith Ranabahu, Amit Sheth and Krishnaprasad Thirunarayan  
Ohio Center of Excellence in Knowledge-Enabled Computing (Kno.e.sis)  
Wright State University, Dayton, Ohio 45435  
Email: { ashwin, ajith, amit, tkprasad }@knoesis.org

## Abstract

The advancements in computing have resulted in a boom of cheap, ubiquitous, connected mobile devices, as well as seemingly unlimited, utility style, pay as you go computing resources, commonly referred to as Cloud computing. However, taking full advantage of this mobile and cloud computing landscape, especially for the data intensive domains, has been hampered by the many heterogeneities that exist in the mobile space, as well as the Cloud space.

Our research attempts to exploit the capabilities of the mobile and cloud landscape by introducing MobiCloud, an online toolkit to efficiently develop Cloud-mobile hybrid (CMH) applications. We define a CMH application as a collective application that has a Cloud based back-end and a mobile device front-end. Using a single Domain Specific Language (DSL) script, MobiCloud toolkit is capable of generating a variety of CMH applications. These applications are composed of multiple combinations of native Cloud and mobile applications. Our approach not only reduces the learning curve, but also shields the developers from the complexities of the target platforms. In this paper, we provide a brief description of the MobiCloud toolkit and the workflow.

## 1 Introduction

The primary motivation for our research comes from the promising nature of the Cloud and Mobile combination. There have been interesting changes at both ends of the spectrum of computing power. There has been a boom in mobile computing devices, as well as a substantial growth in high-end data centers that offer cheap, on-demand computing resources, popularly named, *computing Clouds*.

In the backdrop of these advances in computing and the growth of data intensive domains such as social networks, a new class of applications have emerged taking advantage of not only on-demand scalability of computing clouds but also the sophistication of current mobile computing devices.

This class of applications that we name as *cloud-mobile hybrids* (CMH), are characterized by the need for heavy computations on the back-end and mobile device based front-end. Figure 1 illustrates the structure of CMH application. Developing a CMH application; however, is significantly difficult and complicated than developing a regular application.

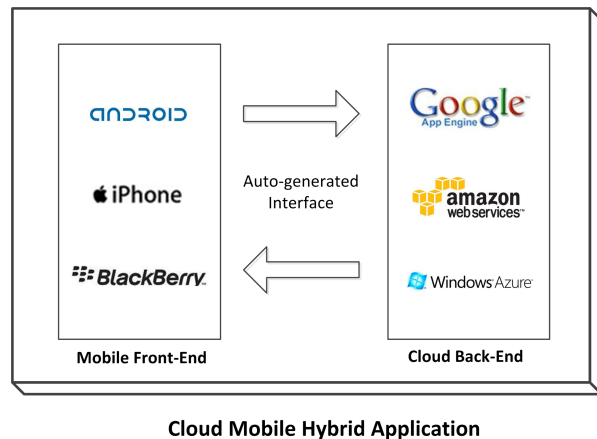


Figure 1: The structure of Cloud mobile hybrid application

Three challenges in the current CMH development process are identified as follows:

(1) The multitude of existing Clouds offer different paradigms, programming environments and persistence storage. The heterogeneity present in the core Cloud services, effectively locks the developers to a particular vendor.

(2) A number of mobile development platforms exist today. Each of them have their own Standard Development Kit (SDK), Application Programming Interfaces (API) and programming language. Fragmentation of APIs even within a single platform forces mobile application developers to focus on only specific platforms and versions [5, 2]. The current practice in the industry is to concentrate the devel-

opment efforts on selected mobile platforms, leaving out a significant portion of devices and platforms.

(3) Developing the back-end and front-end as separate components require managing the communication interfaces. The presence of Remote Procedure Calls (RPC) makes the whole development process tedious, even with an arsenal of sophisticated tools at a developer's disposal. The separation of the front-end and the back-end is also a source of version conflicts with Clients and Services where the service API has to be maintained at the level of the least capable client. Introducing changes to the service API would break the existing clients requiring frequent updates, a common problem faced by many of the mobile application vendors.

We present a solution where CMH applications are developed using a DSL in a platform agnostic fashion. Platform specific artifacts are later generated utilizing this DSL. We demonstrate our current toolkit, capable of generating code for four target platforms using a single DSL script. Treating a CMH application as a single entity and leaving out the details of generating corresponding Cloud and mobile artifacts to the toolkit not only reduces the complexity significantly but also facilitate portability. Auto generating the communication interfaces also shields the developers from the heterogeneities of each of the platforms as well as lengthy debug cycles of remote procedure calls (RPC). Further details of the MobiCloud platform is available in the MobiCloud technical report [4].

## 2 Technology

The core contribution in our work is the MobiCloud DSL and the associated toolkit. The toolkit is capable of generating code for the following four target platforms.

- Android platform (v 1.5)
- Blackberry platform (v 4.5)
- Google Appengine
- Amazon EC2

The DSL is based on the Model-View-Controller (MVC) design pattern. We have presented the philosophy and further details of the design of this DSL in the MobiCloud technical report [4].

We now take an example applications and demonstrate the capabilities of our toolkit.

### 2.1 Hello world example

**Listing 1:** The DSL script for the *hello world* application

```
recipe : helloworld do
  metadata : id => 'helloworld-app'
  model : greeting ,
          { :message => :string }
  controller : sayhello do
    action : retrieve , :greeting
```

```
end
view : show_greeting ,
      { :models => [:greeting] ,
        :controller => :sayhello ,
        :action => :retrieve }
end
```

Listing 1 depicts the DSL script for the Hello world application. This script generates the following components.

- (1) A *model* with only one attribute.
- (2) A *controller* with only one action.
- (3) A *view* demonstrating a basic user interface.

We now describe the translations in detail for the *hello world* application. Although we use this hello world application to describe the salient features of this language, it is not an exhaustive demonstration of all the possible options of this DSL. Some extended sample applications that demonstrate advanced capabilities of the DSL are available at [3].

#### Back-end

1. A Google Appengine (GAE) compatible Java Web application with data storage capabilities.
2. A *pure* Java Servlet / JSP based Web application with data storage capabilities compatible with Amazon EC2. Being *pure* signifies that only standard Java libraries are used in the generated code.

#### Front-end

1. A native Android application containing XML based auto generated user interfaces.
2. A native Blackberry application containing Java based auto generated user interfaces.

For the back-end, i.e., the Cloud portions of the application, the following artifacts are generated.

1. A set of *bean* classes for the models that optionally include the Java Data Object (JDO) standardized annotations. Each of the beans represent an implementation of the relevant model description.
2. A set of JSP based HTML views that mimic the mobile UI. The intention of this UI is to act as a testing tool.
3. The implementation of a relevant persistence manager class.
4. RESTful Web Service that manages inputs via HTTP POST requests and responds with XML.
5. Platform specific configuration files (e.g. the appengine-web.xml file required by GAE).

The front-end, i.e., the mobile portion of the application consists of the following artifacts.

1. An optional set of *bean* classes for the models depending on the platform requirements. For some platforms the beans are omitted.

2. A set of UI classes, event handlers and related artifacts depending on the platform for each of the views.
3. An *Index* view that acts as the default entry point to the application.
4. Web service access classes that use platform specific communication libraries.
5. XML parsing code that use platform specific XML processing libraries. The generated code is based on the specific XML format followed by the back-end serializer.
6. Platform specific configuration and descriptor files (e.g. the *AndroidManifest.xml* file required by the Android platform).

The generators also output an Ant<sup>1</sup> file capable of building deployable artifacts. To execute this Ant build, the relevant SDK may need to be configured in the developers computer.

In this specific example, the hello world GAE application consists of the following important artifacts.

- (1) *Greeting.java*: The bean corresponding to the model definition of *Greeting*. *Greeting* consists of a single string attribute that represents the message.
- (2) *Sayhello.java*: An HTTP Servlet [1] corresponding to the controller definition. This servlet handles the *:retrieve* action by generating an XML output of the list of *Greeting* items in the data store. The XML output is generated by a JSP.

Similarly, in the mobile side, the hello world Android application consists of the following major artifacts.

- (1) *Index.java*: An Android activity that acts as the entry point to the application. Provides access to all the other functions, in this case, to view the greeting message.
- (2) Corresponding XML files that form the UI.
- (3) *AndroidManifest.xml*: Android specific configuration file, considered mandatory to deploy an Android application.

### 3 System Implementation

The system is implemented in Ruby in order to take advantage of the existing Ruby parser and interpreter. Figure 5 illustrates major components of the system. The parser is a top down parser that takes the DSL scripts (a.k.a. *recipes*) and converts them into in-memory object representations. These object models are then converted into platform specific code using the corresponding generator and the associated templates. To support a new platform, the system requires only an additional generator targeted towards the new platform.

The generator tool, complete set of sample programs and XML version of all results, is available on the Kno.e.sis

<sup>1</sup><http://ant.apache.org>



Figure 2: Step 1 : The online editor for the DSL with syntax highlighting and other text editor support

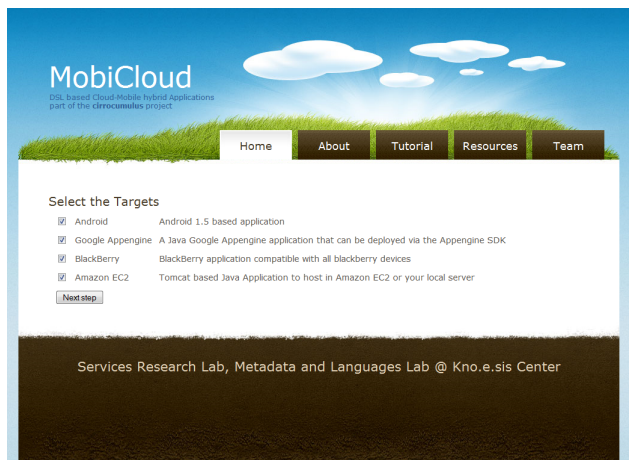


Figure 3: Step 2 : Selection of target platforms

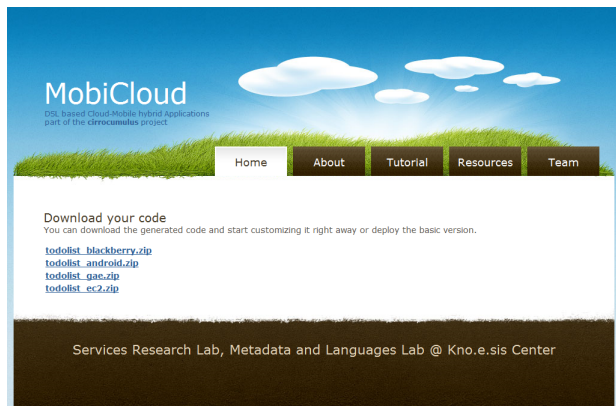
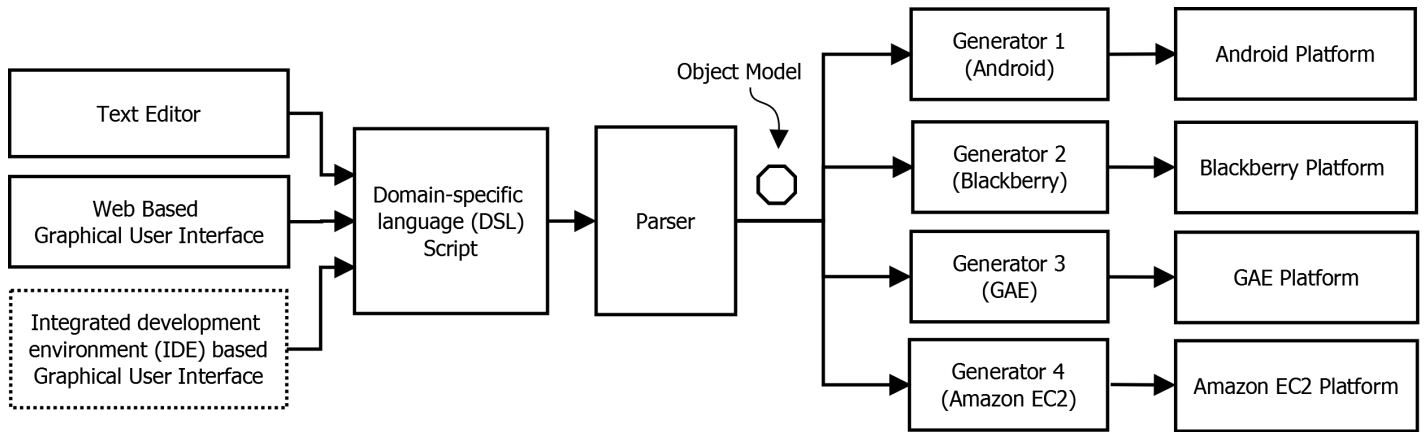


Figure 4: Step 3 : Download the generated applications



**Figure 5:** System Implementation Components and Flow

Website [3]. A detailed evaluation is available in the technical report [4].

#### 4 Tool Workflow

The workflow for the MobiCloud online tool to create this application is kept as simple as possible. Figures 2,3 and 4 illustrate the different steps of the workflow.

- Step 1 of the workflow is to enter the DSL text as illustrated in Figure 2. The online editor supports text highlighting and other text editing actions. As part of this step we plan to add a drag-and-drop graphical UI in the future.
- Step 2, as shown in Figure 3, allows the selection of the target platforms. Users can select various combinations of Cloud and mobile platforms in this step.
- Step 3 enables the users to download the generated code in compressed form. This is illustrated in Figure 4. The generated code can be readily compiled and deployed using an appropriately configured development environment.

#### 5 Conclusion

The conditions are right for a boom in CMH applications and our contributions stand to provide a well-defined methodology to exploit them. The use of DSL shields developers from minor details of the target platform and reduces the number of defects by auto-generating communication interfaces. Although the DSL and the toolkit presented in this research has room for many improvements, it has clearly demonstrated the applicability of DSLs in both Cloud and Mobile spaces.

#### References

- [1] D. Coward. Java servlet specification version 2.3. *Sun Microsystems*, 2001.
- [2] O. Kharif. Android's spread could become a problem. <http://bit.ly/d0IHG8>.

- [3] A. Manjunatha, A. Ranabahu, A. Sheth, and K. Thirunarayan. MobiCloud. Available online at <http://knoesis.org/mobicloud> - Last accessed September 12th 2010.
- [4] A. Manjunatha, A. Ranabahu, A. Sheth, and K. Thirunarayan. MobiCloud Technical Report. <http://knoesis.wright.edu/library/publications/MobiCloud.pdf> - Last accessed September 12th 2010.
- [5] R. Moll. Knowing is half the battle, 2009. Available online at <http://bit.ly/cvvWaR> - Last accessed September 12th 2010.