Cloud Centric Mobile Application Development Using Domain Specific Languages

Ajith Ranabahu, Ashwin Manjunatha, Amit Sheth and Krishnaprasad Thirunarayan,
Kno.e.sis Center, Wright State University, Dayton OH USA
{ ajith,ashwin,amit,tkprasad}@knoesis.org

1. Introduction

Stellar growth in use of mobile platforms for business users, combined with rapid business growth in developing countries that have much higher utilization of mobile access have put cloud back-end hosted mobile applications in a sweet spot. By using a cloud mobile combination, computationally intensive services can be delivered right to the consumer anywhere, anytime. Two unmet challenges in developing such applications are managing the development of applications for heterogeneous mobile platforms with equivalent functionality, and maintaining a portable back-end to mitigate any catastrophic outage (such as the recent outage of Amazon EC2[1].)

This article discusses the mobile application aspects of the MobiCloud [1] approach for rapidly developing cloud-mobile hybrid applications for heterogeneous mobile front-ends and cloud back-ends. MobiCloud exploits the features of Domain Specific Languages (DSLs) to address the difficulty of programming for multiple mobile platforms. It also helps to overcome some of the limitations of mobile platforms by pairing with cloud back-ends.

2. Designing Hybrid Applications

Typical applications being developed as mobile device based front-ends for Cloud based applications have the following features.

1. The front-end and back-end applications are usually managed as separate projects. These projects depend on well-defined service interfaces to implement either the client or the service functions.
2. Front-end applications are tied strongly to the back-end application. Updates to the back-end applications would eventually have to be propagated to the front-end applications, vice versa. Such change propagation requires sweeping changes to the front-end application code, often to multiple front-end applications targeted towards different mobile devices.
3. A significant effort is needed to debug these applications, due to their use of remote procedure calls (RPC).

Regardless of the separation of the back-end and the front-end through a service layer, the application components still maintain their relevance to the well-known Model View Controller (MVC) design pattern [2, 3]. For example, the model data structure can be easily identified (equivalent in function yet different in implementation) in both the back-end and front-end.

Thus, in MobiCloud, the entire functionality, i.e. the functionality of the front-end mobile application as well as the back-end cloud application, is modeled as a single unit. The DSL based representation of this modeling can be transformed to the required software components.

This approach has the following benefits;
1. The same model can be used to generate functionally equivalent, mobile front-end applications as well as cloud back-end applications targeting different platforms.
2. Many developers are familiar with the MVC design pattern, thus this modeling has a gentler learning curve.
3. Modeling complexity is reduced by treating the entire functionality as a single unit.
4. Differences in service interfaces are non-existent since the service and the client are generated using the same specification. This relieves a significant burden in debugging.

In the next section, we present a brief overview of the MobiCloud language using an example.

3. The MobiCloud DSL

We present a simple, yet useful example application, made using MobiCloud. Listing 1 illustrates the DSL script to create a task manager application. This script can be generated using the online MobiCloud composer or just written using a basic text editor.

[1] http://blog.rightscale.com/2011/04/25/amazon-ec2-outage-summary-and-lessons-learned/

```
recipe(:todolist) do
metadata({:id => 'task-manager'})
    # models
   model(:task, {:name=>:string,
   :description => :string,
   :time => :date,
   :location => :string})
  #controllers
  controller(:taskhandler) do
    action :create, :task
    action :retrieve, :task
  end
  # views
  view :add_task, {
   :models =>[:task],
   :controller=> :taskhandler,
   :action => :create}
  view :show_tasks, {
   :models =>[:task],
   :controller=>:taskhandler,
   :action => retrieve}
end
```

Listing 1 : MobiCloud script to generate a task manager application.

Figure 1 illustrates the graphical representation of the same application, created using the MobiCloud composer. The composer also supports direct deployments to supported platforms.

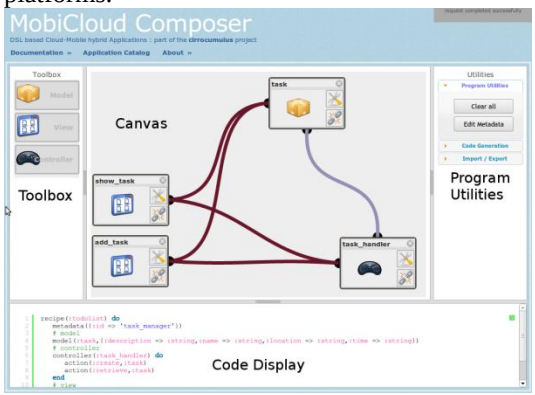

Figure 2 : The MobiCloud composer UI.

For example, an installable Android package (APK) can be downloaded as a result of the compilation when the Android platform is selected.

Figure 2 illustrates the relevance of the generated application to the MVC components. The *model* specification is used to generate two data structures, one for the mobile device and another for the service implementation. The *controller* specification mainly contributes to the server side implementation while the *view* specification contributes to the user interface, primarily on the mobile device.

We conclude the details of the language here for brevity. Manjunatha et al [1] has discussed in detail, the philosophy behind these constructs and their relationships.

4. Advantages to the Mobile Developer

There are many advantages to the mobile application developer, when using MobiCloud

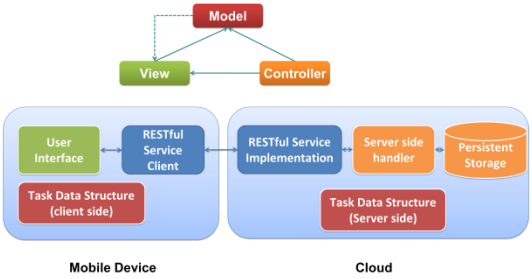

Figure 3 : The relevance of the generated artifacts to the MVC design.

I. *Ability to cover multiple platforms with one code base*: A number of mobile development platforms exist today, each with different development environments, Application Programming Interfaces (API), and programming languages. Fragmentation of APIs, even within a single platform forces mobile application developers to focus on only specific platforms and versions.
MobiCloud helps to alleviate these issues by providing the DSL and the generators that convert high level abstractions to platform specific code representations. Only a simpler high level DSL script needs to be saved and the platform specific code can be generated on demand.

II. *Reduced programming effort*: Developing Mobile applications take significant effort, yet most of it is focused on repetitive code segments. Furthermore some of these code segments depend on external service specifications that are hard to debug in case of errors.
MobiCloud can be used as a boilerplate code generator to cover most of the required but repetitive code segments. Developers can then pursue more creative aspects, such as improving the user interface. MobiCloud also takes care of generating the service clients, ensuring that there are no glitches in the communication aspects. This significantly lowers the programming effort.

III. *Platform specific UI and code optimizations*:
There are many platform specific optimizations that can be done on each of the mobile platforms. Interpreter based universal solutions such as J2ME are not very popular primarily due to their inability to exploit these platform specific features. MobiCloud uses a generative approach and thus, has the ability to generate the optimal code for a given platform.
For example, in the case of Android, the UI components are created using Android widgets while for Blackberry; the corresponding Blackberry widget classes are extended. This enables the applications to present native widgets in their interfaces rather than adhering to universal look and feel.

5. Future Enhancements
There is a host of planned enhancement to MobiCloud and we highlight the key enhancements targeted towards the mobile front-ends.

1. *Sensor Integration*: Smartphones are equipped with myriad of sensors. These sensors play a vital role in developing intelligent applications. We plan to provide abstractions over commonly used sensors to improve the applicability of MobiCloud.
The following DSL snippet highlights how we plan to integrate the sensor data via special data types.

```
model(:bird,
{:name=>:string,
 :image => :camera_image,
 :time_seen=>:current_clock_time,
 :location_seen
=> :current_gps_coords})
```

Listing 2 : A model construct for a bird watchers application that integrates sensor data as data types.

In listing 2, data types like **:camera_image** and **:current_gps_coords** can generate platform specific code to obtain an image from the camera or populate the location from the built in GPS. The generators can also insert code that support graceful degradation when the sensors are absent or not functional. For example, when the GPS sensor is not active, the UI can simply present the user with two text boxes for the coordinates, indicating that the GPS sensor is inactive.

2. *Multimedia and third party integrations*:
Many mobile applications are integrated with third party services. Maps, videos and social network features are a few of the popular 'must have' integrations.
We have already introduced an extension mechanism to MobiCloud that enables third party extensions to the language constructs. We plan to add extensions to enable integrations with the popular services. The extension mechanism also gives the flexibility to advanced programmers to write custom integrations.

3. *Power and connectivity aware processing*:
Mobile devices have limited storage, processing power and connectivity. Applications can be made to make smarter use of the battery power and connectivity by dynamically changing the behavior of the application. This however, takes significant effort in programming.
MobiCloud can automatically generate the required code to efficiently manage the communication. For example, it can store content locally when the connectivity is via 3G and the data size is large, and complete the transfer when Wi-Fi is available, thereby saving power and bandwidth. We plan to add improvement to the code generators to automatically add data transfer optimization code.

6. Conclusion
Combining mobile devices and clouds is becoming important, but the difficulty of programming remains a serious problem. Our MobiCloud project has shown that by providing proper abstractions via a DSL, one can overcome the complexities in programming cloud-mobile hybrid applications. Given that mobile platforms are becoming more complex every day, we believe that proper abstractions will be the most important programming tool for mobile devices in the future.

References
[1] Manjunatha, A.; Ranabahu, A.; Sheth, A.; Thirunarayan, K.; , "Power of Clouds in Your Pocket: An Efficient Approach for Cloud Mobile Hybrid Application Development," Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on , vol., no., pp.496-503, Nov. 30 2010-Dec. 3 2010.

[2] T. Reenskaug. The Original MVC Reports. Xerox Palo Alto Research Laboratory, PARC, 1978.
[3] S. Burbeck. Applications Programming in Smalltalk-80: How to Use Model-View-Controller (MVC). Softsmarts,Inc., 1987.

Ajith Ranabah is a PhD candidate in computer science at the Kno.e.sis center in Wright State University. His primary research is focused on application and data portability in Cloud computing. Contact him at ajith@knoesis.org.

Amit Sheth is an IEEE Fellow, the LexisNexis Ohio Eminent Scholar and the director of the Ohio Center of Excellence in Knowledge-enabled Computing (Kno.e.sis) at Wright State University. Contact him at amit@knoesis.org

Ashwin Manjunatha is a recent graduate from the Ohio Center of Excellence in Knowledge-enabled Computing (Kno.e.sis) at Wright State University. His research interests include Cloud Mobile Hybrid Applications and Large Scale Data Analysis. He is presently working on a technology to bridge cloud computing and mobile computing. Contact him at ashwin@knoesis.org

Krishnaprasad Thirunarayan is a Professor in the Ohio Center of Excellence in Knowledge-enabled Computing (Kno.e.sis) at Wright State University. Contact him at tkprasad@knoesis.org