MobiCloud

Power of Clouds in Your Pocket : An Efficient Approach for Cloud Mobile Hybrid Application Development

Ashwin Manjunatha, Ajith H. Ranabahu , Amit Sheth, Krishnaprasad Thirunarayan

Kno.e.sis Center

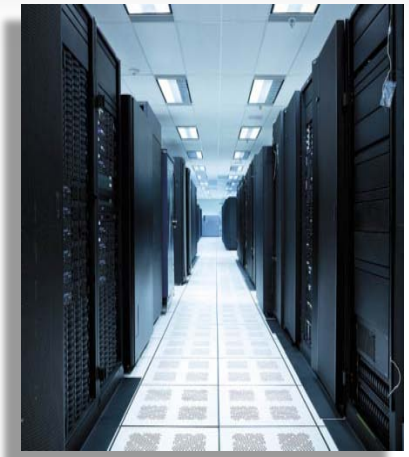
Wright State University

Agenda

- What are Cloud-Mobile hybrids ?
  - Why are they hard to build ?
- Building Cloud-Mobile hybrids with DSLs
  - Pros and cons
  - Current DSL
- MobiCloud Toolkit
  - Ongoing work
  - Future work
- Questions

What are Cloud-Mobile Hybrids ?

Spectrum of Computing Power



Google App Engine

amazon web services

Salesforce

Windows Azure

MOTOROLA

android

NOKIA Connecting People

BlackBerry

Portability →

← **Computing Power**

Scalable
On-demand
Economical

Portable
Ubiquitous
Connected

Cloud-Mobile Hybrids (CMH) ?

- Applications that span over multiple devices
  - Back-end in Cloud
  - Front-end in a mobile device
- Both components needed for the app to function
- Front-end is not just a webpage !
  - A native Iphone / Android / Blackberry app

Simple Example

- The Facebook App
  - The Facebook client you have in your smart phone !
    - Mobile front-end for Facebook
      - Most Facebook activity happens via Mobile devices !
    - Many actions need extensive processing in the Facebook backend
    - Both parts (back-end and the front-end) required for the complete experience

Another Research Oriented Example

- Privacy score[1]
  - Introduced by IBM researchers
  - Measures relative exposure of private data on a social network
  - Provides a similar number to “credit score”
- Requires heavy calculations in the back-end
- Front-end is simply one number !
  - Perfect for a mobile device !

1. A Framework for Computing the Privacy Scores of Users in Online Social Networks,
Kun Liu, Evimaria Terzi
Ninth IEEE International Conference on Data Mining, 2009

Are CMH Apps hard to build ?

- Yes - very much !!
- Clouds are heterogeneous
  - Write the back-end to suit Amazon – You can't move to Google !!
- Same in mobile devices
  - Need to write different apps for different devices !
    - Android / Iphone / Blackberry are all different

Our solution to this Problem…

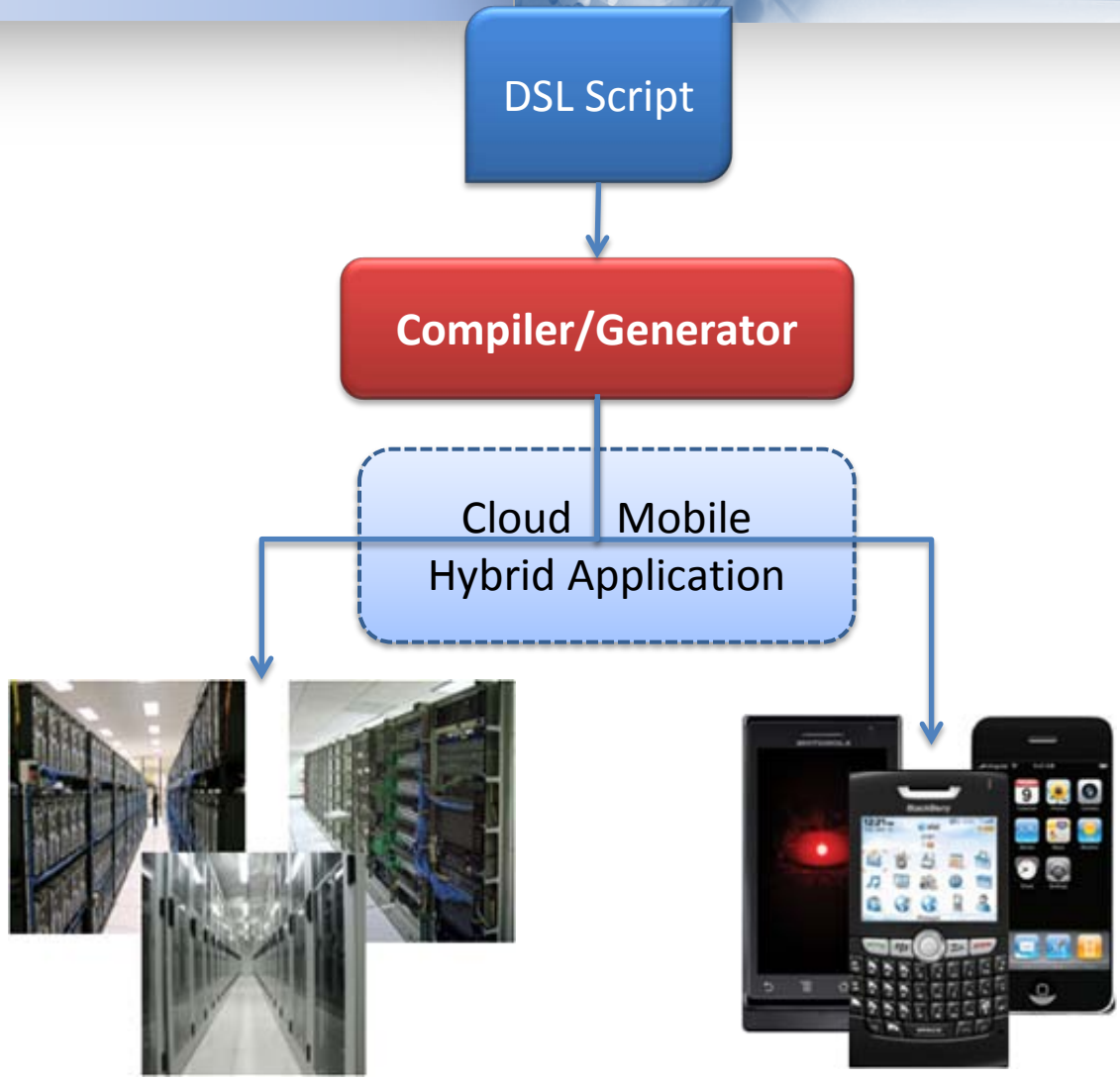
Use a DSL

- DSL ?
  - **D**omain **S**pecific **L**anguage
  - A mini language for a special purpose
    - Make / Ant
    - Matlab
    - Many other examples
- Use a DSL to ***generate*** an application

What changes by using a DSL ?

- Reduce Complexity
  - As DSLs are designed for specific domain, they reduce complexity!
- Increases Flexibility and avoids vendor lock-in
  - Use single DSL to design applications with different combinations of Cloud and mobile platforms
  - Generate apps for Amazon, Google, Android, Blackberry using just a single script
- Ease of use
  - No separation between front-end / back-end
  - No service interfaces to worry about (auto generated)
  - Communication interfaces are a major source of errors and incompatibilities

A birds-eye view of what we do



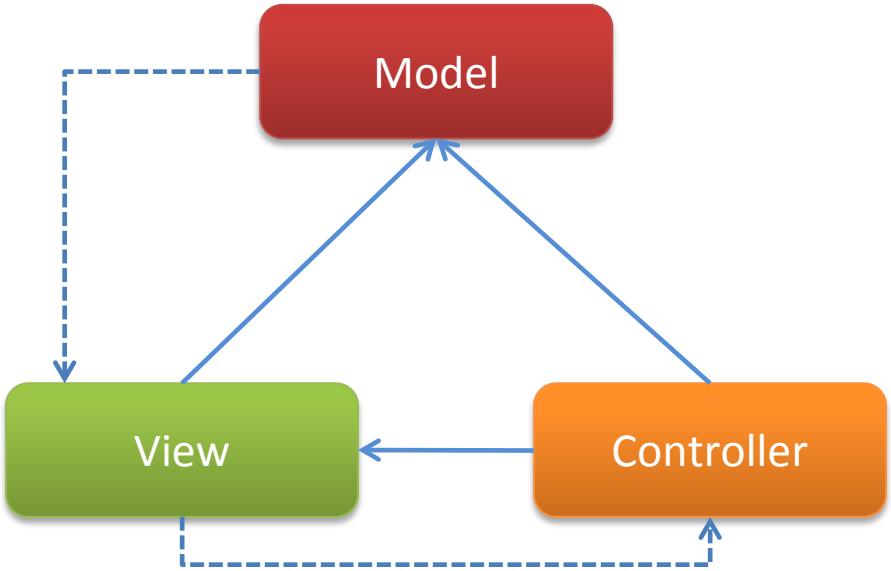
Is this the silver bullet ?

- Nope !
  - Less control over the code
    - E.g. Extensive GUI customization and device integration not possible
  - Covers only the 80% case
    - E.g. Not suitable for games or other UI intensive applications

Our current DSL

DSL design principle

- Based on Model-View-Controller (MVC) design pattern



A very simple "Hello World"

```
recipe :helloworld do
 metadata :id => 'helloworld-app'
  # models
   model :greeting, {:message => :string}

  #controllers
   controller :sayhello do
    action :retrieve,:greeting
   end

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

Metadata – details that need to be attached to the whole application

Models

Controllers

Views

A slightly complicated Example (task manger)

```
recipe(:todolist) do

  # specific metadata for this app
  metadata({:id => 'todo-list'})

  model(:todoitem, {:name=>:string, :description => :string,:time =>
:string, :location => :string})
  model(:user, {:name=>:string, :bday => :string})

  #controllers
  controller(:todohandler) do
    action :create,:todoitem
    action :retrieve,:todoitem
    action :update,:todoitem
    action :delete,:todoitem
  end

  # views
  view :todo_add, {:models >[:todoitem],:controller => :todohandler,:action
=> :create}
  view :todo_show, {:models =>[:todoitem],:controller =>
:todohandler,:action => :retrieve}

end
```

Regular development vs DSL for CMH

Regular Development

- Developed as two applications
- Different platforms need new effort.
- Significant effort in creating code and other artifacts
- Highly customizable

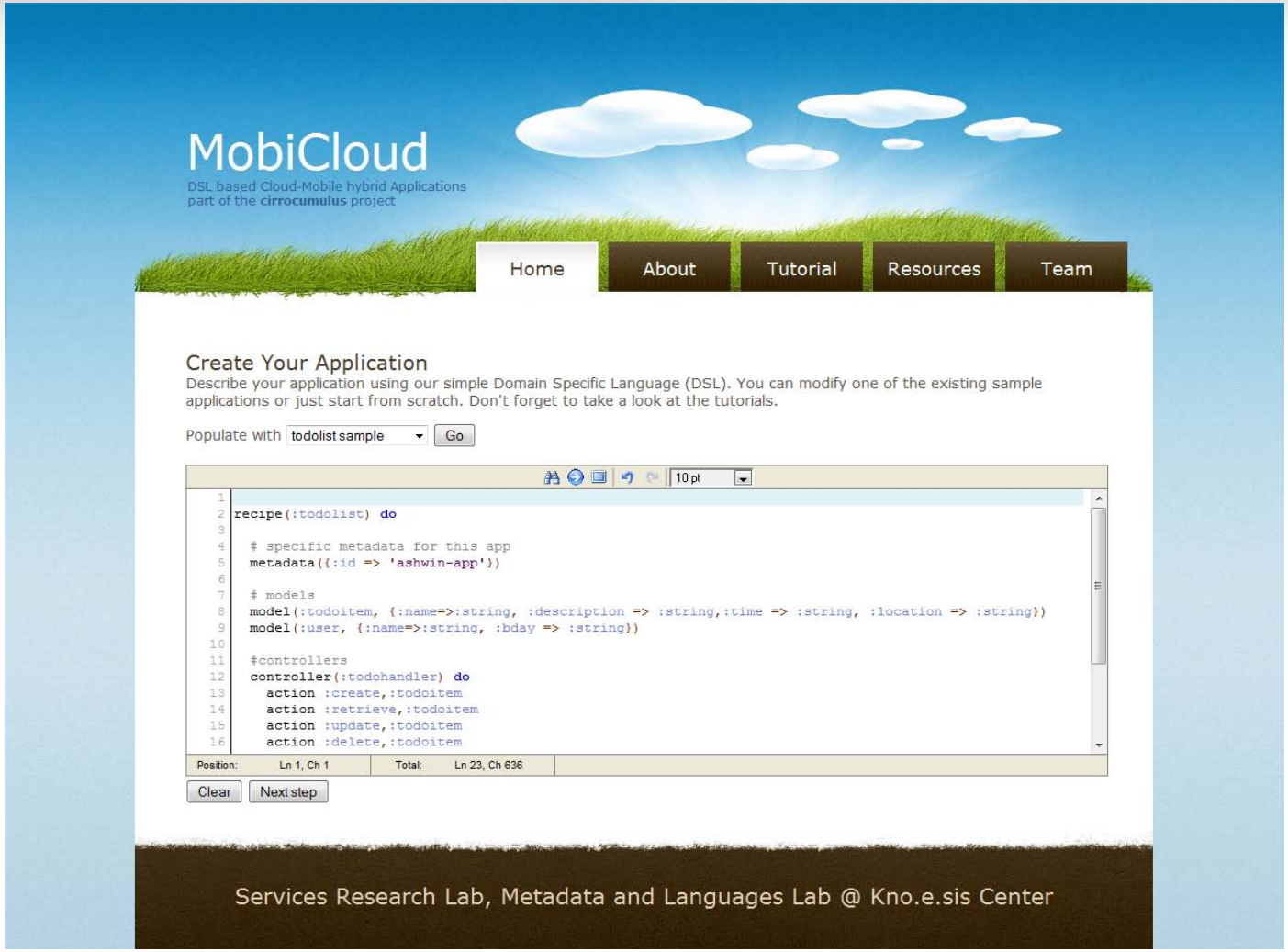
DSL based Development

- Developed as a single application
- Generators create functionally equivalent applications for multiple platforms
- Minimum effort in creating all required artifacts
- Limited customization

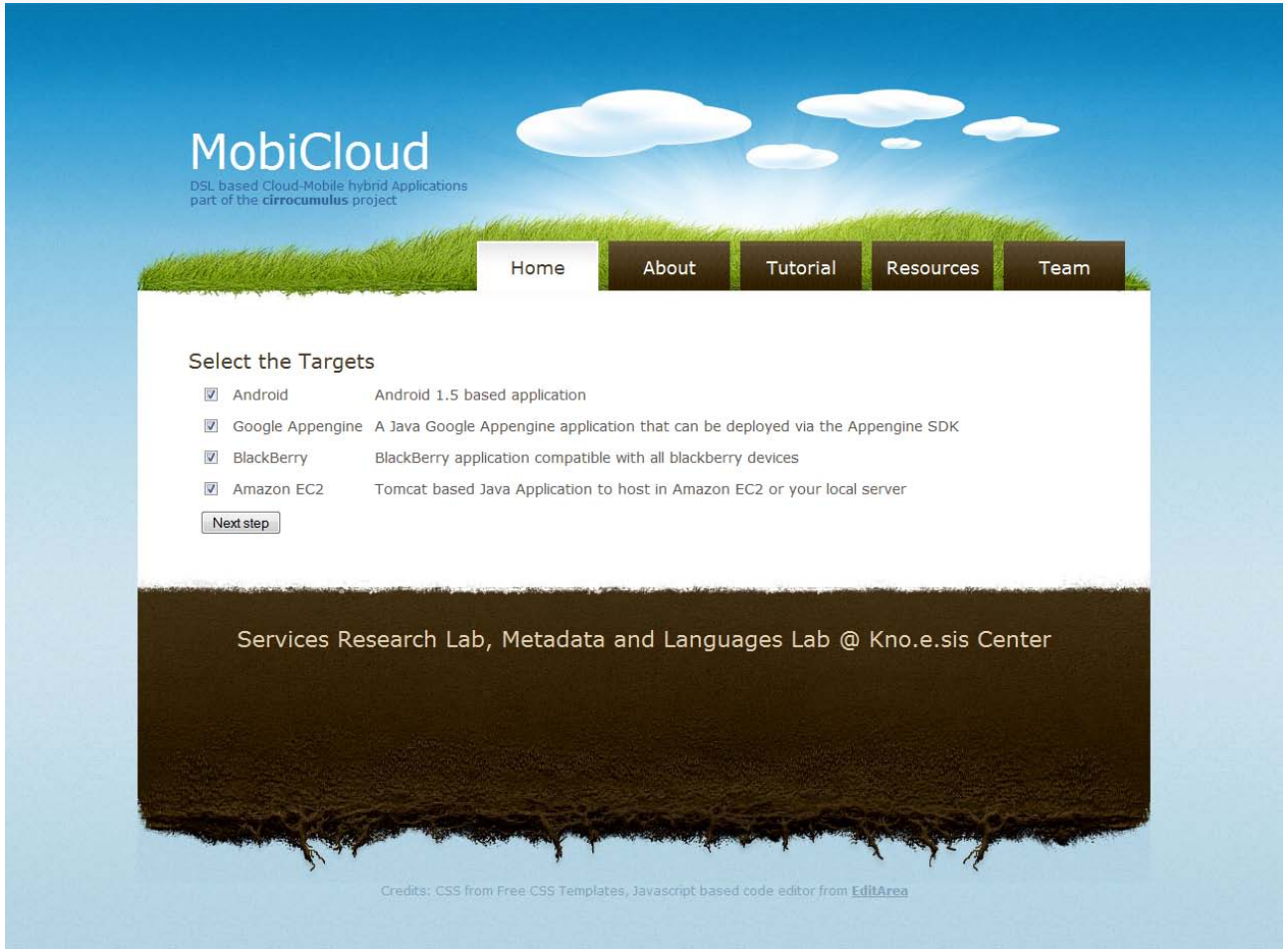
Other Benefits of DSL Based Development

- Convenient integration of other features
  - Location based services
    - Integration with available location sensors, e.g GPS
  - Non functional features
    - Security
  - Social Network features
    - Publish to Facebook, Twitter etc

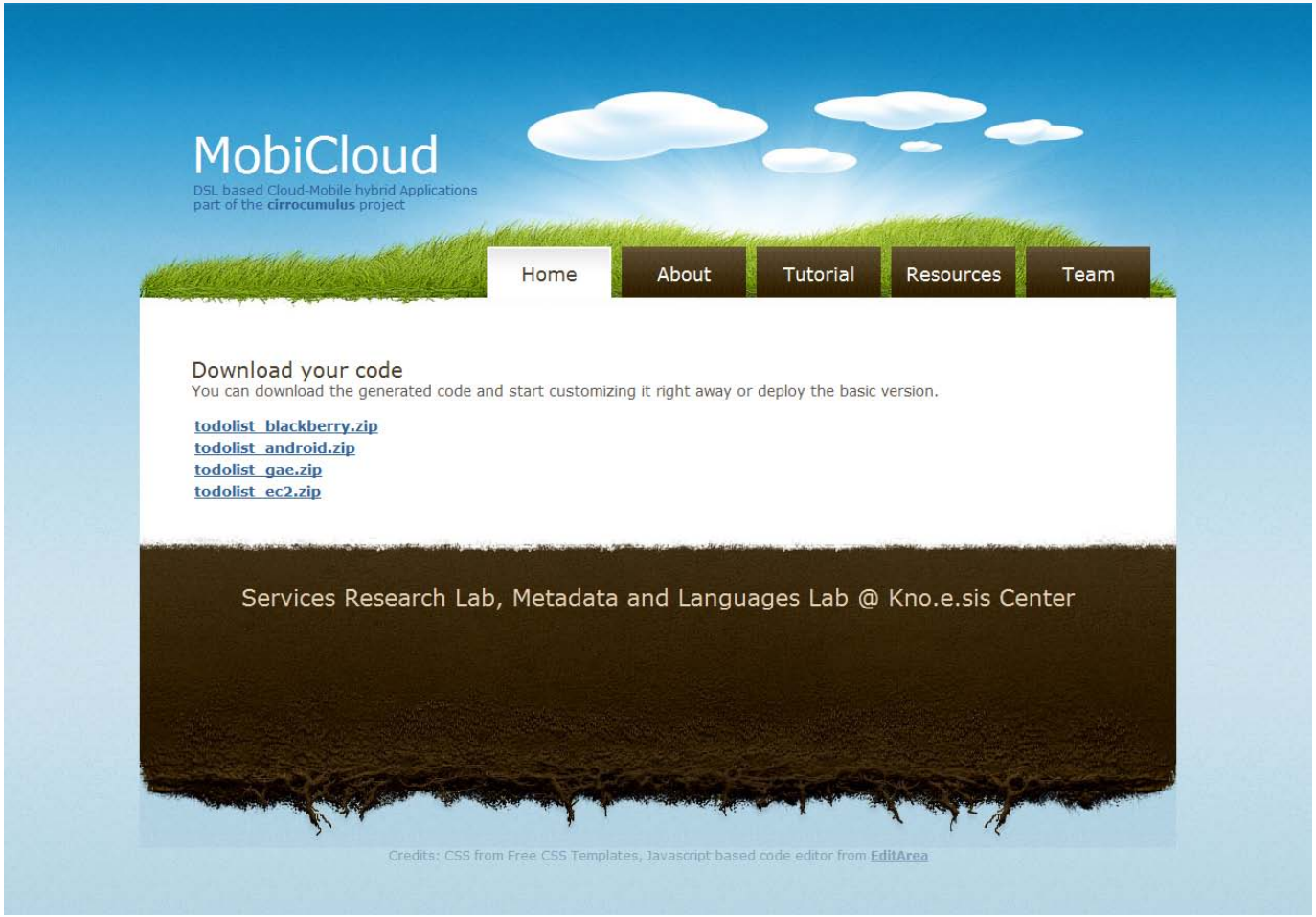
MobiCloud Online Toolkit – Step1



MobiCloud Online Toolkit – Step2



MobiCloud Online Toolkit – Step3



References / More details

- Technical report on MobiCloud
  - http://knoesis.wright.edu/library/publications/MobiCloud.pdf
- Publicly hosted MobiCloud tool
  - http://knoesis.org/mobicloud
- Code repository
  - Coming soon !
- Privacy score paper
  - http://portal.acm.org/citation.cfm?id=1674659.1677075

Researchers

Ashwin Manjunatha

Ajith Ranabahu

Amit Sheth

Krishnaprasad Thirunarayan

Thank you

Questions ?