Semantic Centric Solutions for Application and Data Portability in Cloud Computing

Ajith H. Ranabahu, Amit Sheth
Kno.e.sis Center
Wright State University

2nd IEEE International Conference on Cloud Computing and Technology
Indianapolis IN
Agenda

• Challenges in Cloud Computing
  – The portability problem
• Role of Semantics and the Modelling space
• Use of semantics in programming for the Cloud
  – Semantic Enhancements to a Domain Specific Language
• A brief overview of our vision
  – The Cirrocumulus project
• Questions
Challenges in Cloud Computing
Cloud Landscape as of Today

- **Increasing portability**
  - Cloud User Responsibilities
  - Cloud Types
  - Providers

- **Increasing automation**
  - Infrastructure as a Service (IaaS)
    - Amazon EC2
    - Rackspace
    - IBM HiPODS
  - Platform as a Service (PaaS)
    - Google AppEngine
  - Software as a Service (SaaS)
    - Salesforce CRM

Application artifacts, Support software and configurations, System level configurations and management

Application artifacts and limited set of configurations.

Application specific configurations and customization.
The Vendor Lock-in Problem

• Once you select a Cloud then it is hard (almost impossible) to move to a different one
  – Both application logic as well as data!

• Vertical vs Horizontal
  – Vertical
    • Move within the same type of Clouds
  – Horizontal
    • Move across different types of Clouds
Objective of Our Research

- A methodology to develop applications in a portable manner
  - Use Semantic technologies to streamline this process
Knowing is half the battle!
A Clear Understanding of the Applications
Four Types of Semantics for an Application

• Inspired by the four types of Semantics for Services¹,²

• Enables a clear separation of issues


Applications development is a complicated process
- Multiple parties / Multiple stages
- Many modeling approaches already available

Four types of Semantics need to be carried across these stages
Partitioning of the Modeling space
From High level models to a Concrete Grounding

• We need both high level modeling and running code!

• High level models
  • Easier to understand
  • A birds-eye view

• Executable code
  • Detailed and intricate
  • May not give a clear understanding of what it does
  • It works!
Semantics and Domain Specific Languages
Using DSLs to Develop applications

• DSL?
  – Domain Specific Language
  – 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?

- DSLs are abstract
  - Reduced complexity

- Minimum separation between multiple components
  - E.g. No service interfaces to worry about (auto generated)
Why Semantics?

- Semantic Web community has tackled the portability problem for years!
  - Semantic models are capable of providing generic abstractions over data, functional, non-functional and system aspects
  - Referring to these models rather than re-implementing code, significantly reduces the development effort
Example

• The MobiCloud DSL
  – Generates Cloud-Mobile hybrids
  – Capable of addressing multiple clouds as well as mobile devices

• Presented tomorrow in the Work In Progress track as well as the demo session.
Is this the silver bullet?

• Nope!
  – Less control over the code
    • E.g. Extensive customization and device integration may not possible
  – Covers only the 80% case
    • A given DSL is only good in covering its own domain.
    • E.g. The MobiCloud DSL is not suitable for games or other UI intensive applications
Using the MobiCloud DSL as a Base for Semantics
MobiCloud 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

Linking the model using its unique name

Linking the controller using its unique name
Mapping of Hello World to Cloud and Mobile Spaces

Model

View

Controller

Mobile Device

Cloud

UI

RESTful Service Client

Greeting Data Structure

RESTful Service Implementation

Server side handler

Persistent Storage

Greeting Data Structures
Breaking the Hello World with Respect to Semantics

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

Non-functional details and Metadata

Data Definitions

Logic and Process Definitions
A slightly complicated Example (task manager)

```ruby
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
```
Data Semantics

• Reusable Data Definitions
  – RDF based Platform-neutral data definitions
  – Reference from the app rather than redefine
  • E.g. Use FOAF definition for person

  ```ruby
  model: person , { :ref => "foaf:Person" }
  ```
Non-Functional Semantics

• Non-functional Capabilities
  – Attach non-functional configurations (e.g. Security) via profiles
  – Define abstract profiles using semantics
    • The generator can insert the necessary code
    • May need global (application-wide) configurations
      – declared in the metadata section to be applicable to the whole application
      – E.g. Strength of encryption for security
System Semantics

- **System Configurations**
  - Define abstract deployment / Management configurations using semantics
  - Already available in Elastras Language suite
    - ECML, EDML, EMML
      - RDF based modeling of core system aspects
recipe(:todolist) do

metadata({:id => 'todo-app'},
    {:deployment_profile => "http://www.elastra.com/sites/default/files/ExampleApacheStack-full.ttl" })

model(:todoitem, {:name=>:string, :description => :string,:time => :date, :location => :string})
model(:user, {:ref => "foaf:Person"})

controller(:todoscheduler) do
    action :create,:todoitem,
        {:security_profile => "ssl"}
    action :retrieve,:todoitem

    ...
end

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

end
Evaluating the Language is Hard!

- Preliminary code metrics
- Not entirely fair
  - Some generated code such as XML descriptors is not counted
Porting Code

• The generators take care of translating the DSL to platform native code
  – DSL code is portable as long as the platform adapter exists
Porting Data

• The lifting-lowering mechanism for data transformation
  – Availability of a platform independent model allows the transformation of a platform specific instance to a independent one.
Our Broader Vision
The *Cirrocumulus* Project

- **End-to-End coverage for Cloud application development**
  - Develop, deploy and manage Cloud applications without depending on specificities of any Cloud

- **Use DSLs and Middleware together**
  - Inspired by IBM Altocumulus and IBM Sharable code projects
  - Use semantics as a core binding across all stages
The *Cirrocumulus* Project

- DSL Based Development fits in here

- DSL based design and development process
- DSL Verification
- Cloud middleware layer for deployment and management

- Semantic meta models
- DSL assembly encapsulating data, functional and non-functional characteristics

- Google App Engine, Amazon Web Services, Microsoft Azure
<table>
<thead>
<tr>
<th><strong>Libcloud</strong></th>
<th><strong>Deltacloud</strong></th>
<th><strong>Cirrocumulus</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="http://incubator.apache.org/libcloud/" alt="Link to Libcloud" /></td>
<td><img src="http://deltacloud.org/" alt="Link to Deltacloud" /></td>
<td><img src="http://circucl.org/" alt="Link to Cirrocumulus" /></td>
</tr>
<tr>
<td>• Developer centric program abstraction over Infrastructure cloud operations</td>
<td>• Developer centric service abstraction over Infrastructure cloud operations</td>
<td>• Developer centric abstraction over applications and cloud operations</td>
</tr>
<tr>
<td>• Support in only Python and Java</td>
<td></td>
<td>• Supports IaaS as well as PaaS Clouds</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Code can be generated in any language</td>
</tr>
</tbody>
</table>
The Cirrocumulus Project (Cont)

LibCloud / DeltaCloud can be used in building this component
References / More details

- Technical report on MobiCloud
- Cirrocumulus project
Time for Questions?
Thank you
• Why Cloud will never be free