Overstock.com
slandis@overstock.com

Restlet

Lightweight REST Framework for Java

Overstock.com
slandis@overstock.com

Outline

- REST Architectural Style
- Restlet Project
- Restlet Programming
- Restlet & Other Technologies
- Deployment Options
- Overstock.com Experience
- Q & A

Overstock.com
slandis@overstock.com

REST Architectural Style

Overstock.com
slandis@overstock.com

What is REST?

- REpresentational State Transfer
- Formalized by Roy Fielding in his PhD Dissertation
- Primarily applicable to distributed hypermedia systems
- Think of it as resource-orientation
  - Resources represent the domain concepts

Overstock.com
slandis@overstock.com

Roy’s Motivation for REST

- Architectural model for how the Web was designed and *should* work
- Serves as a guide for Web standards
- REST has been applied to:
  - Describe the desired Web architecture
  - Help identify existing problems
  - Compare alternative solutions

Overstock.com
slandis@overstock.com

Your Motivation for REST

- Take advantage of what the Web does well
  - Simplicity
  - Scalability
  - Performance
  - Ease of use
- So much nicer than the alternatives
  - SOAP & WS - *
- Unifies Web Sites and Web Services into consistent Web Applications

Overstock.com
slandis@overstock.com

A Style, Not a Standard

- But REST guides the use of standards
- For example:
  - HTTP (Connector)
  - URI (Resource)
  - XML, HTML, GIF, etc. (Representations)
  - text/xml, text/html, image/gif, etc. (Media types)
- The Web is a REST system

Overstock.com
slandis@overstock.com

What is an Architectural Style?

"…a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style."
- Dr. Roy Fielding

- Some Network-based Architectural Styles
  - Pipe-and-Filter
  - Client-Server
  - Layered
  - Virtual Machine
  - Code on Demand
  - Mobile Agent
  - Event-based
  - Distributed Objects

Overstock.com
slandis@overstock.com

REST Architectural Style

- Composition of styles that gains their benefits:
  - **Client-Server** - separation of concerns, scalability
  - **Layered** – allows intermediaries (proxies, firewalls) without affecting interfaces
  - **Stateless** –scalability
  - **Cacheable** – reduces payload & latency
  - **Pipe-and-Filter** – dynamic component connection

Overstock.com
slandis@overstock.com

Representational State Transfer

- Imagine an application as a network of web pages
  - Virtual state-machine
- The user progresses by selecting links…
  - State transitions
- …resulting in the next page…
  - <u>Representing</u> the next <u>state</u> of the application
- …being <u>transferred</u> to the user

Overstock.com
slandis@overstock.com

State Transitions in REST

- *Numbers are resources (URIs, eg., hyperlinks)*
- *Letters are representations (HTML, XML, jpg, etc), that may contain hyperlinks to next states*

$$\begin{tikzcd}
 & S_1 \arrow[rd, "\underline{3}/c"] & & \\
S_0 \arrow[ru, "\underline{1}/a"] \arrow[rd, "\underline{2}/b"'] & & S_3 \arrow[r] & {} \\
 & S_2 \arrow[ru, "\underline{3}/c"] \arrow[rd, "\underline{4}/d"] & & \\
 & & S_4 \arrow[r] & {}
\end{tikzcd}$$

Overstock.com
slandis@overstock.com

Resources

- A Resource should be a fixed target of a URI
- Is semantic: "Today's weather in Park City"
- The URI-to-Resource mapping shouldn't change, but the representation can
- Resources may map to multiple representations, called *variants*
  - *Example: png, gif, jpg are variant representations of an image*
  - *Content negotiation selects the best variant*

Overstock.com
slandis@overstock.com

Uniform Interface

- Supports the constraints of *Client/Server*, and *Layered* architectural styles
- Resources are manipulated by HTTP *methods*
  - GET – retrieve a resource
  - PUT – create a resource
  - POST – update (create if necessary) a resource
  - DELETE – delete a resource

Overstock.com
slandis@overstock.com

Interoperability on a Global Scale

- REST advocates (and constrains the use of) existing Web standards:
  - **URI** – how resources are named and referenced
  - **Methods** – how resources are manipulated
  - **HTML, XML, GIF, etc** – how resources are represented
  - **Media types (text/plain, etc)** – metadata for representations

Overstock.com
slandis@overstock.com

Cachability

- Reduces latency, increases scalability through reduced bandwidth utilization
- REST architectural constraints allow caches to be injected anywhere in the application
- A cache can return copy in response to a GET, therefore prefer GET over POST

Overstock.com
slandis@overstock.com

Principles of REST

- Some principles are still debated
- Do what makes sense for your application, but be conscious of the tradeoffs

“You're pirates. Hang the code, and hang the rules. They're more like guidelines anyway.”
– *Elizabeth (Pirates of the Caribbean)*

Overstock.com
slandis@overstock.com

Principles of REST

- URIs refer to resources, not representations
  - ✓ `www.overstock.com/home+and+garden`
  - ✗ `www.overstock.com/home+garden.html`
- Resources are nouns, not verbs
- GET never has side effects, and anything that has no side effects should use GET
- Use links in responses enable state transfer

Adapted from Roger L. Costello

Overstock.com
slandis@overstock.com

Principles of REST

- URI “/” means *parent-child* or *whole-part relationship*
- Avoid query strings in URIs (debatable)
  - ✗ `www.overstock.com/products/id=123`
  - ✓ `www.overstock.com/products/123`
  - In the later case, the relationship is clear and can be extended for subresources
- Provide data to clients via *gradual unfolding*

Overstock.com
slandis@overstock.com

References

- Roy Fielding, *Architectural Styles and the Design of Network-based Software Architectures*
- Roger L. Costello, `www.xfront.com`
- Paul Prescod, `www.prescod.net/rest`
- Tim Berners-Lee, *Universal Resource Identifiers -- Axioms of Web Architecture,* `www.w3.org/DesignIssues/Axioms.html`
- Hao He, *Implementing REST Web Services: Best Practices and Guidelines,* `www.xml.com/pub/a/2004/08/11/rest.html`

Overstock.com
slandis@overstock.com

Restlet Project

Overstock.com
slandis@overstock.com

What is Restlet?

- An open source REST framework for Java
- A good mapping of REST principles
- Founded by Jérôme Louvel, Noelios Consulting, Paris, France
  `www.restlet.org`
- Built in response to:
  - Need for a simple, RESTful web application framework
  - Servlet limitations

Overstock.com
slandis@overstock.com

Restlet Programming

Overstock.com
slandis@overstock.com

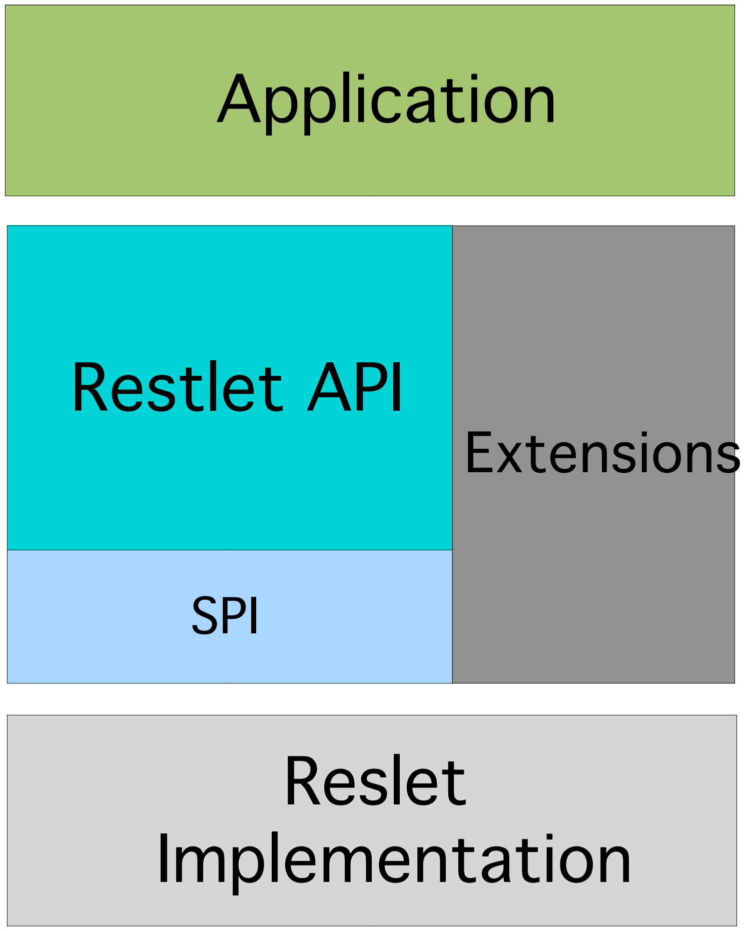
Restlet Framework

Restlet API – Supports REST call handling

Extensions – *For integrating external technologies (JDBC, JSON, alternate containers, connectors, template engines, etc.)*

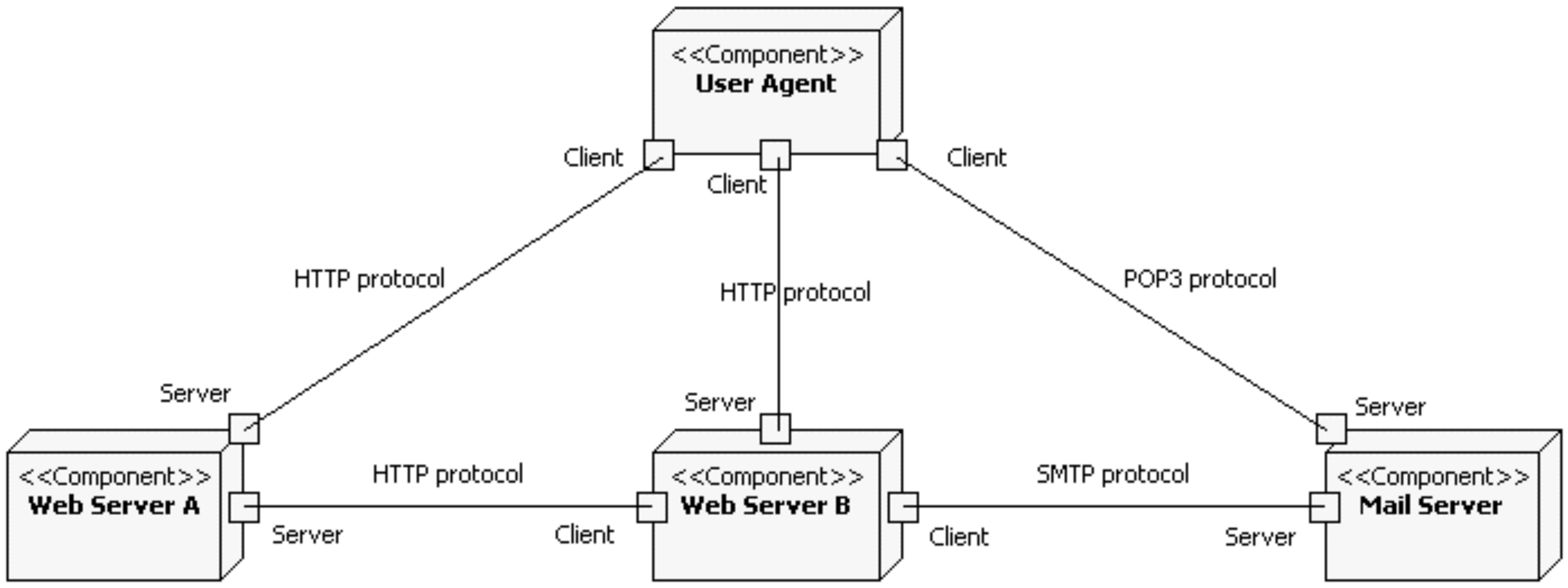
SPI – *Plugin point for alternate implementations*

Restlet Implementation – *Currently just Noelios Engine*



Overstock.com
slandis@overstock.com

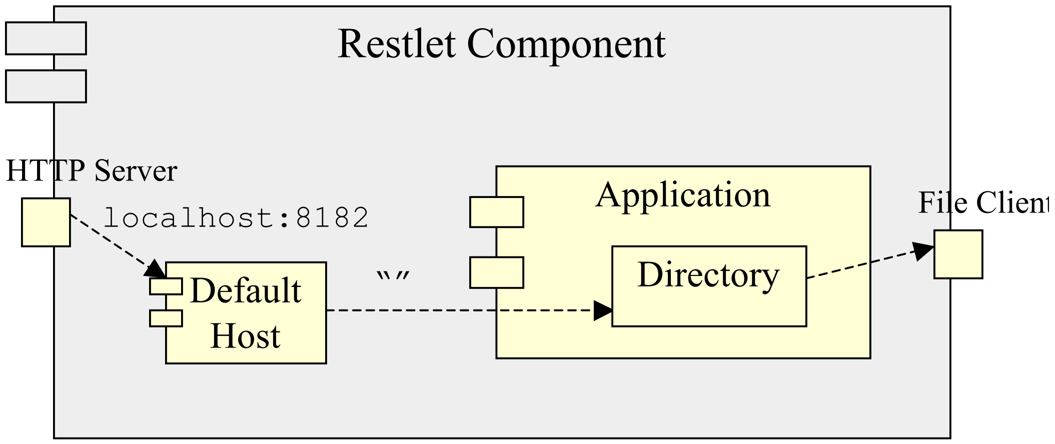
A REST Architecture



Overstock.com
slandis@overstock.com

File Browsing Example

- `Component` contains `VirtualHosts`, `Application`s and `Server` and `Client` connectors
- Default Host is 'built-in'
- Notice the beginnings of a pipes-and-filters architecture within a client-server architecture



Overstock.com
slandis@overstock.com

Application Class

- Contains your “application” logic
- Contains useful services that can be overridden, such as:
  - connectorService
  - decoderService
  - statusService

Overstock.com
slandis@overstock.com

Directory Class

- Finder of file system resources
- Automatic content negotiation similar to Apache HTTP server
  - Selects best representation based on
    - available variants
    - client capabilities and preferences

Overstock.com
slandis@overstock.com

FileServer Example

```
public class FileServer implements Constants {

  public static void main(String[] args) throws Exception {
    Component component = new Component();
    component.getServers().add(Protocol.HTTP, 8182);
    component.getClients().add(Protocol.FILE);

    Application application = new Application(component.getContext()) {
        @Override public Restlet createRoot() {
          Directory directory = new Directory(getContext(),
                                              "file://" + ROOT);
          directory.setListingAllowed(true);
          directory.setDeeplyAccessible(true);
          return directory;
        }
    };

    component.getDefaultHost().attach("", application);
    component.start();
  }
}
```

Overstock.com
slandis@overstock.com

VirtualHost Class

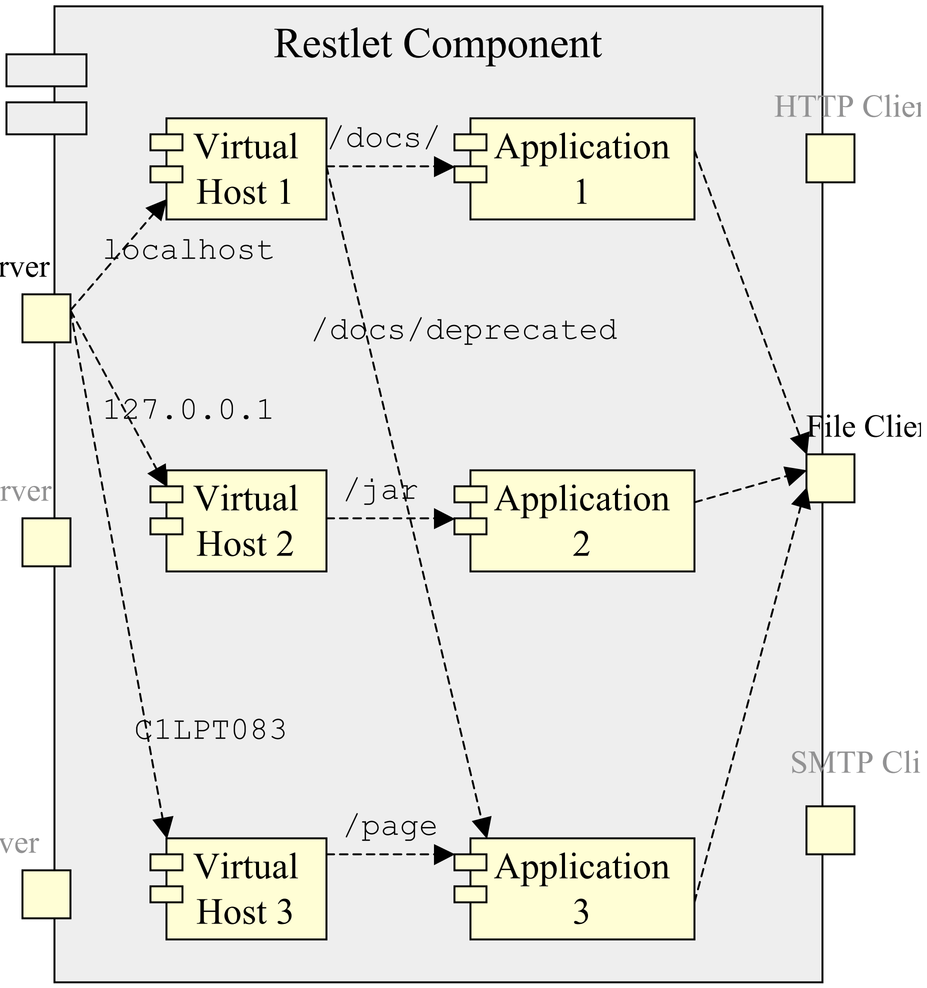
- `Router` of calls from `Server` connectors to `Restlet`s; typically an `Application`
- Defined along three properties:
  - Request's "hostRef"
  - Request's "resourceRef"
  - Response's "serverInfo"
- Host multiple applications in a single JVM
  - Same IP address shared by several domain names
  - Same domain name load-balanced across several IP addresses

Overstock.com
slandis@overstock.com

Virtual Hosts Example

- A `VirtualHost` routes requests to `Application`s by regular expression matching

- Grey items are included for illustration



Overstock.com
slandis@overstock.com

VirtualHost Example

```
public class VirtualHostServer implements Constants {

  public static void main(String[] args) throws Exception {
    Component component = new Component();
    component.getServers().add(Protocol.HTTP, 8182);
    component.getClients().add(Protocol.FILE);

    VirtualHost vh1 = new VirtualHost(component.getContext());
    // Host names must be distingushed and not made up.
    vh1.setHostDomain("localhost");

    Application application1 =
           new Application(component.getContext()) {
      @Override public Restlet createRoot() {
        Directory directory = new Directory(getContext(),
                                                    DOC_URI);
        return directory;
      }
    };
```

Overstock.com
slandis@overstock.com

...Continued

```
VirtualHost vh2 = new VirtualHost(component.getContext());
vh2.setHostDomain("127.0.0.1");
Application application2 =
         new Application(component.getContext()) {
  @Override public Restlet createRoot() {
    Restlet jarRestlet = new Restlet(getContext()) {
      @Override public void handle(Request request,
                                   Response response) {
        File file = new File(JAR_PATH);
        FileRepresentation frep =
               new FileRepresentation(file,
               MediaType.APPLICATION_JAVA_ARCHIVE, 1000);
        response.setEntity(frep);
        response.setStatus(Status.SUCCESS_OK);
      }
    };
    return jarRestlet;
  }
};
```

Overstock.com
slandis@overstock.com

...Continued

```
VirtualHost vh3 = new VirtualHost(component.getContext());
vh3.setHostDomain("C1LPT083");
Application application3 =
             new Application(component.getContext()) {
  @Override public Restlet createRoot() {
    Restlet pageRestlet = new Restlet(getContext()) {
      @Override public void handle(Request request,
                                   Response response) {
        File file = new File(PAGE_PATH);
        FileRepresentation frep =
          new FileRepresentation(file, MediaType.TEXT_HTML,
                                  1000);
        response.setEntity(frep);
        response.setStatus(Status.SUCCESS_OK);
      }
    };
    return pageRestlet;
  }
};
```

Overstock.com
slandis@overstock.com

...Continued

```
      vh1.attach("/docs/", application1);
      vh1.attach("/docs/deprecated",
                 application3);
      component.getHosts().add(vh1);
      vh2.attach("/jar", application2);
      component.getHosts().add(vh2);
      vh3.attach("/page", application3);
      component.getHosts().add(vh3);
      component.start();
    }
}
```

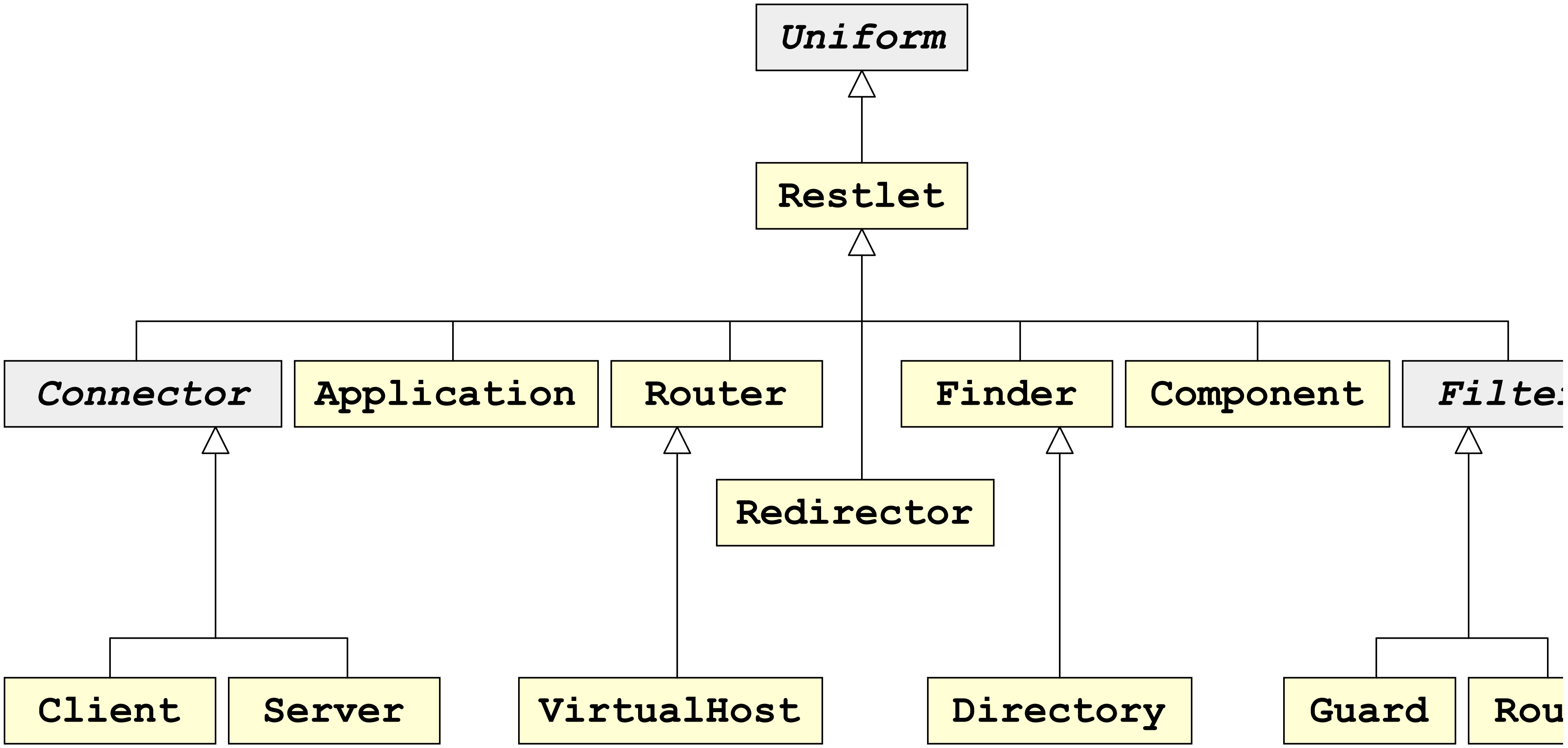
Overstock.com
slandis@overstock.com

Restlet Class

- Uniform interface class
  - Get, Put, Post, Delete
- Context
- Life cycle support
- Its subclasses implement specific ways to process calls

Overstock.com
slandis@overstock.com

Restlet Class



Overstock.com
slandis@overstock.com

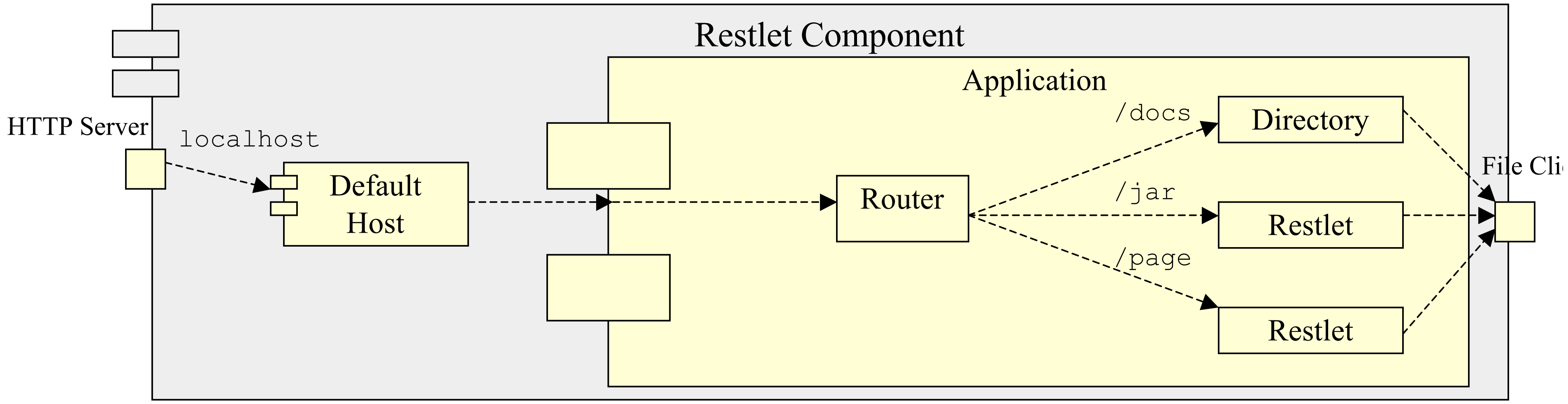
Router Class

- `Restlet` for routing calls to one of the attached routes (e.g., to another `Restlet`)
- `attach(pattern, Restlet)`
  - Creates a route based on URI patterns matching the beginning of a the resource reference's remaining part

Overstock.com
slandis@overstock.com

Example Application

- Same host/port
- Routes based on URI



Overstock.com
slandis@overstock.com

Router Example

```
public class RouterServer implements Constants {
  public static void main(String[] args) throws Exception {
    // Initialize connectors as before...

    Application application =
                new Application(component.getContext()) {
      @Override public Restlet createRoot() {
        Router router = new Router(getContext());
        // Create the Restlets as before...
        router.attach("/docs", directory);
        router.attach("/jar", jarRestlet);
        router.attach("/page", pageRestlet);
        return router;
      }
    };

    component.getDefaultHost().attach("", application);
    component.start();
  }
}
```

Overstock.com
slandis@overstock.com

Router URI Patterns

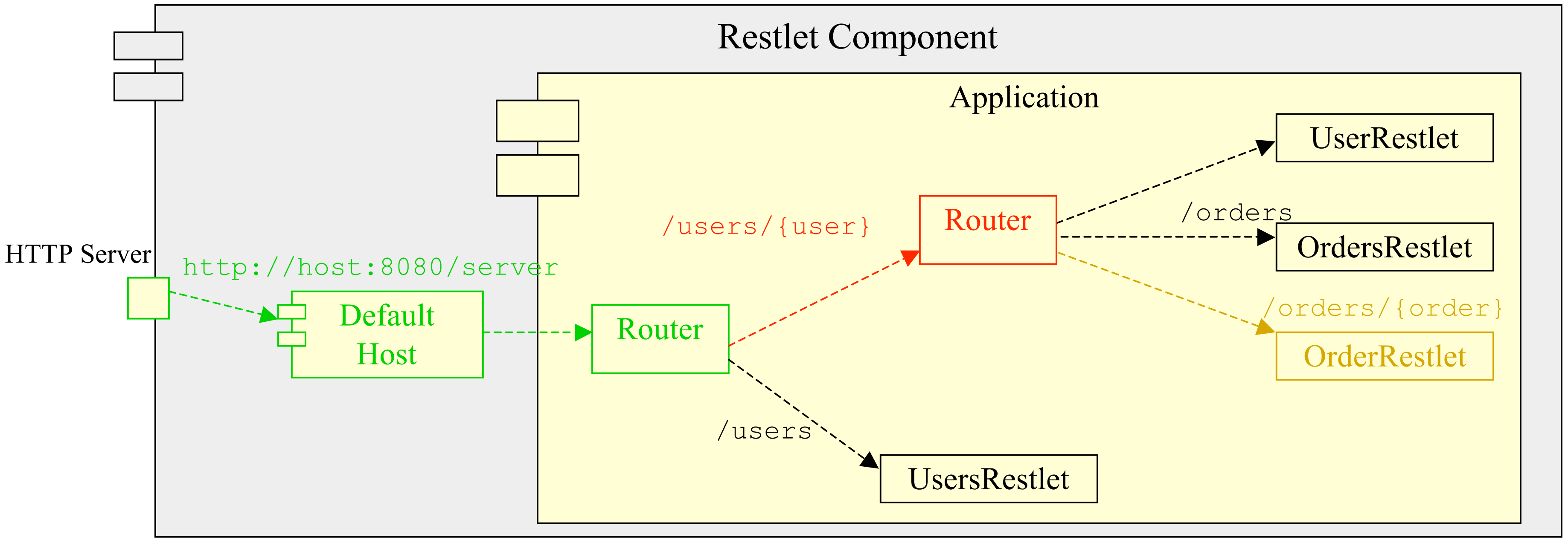
- URI Template Spec for variables
- Example URI patterns:
  - `/docs/` to display static files
  - `/users/{user}` to display a user's account
  - `/users/{user}/orders` to display the orders of a particular user
  - `/users/{user}/orders/{order}` to display a specific order

Overstock.com
slandis@overstock.com

URI Routing

`http://host:8080/server/users/123/orders/456`

- Router sees: `/users/123/orders/456`
- Router sees: `/orders/456`



Overstock.com
slandis@overstock.com

Advanced Router

- A `Route` can compute a score for each call depending on various criteria
- Several routing modes are supported:
  - Best match (default)
  - Round robin
  - Random match
  - First match
  - Last match
  - Custom

Overstock.com
slandis@overstock.com

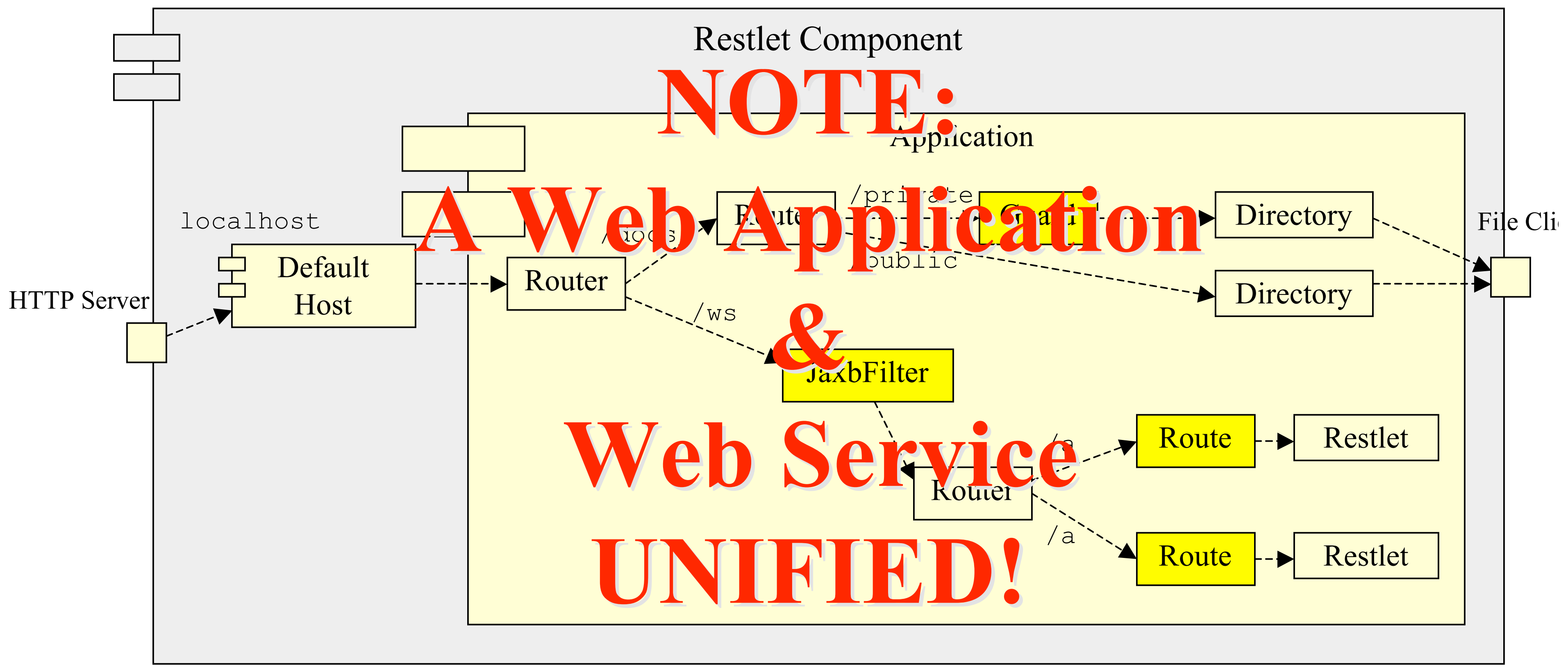
Round Robin Example

```
@Override public Restlet createRoot() {
  Router router = new Router(getContext());
  Restlet restlet1 = new Restlet(getContext()) {
    @Override public void handle(Request request,
                                 Response response) {
      StringRepresentation rep =
                new StringRepresentation("Restlet 1");
      response.setEntity(rep);
      response.setStatus(Status.SUCCESS_OK);
    }
  };

  Restlet restlet2 = new Restlet(getContext()) { ...};

  router.setRoutingMode(Router.NEXT);
  router.attach("", restlet1);
  router.attach("", restlet2);
  return router;
}
```

Overstock.com
slandis@overstock.com

Filter Class

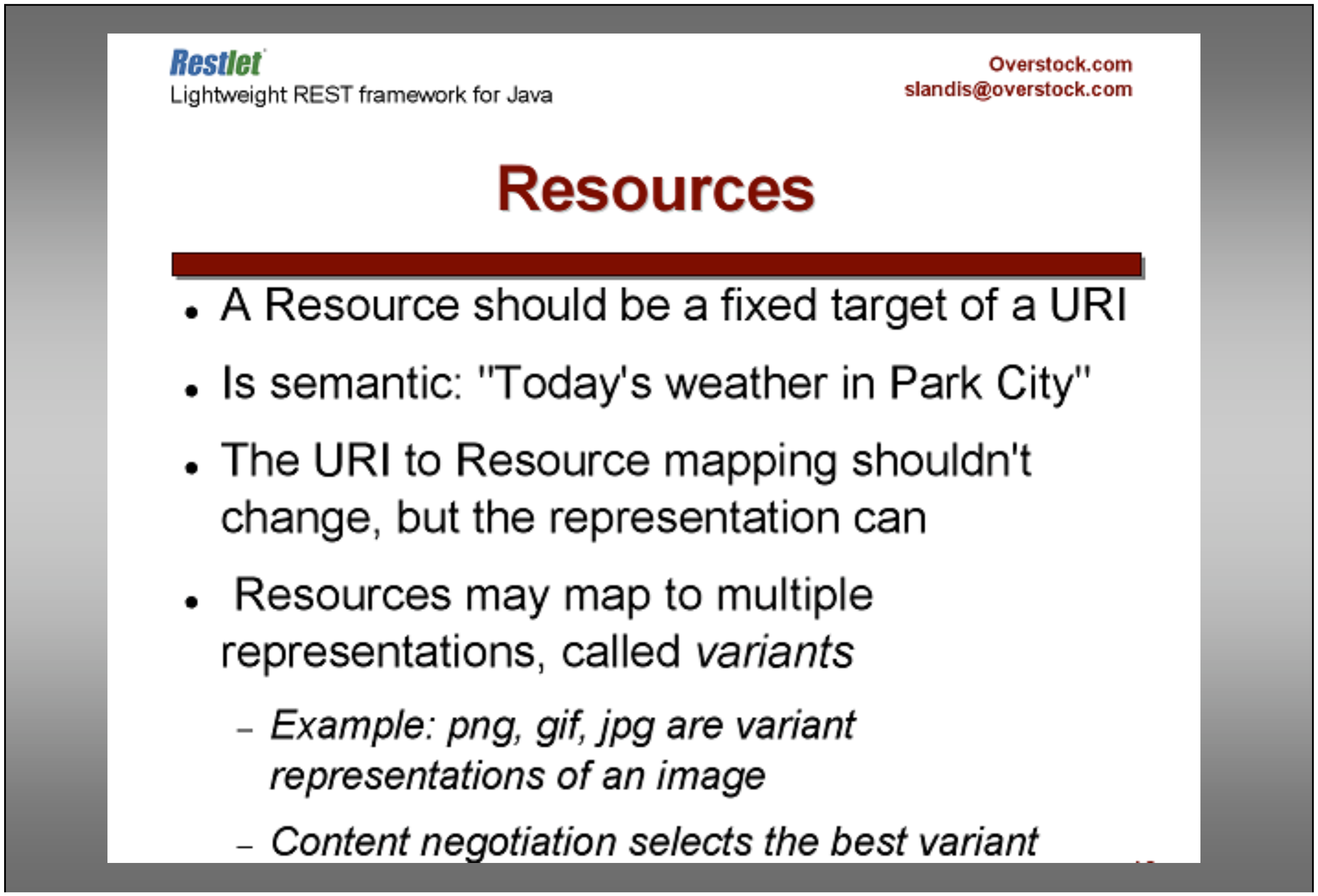
- Impose before/after handling in call flow



Overstock.com
slandis@overstock.com

Resource

- Remember this?



Restlet
Lightweight REST framework for Java

Overstock.com
slandis@overstock.com

Resources

- A Resource should be a fixed target of a URI
- Is semantic: "Today's weather in Park City"
- The URI to Resource mapping shouldn't change, but the representation can
- Resources may map to multiple representations, called *variants*
  - *Example: png, gif, jpg are variant representations of an image*
  - *Content negotiation selects the best variant*

Overstock.com
slandis@overstock.com

Resource Class

- Typically created by a `Finder`
- Selects a variant `Representation`
- A final handler of calls in the pipeline
- Not shared between calls; can be thread-unsafe
- Where the RESTful view of your Web application can be integrated with domain objects
  - Databases, beans, other services, etc.
- By default, ony the GET method is enabled

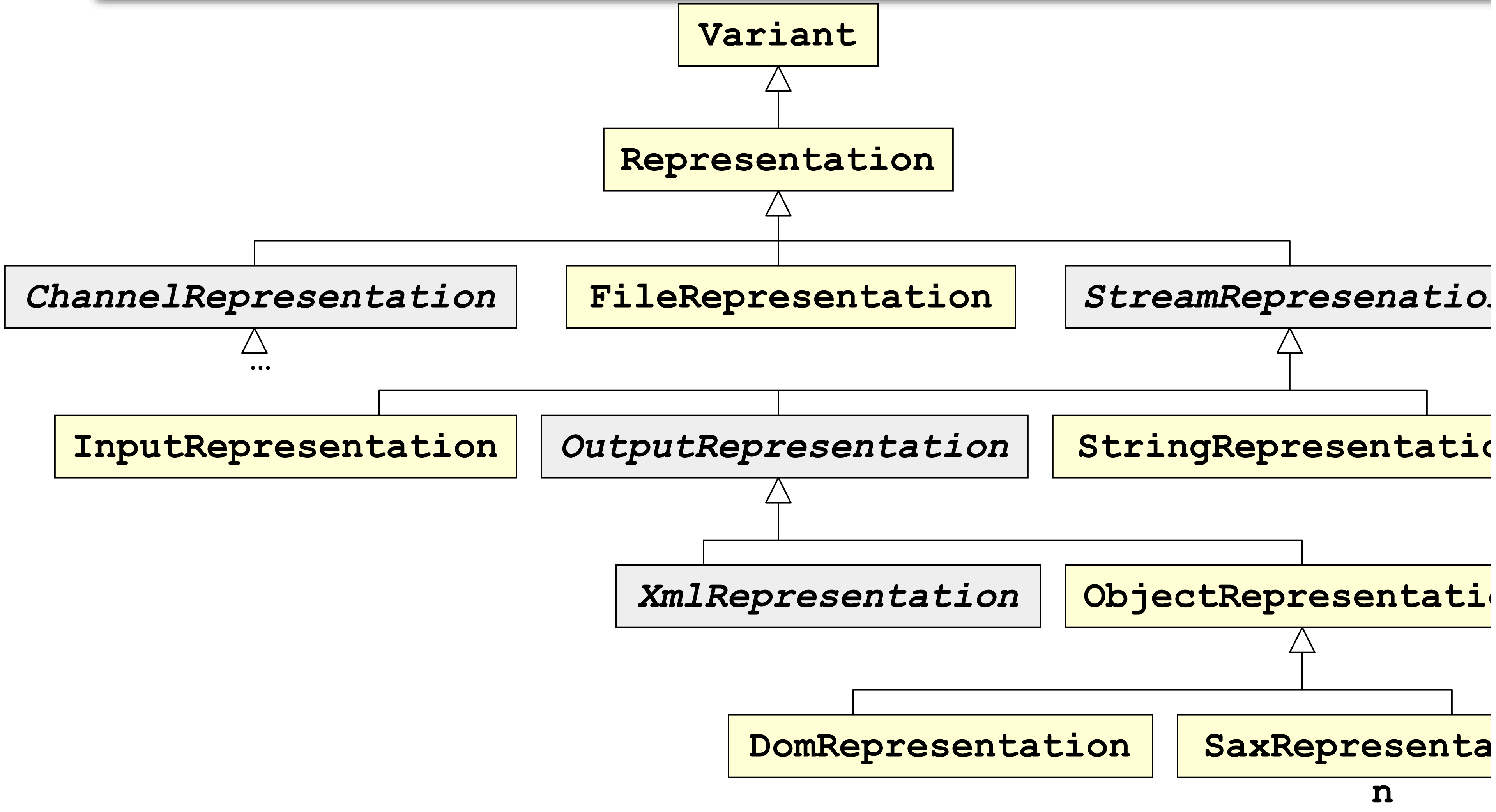
Overstock.com
slandis@overstock.com

Using Resource

- Override REST methods you support: `post()`, `put()`, `delete()`
- Override the matching `allow*()` methods
- Optionally override `handle*()` method for custom content negotiation
- Restlet calls are dynamically dispatched to the `handle*()` methods via introspection
- To support a custom MOVE method
  - add `handleMove()`

Overstock.com
slandis@overstock.com

Representation Class



Overstock.com
slandis@overstock.com

Bookmark Example

```
@Override public Restlet createRoot() {
  Router router = new Router(getContext());
    router.attach("/users/{username}",
                  UserResource.class);
    router.attach("/users/{username}/bookmarks",
                  BookmarksResource.class);
    Route uriRoute =
      router.attach("/users/{username}/bookmarks/{URI}",
                    BookmarkResource.class);
    uriRoute.getTemplate().getVariables().put("URI",
                new Variable(Variable.TYPE_URI_ALL));

        return router;
    }
```

Overstock.com
slandis@overstock.com

UserResource.java

```
public class UserResource extends Resource {
...
  public UserResource(Context context, Request request,
                      Response response) {
    super(context, request, response);
    this.userName = (String)
                    request.getAttributes().get("username");
...
    this.user = findUser();
    if (user != null) {
      getVariants().add(new Variant(MediaType.TEXT_PLAIN));
    }
  }

  @Override public boolean allowDelete() { return true; }

  @Override public boolean allowPut() { return true;}
...
```

Overstock.com
slandis@overstock.com

UserResource.java (continued)

```
@Override
public Representation getRepresentation(Variant variant) {
        Representation result = null;
  if (variant.getMediaType().equals(MediaType.TEXT_PLAIN)) {
    StringBuilder sb = new StringBuilder();
    sb.append("------------\n");
    sb.append("User details\n");
    sb.append("------------\n\n");
    sb.append("Name: ")
              .append(this.user.getFullName()).append('\n');
    sb.append("Email: ")
              .append(this.user.getEmail()).append('\n');
    result = new StringRepresentation(sb);
  }
  return result;
}

@Override public void put(Representation entity) {
   // Creates a user in a database ...
}
```

Overstock.com
slandis@overstock.com

Deployment Options

Overstock.com
slandis@overstock.com

Many Ways to Deploy

- Deploy as a jar file
- Any Servlet compliant container
  - Tomcat, Jetty
- Native service using Java Service Wrapper

Overstock.com
slandis@overstock.com

Restlet & Other Technologies

Overstock.com
slandis@overstock.com

Plays Well With Others

- Various Connectors
  - HTTPS, AJP, Apache HTTP Client, SMTP[S], JDBC, FILE
- Lots of Representations
  - DOM, SAX, XPath, XSLT
  - Template Engine: Velocity, FreeMarker
  - NIO, Apache Upload
- Easy 3rd party integration
  - Eg., Struts, Spring, Hibernate, Acegi, Seam, etc

Overstock.com
slandis@overstock.com

Overstock.com Experience

Overstock.com
slandis@overstock.com

Restlet @ Overstock.com

- Created in-house Web Services framework
  - XSD for requests and responses
  - JAXB `Filter` converts between XML and our object model
- About 6 active developers, more to come
- 3 projects in production, 3 in DEV or QA
- Easy to learn, quick to code, reliable & fast

OVERSTOCK IS HIRING!! TALK TO ME

Overstock.com
slandis@overstock.com

Not Covered

Overstock.com
slandis@overstock.com

Lot’s of Other Things

- Finders
- Restlet on the client
- Redirection
- Guards
- NIO
- Logging and error handling
- JSR 311 – REST Annotations

Overstock.com
slandis@overstock.com

Q & A

Overstock.com
slandis@overstock.com

Resource Content Negotiation

Overstock.com
slandis@overstock.com

GET vs. POST

- The result of a GET is to return a representation of the resource
- The result of a POST is to post something to a processing resource, which may create a new subordinate resource
  - In general, the response entity of the POST will describe the status of the method execution, if it succeeded or if it failed

Overstock.com
slandis@overstock.com

Returning a Represenation from POST

- Example: a complex search request
- Reasons for bending the REST style:
  - URI length overflow – search requests can be very large
  - Information hiding – keep information off the URI
  - Tradition/legacy/migration – “this is how we've done it before”

Overstock.com
slandis@overstock.com

Returning a Represenation from POST

- When POST modifies the target resource, and you want to return the best representation, do this at the end of the `post()` method:

```
getResponse().setEntity(
    getPreferredRepresentation());
```

Overstock.com
slandis@overstock.com

Returning a Represenation from POST

- To directly return the representation of the created resource, instantiate a new Resource, and manually call:

```
Resource res = new
   MyDelegateResource(...);
Representation rep =
   res.getPreferredRepresentation();
getResponse().setEntity(rep);
```

Overstock.com
slandis@overstock.com

Returning a Represenation from POST

- Finally, if you don't need to take advantage of content negotiation, you can directly set the response entity manually in your `post()` method:

```
getResponse()
   .setEntity(myJaxbRepresentation);
```

Overstock.com
slandis@overstock.com

Multi-step Strategies

- POST `/queries`
  - Pass the query document in the request entity
  - Return a status document and a redirectRef with the created query URI
- GET `/queries/264794`
  - Returns the result of the query
  - Idempotent, can be cached
- DELETE `/queries/264794`
  - Cache auto-deletes old queries

Overstock.com
slandis@overstock.com

Content Negotiation

Overstock.com
slandis@overstock.com

Types of Content Negotiation

- **Server-driven** - Server picks representation from prior knowlege of client, or uses HTTP header information
- **Client-driven** - Client requests; server returns list of representations; client picks one...requires 2 calls
- **Proxy-driven** - Proxy chooses from a list returned by server, using client preferences
- **URI-specified** – Uses the query string