infrae

Mobi:
a mobile user agent lib

EuroPython Conference
Birmingham, UK
21 July 2010

KIT BLAKE
ANTONIN AMAND

Agenda

- Introduction
- Background: development at MIT
- Mobi lib implementation
- Mobi demonstration
- Roadmap
- Code examples

Announcing Mobi

What is it?

- A set of Python libraries and WSGI middleware
- That does two things:
  1. Device detection
  2. Content rendering
- Rendering depends on the detection

Background

- Mobi was inspired by the MIT's Mobile Web implementation
- "A suite of web-enabled mobile services"
- Offers *on the fly* information to MIT community and visitors
- Point your mobile device at: http://m.mit.edu/

MIT Documentation

- Case Study:
  “Massachusetts Institute of Technology: Transforming the Campus Experience with the MIT Mobile Web”
- by Bob Albrecht and Judith A. Pirani
- ECAR Case Study 3, 2009 (PDF)
- http://mobi.mit.edu/about/

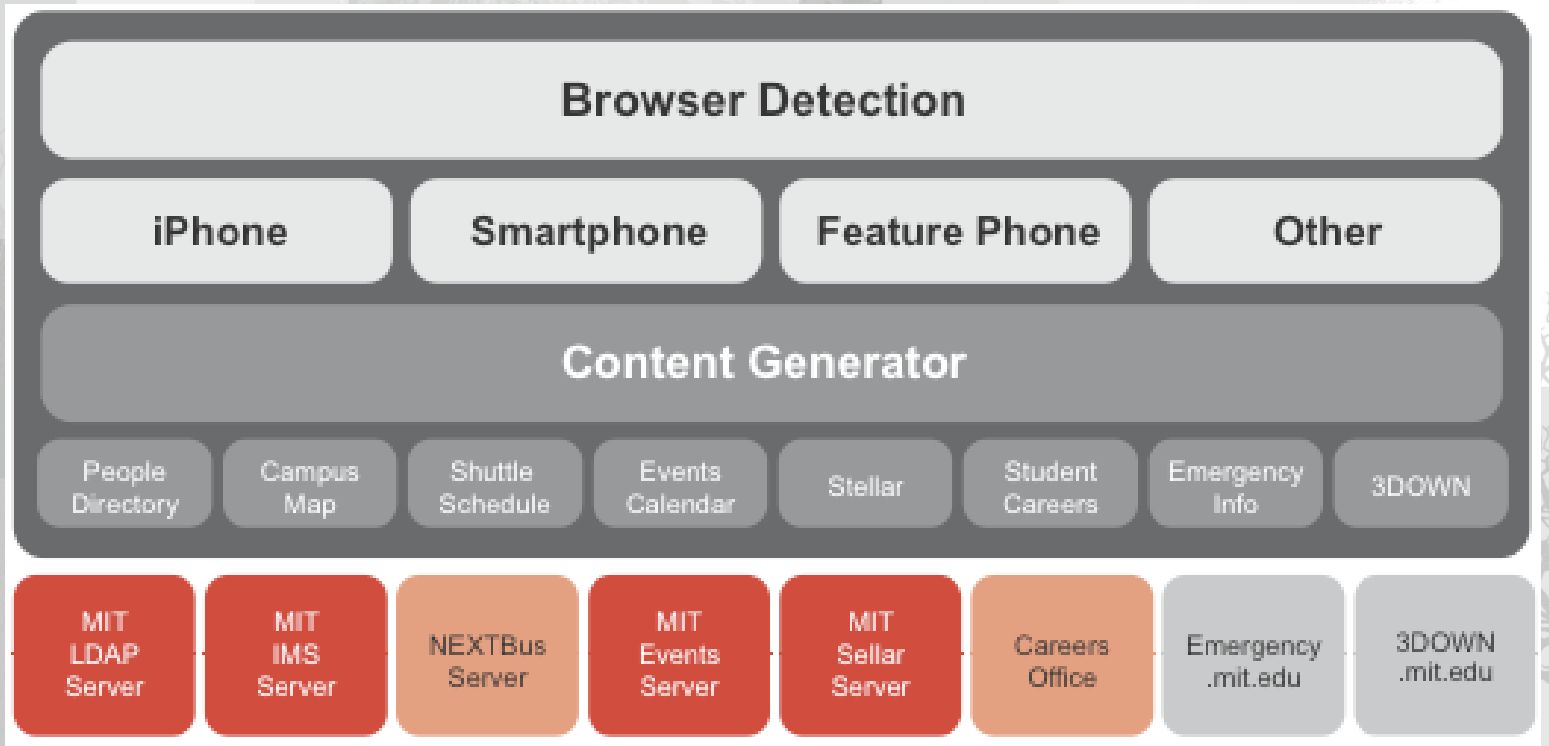
Development at MIT

- IS&T *Big Initiative*
- Initial budget was $250,000
- Began in 2007, released in June 2008
- Team had members from various depts plus external consultants
- Initial release included 7 mobile services

Project Fundamentals

- IS&T observed two fundamental principles when creating its mobile services:
  1. create a device-independent, standards-based platform
  2. piggyback on currently available web content
- MIT chose to facilitate further development by releasing the code as an open source resource

Neutral Service Platform



EDUCAUSE Center for Applied Research

Layer 1: Browser Detection

- System reads the device's browser user agent string
- Mobile web space is much more fractured than the desktop web
- Tens of thousands of different user agent strings
- System references a configuration file, the WURFL "Wireless Universal Resource File" database

What is WURFL?

- An open source XML configuration file maintained at SourceForge
- http://wurfl.sourceforge.net/
- Contains info about all known wireless devices on Earth!
- Lists approximately 100 capabilities and features for over 600 mobile phone models
- Maintained by Luca Passani and contributors

Layer 2: Device categorization

- System categorizes mobile user agents into one of three categories:
  1. Advanced (smartphone: iPhone/iPod Touch and Android)
  2. Standard (touch screen: Blackberry, Palm Treo, Windows Mobile)
  3. Basic (cellphones with WAP)
  4. Other, for desktops

Layer 3: Content Generation

- MIT chose not to create new content for their Mobile Web
- Instead *piggyback* on currently running services
- Backend communicates via standard interfaces with preexisting databases and services
- Generates specially designed Mobile Web content pages

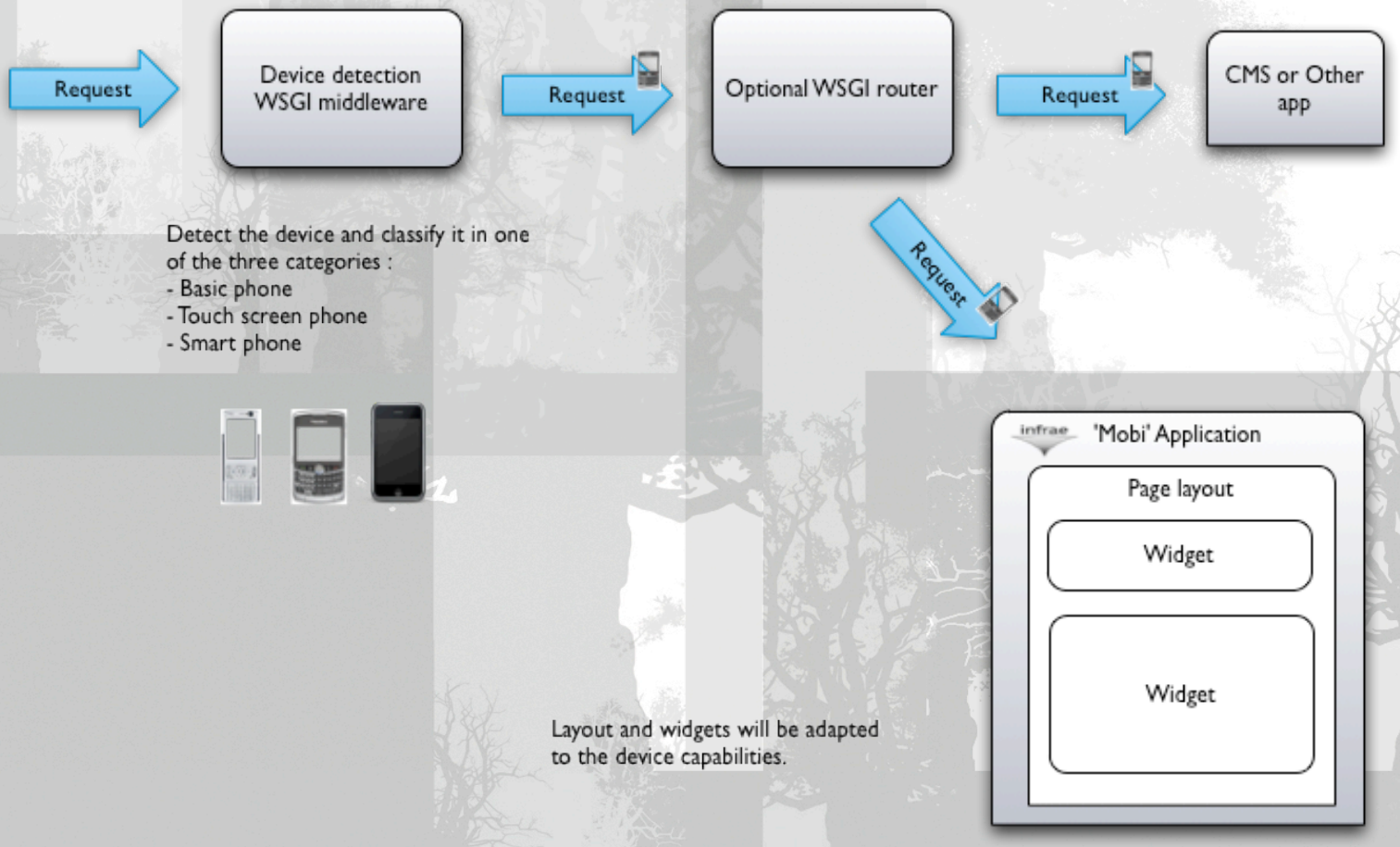
Development cycle

- Four phases:
  1. Surveys and service selection
  2. Feature development
  3. User testing (including two usability studies)
  4. Rollout

Setting up Mobi

- Our objective was to retain the *embedded knowledge*
- MIT's solution includes templates and code
- Templates were easy to convert to TAL
- Code is written in PHP
- We *walked through* the code, then rewrote it from scratch in Python

Mobi architecture



Mobi devices library

- Consists of classifiers that can detect mobile phone devices. Two classifiers are currently implemented:
  1. MIT Classifier
  2. WURFL Classifier
- An incoming User Agent string is first checked against the MIT classifier
- If the User Agent string isn't found the library falls back to the WURFL file

WSGI detection middleware

- To increase performance the MIT Classifiers are stored in a JSON file
- Exported from the database of the MIT mobile service
- Can be downloaded from their site
- In our tests, all the old phones we have were found in the database

Mobi in action

- See Mobi integrated with an LDAP server
- Point your mobile device at: http://m.test.infrae.com/
- What You See Is... Dependent On Your UA

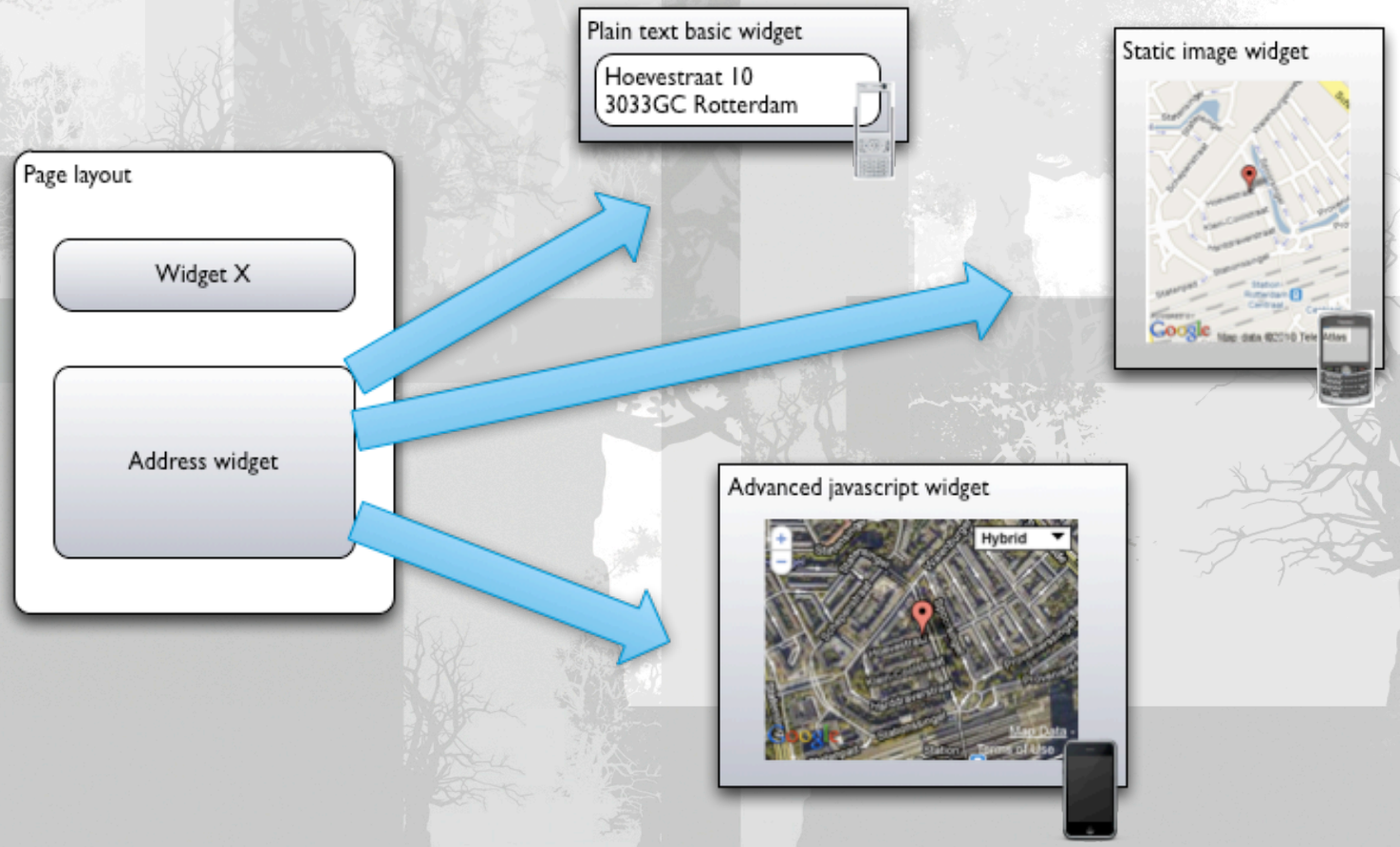
Mobi rendering

- Adapts the presentation for the device requesting it
- Viewing with a desktop browser? Use:
  http://m.test.infrae.com/?__dt=advanced
  http://m.test.infrae.com/?__dt=standard
  http://m.test.infrae.com/?__dt=basic

Mobi widgets

- Widgets are classes and snippets of HTML that are rendered for a target set of devices
- The goal is to render a piece of content, like a phone number link, in the most suitable way for the device that issued the request
- For example one widget is responsible for the rendering of the address of a person

Mobi widget example



Rendering system technical overview

- Based on the Chameleon templating markup language and Zope CA
- Three types of widgets:
  1. Base widget: renders an arbitrary object
  2. Page widget: renders a page with some other widgets inside
  3. Field widget: renders a field / attribute (`zope.schema`) of an object

Writing a widget

- Writing a widget consists of three steps:
  1. Writing a Python class for the widget
  2. Writing a Chameleon template
  3. Registering it in the Zope component architecture

Roadmap

- A new middleware has been added to redirect phone devices to a mobile virtual host
- It will be under the Mobi device detection middleware, and be called `mobi.router`
- The wsgi stack will be as follows:
  - `mobi.devices` detection middleware
  - `mobi.router` middleware
  - application (e.g. silva.app or other)

Roadmap II

- Open Mobile Alliance: encourages mobile device manufacturers to publish UAProf (RDF) files, which describe their devices
- WURFL uses these to add devices
- MIT database remains at revision 2
- We've added devices, like Firefox Mobile and the iPad, to our database
- We could develop tools to automate management and distribution of this database

Layers

- Layers allow you to register :
  - static files: css, javascript
  - views
  - layout elements

1 device type - 1 layer

```
class IBasicLayer(IBrowserLayer):
    pass

class IStandardLayer(IBasicLayer):
    pass

class IAdvancedLayer(IStandardLayer):
    pass
```

Rendering an address

```
<div class="person">
  <div>
    ${person.first_name}
    ${person.last_name}</div>
  <div class="map"
       tal:define="view path:person/address/@@map"
       tal:content="structure view" />
</div>
```

Basic address view

```
class AddressView(grok.View):
        grok.layer(IBasicLayer)
        grok.name('map')
        grok.context(IAddress)

        def render(self):
            format = u'Address: %s, %s<br/>%s %s, %s'
            return format % (
                self.context.street,
                self.context.number,
                self.context.postal_code,
                self.context.city,
                self.context.country)
```

Address view with static image

```
class StaticGoogleMapAddressView(grok.View):
        grok.layer(IStandardLayer)
        grok.name('map')
        grok.context(IAddress)

        # [...]

        def render(self):
            html = u'<img src="%s" alt="location" />'
            return html % self.map_image_url()
```

Javascript Google map address view

```
class JavascriptGoogleMapAddressView(grok.View):
      grok.layer(IAdvancedLayer)
      grok.name('map')
      grok.context(IAddress)

      # do advanced stuff with javascript
      # the actual code doesn't really matter...
```

Documentation and code

- Source code at: https://hg.infrae.com/

Questions?

Thank you

and thanks to the MIT Mobile Web group