RESTful Spring

Brian Sletten

brian@bosatsu.net

Speaker Qualifications

- Independent Software Consultant
- 13 years of software experience
- Work in Semantic Web, AOP, Grid Computing, P2P and security consulting spaces
- Built Spring-based application for large customer

Agenda

- Why Are We Here?
- REST Overview
- Spring w/ Restlets
- Securing RESTful Spring
- Spring w/ NetKernel
- Conclusion

Why Are We Here?

WS-Tenacity

SOA = WSDL + SOAP + UDDI

WS-Tenacity

SOA = WSDL + SOAP + UDDI!!!!

WS-Complexity

- Real Complexity
  - Hard things hard
- Artificial Complexity
  - Easy things are (still!) hard

WS-Inoperability

- SOAP
  - Message-oriented request
  - Mixes verb space and content space (No nouns!)
- WSDL
  - What You See Is What You Get
- UDDI
  - Published metadata about service
  - Simultaneously complex and limited

WS-Insecurity

- Conventional Web Service bigots have the gall to claim the moral highground on security
  - Complexity is the bane of security
  - Routing Around Network Architecture
  - Impotent Intermediaries

WS-Flexibility

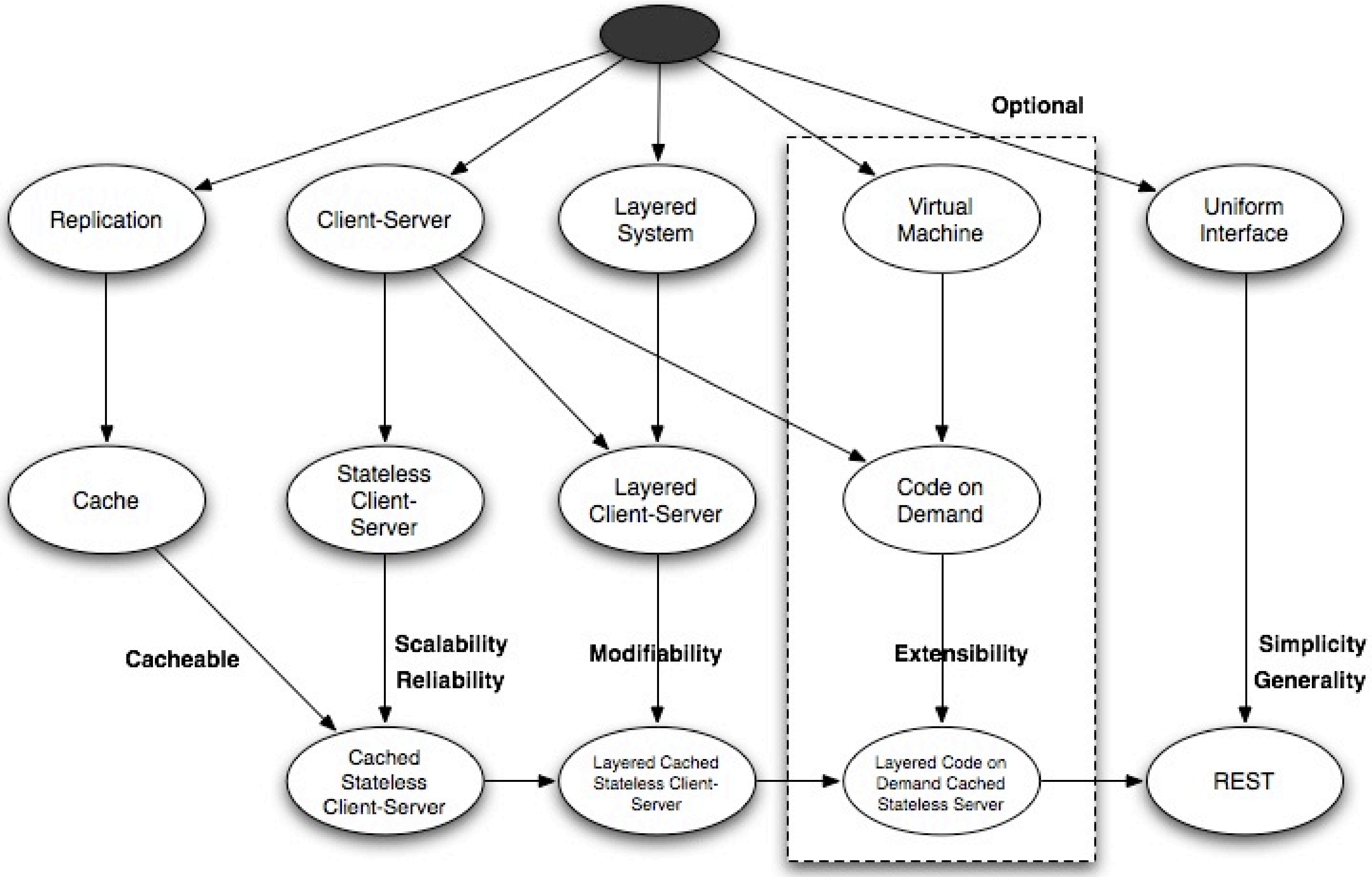
- If you build it…
  - Amazon supports both SOAP-based and RESTful Web Services
  - Developers have spoken (80-90% prefer REST)
- Architectural improvements when you support both styles via the Command Pattern

REST Overview

REST is History

- REST (REpresentational State Transfer)
- Based largely on Roy Fielding's Ph.D. thesis
- Architectural style designed to promote
  - Performance
  - Scalability
  - Generality
  - Simplicity
  - Modifiability

REST Architecture



From Roy Fielding's Thesis

RESTful Web Services

- Putting the “Web” in Web Services
- Reusing existing technologies
- Simple things easy, hard thing possible
  - Can layer on complexity as necessary
- Nothing necessarily to buy

Comparison to SOAP

- Separation of Concerns
  - Noun space, Verb space, Content space
- Identifiable resources
- Identifiable requests
- Constrained semantics
- Empowered intermediaries
- Contracts not required (but possible)

Noun Space

- Resources are an abstraction for what is available
  - Files
  - Generated Content
  - Computational Results
  - Concepts/Organizations/People
- What comes back can change over time
  - Think about today’s /. Page

What’s In a Name?

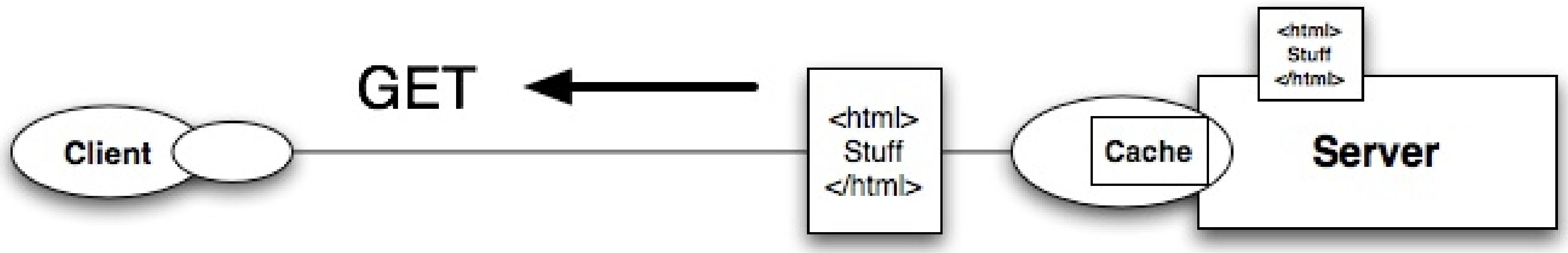
- Resources are referenced via Identifiers
  - http://www.bosatsu.net
  - http://www.菩薩.net
  - urn:isbn:0977616665
- Dereferencing URIs is orthogonal to transferring content

Verb Space

- Constrained semantics for acting upon resources
- Traditionally
  - GET
  - POST
  - PUT
  - DELETE
- Allows intermediaries to apply security/ caching policies

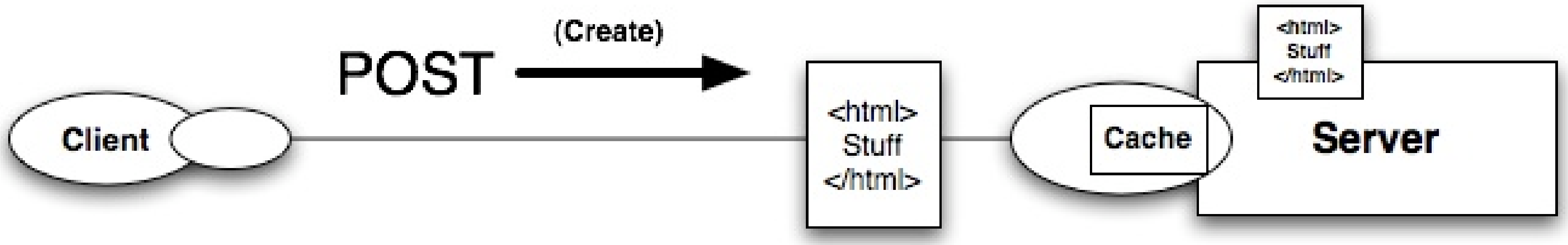
GET

- Consequence-free (idempotent) request for a resource



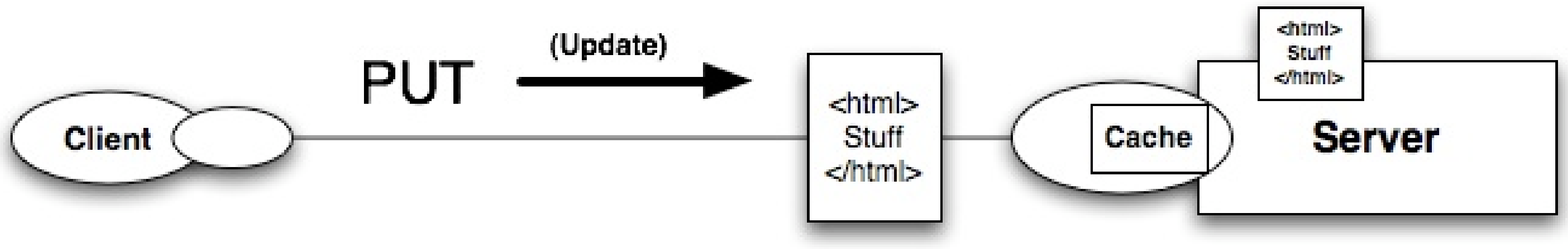
POST

- Transfer all or some portion of a resource to a processing engine on the server
- Create/Update depending on context



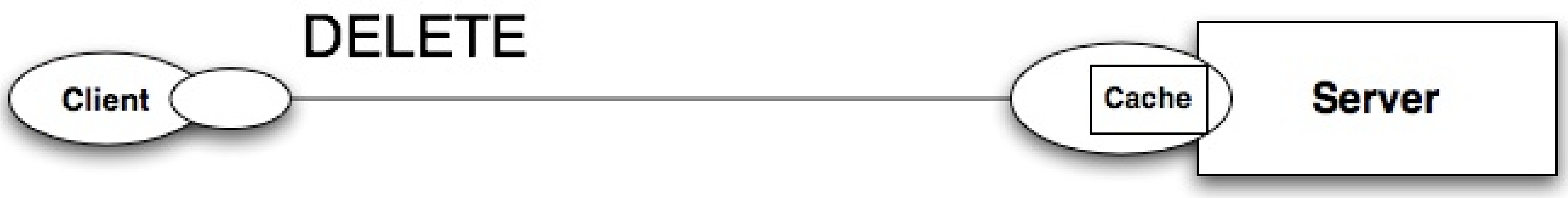
PUT

- Create a resource if it doesn't exist or overwrite it if it does



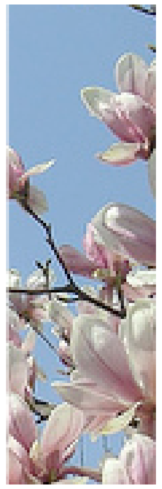
Delete

- Remove a resource from a server



Content Space

- Byte streams annotated with metadata
  - Last-modified
  - MIME-type
- No deeper structure specified
  - Clients have freedom/responsibility to know how to interpret
  - Clients can negotiate content form with server
  - Resources are sent to/from client as a concrete Representation



Spring w/ Restlets

Why not WebLogic? Geronimo? Tomcat?

- Possible to expose RESTful services through these containers
- We lose many of the architectural benefits by tying ourselves to non-REST-oriented containers
  - URLs + verbs + resource abstraction provides benefits
  - URLs != scalability

Why not Spring MVC WebFlow?

- Again, you certainly are encouraged to build RESTful APIs w/ these
- However, tied to Servlet API
  - Assumes synchronous I/O in request mechanism
  - Not as easy to support other transports

Restlet API

- Simple Java API for providing/consuming RESTful Web Services
- Replacement for Servlet API to avoid I/O and transport limitations
- Container independent
- Object model for RESTful concepts
- Useful to help define RESTful APIs

Restlet Features

- Blocking/Non-blocking IO
- Representations
  - String, XML, JSON, Freemarker templates
- Transports/Protocols
  - HTTP/HTTPS, SMTP, JDBC, FILE, AJP
- Filters
- Spring integration!

Simple Server

```
package net.bosatsu.spring;

import org.restlet.Restlet;
import org.restlet.Server;
import org.restlet.data.MediaType;
import org.restlet.data.Protocol;
import org.restlet.data.Request;
import org.restlet.data.Response;

public class SimpleServer {
    public static void main( String [] args ) throws Exception {
        Restlet restlet = new Restlet() {
           public void handle(Request request, Response response)
           {
              response.setEntity("Hello, Florida!", MediaType.TEXT_PLAIN);
           }
        };

        new Server(Protocol.HTTP, 8183, restlet).start();
    }
}
```

Client Side Support!

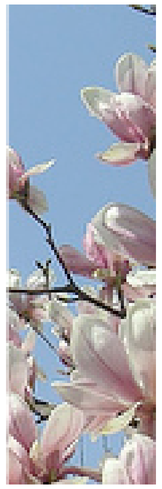
```
package net.bosatsu.spring;

import org.restlet.Client;
import org.restlet.data.Protocol;

public class SimpleClient {
    public static void main( String [] args ) throws Exception {
        Client client = new Client(Protocol.HTTP);
        client.get("http://localhost:8183").getEntity().write(System.out);
    }
}
```

Spring Restlet Examples

Securing RESTful Spring



Basic/Digest Auth

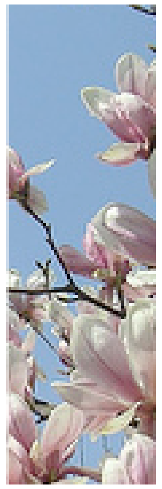
- Same mechanisms used by web servers can be used to protect access to simple RESTful APIs
- Simple and easy
- Assumes trusted sources and should leverage SSL

ACEGI

- Same security framework that protects normal Spring apps can be used to secure Spring-based RESTful APIs
- Interception-based checks can be as elaborate as need be

Securing RESTful Spring Examples

Spring w/ NetKernel



NetKernel

- Java-based microkernel architecture built around the ideas of REST, Unix pipes and SOA
- Tremendously productive and scalable architecture
- Homogenizes everything into URI-addressable features
- Advanced features allow us to improve upon our Spring-based RESTful architecture

Spring NetKernel Examples

Conclusion

REST is AN Answer

- For architectural styles that support it, REST allows systems to be simple but complete
- It is possible to layer on extra complexity as needed
- Promotes separation of noun, verb and content spaces for simplicity and extensibility
- Systems built on principles of REST demonstrate great scalability
- Some places it is not the right answer

References

Roy Fielding's Thesis	http://tinyurl.com/cvamh
Restlet	http://www.restlet.org
NetKernel	http://www.1060.org
ACEGI	http://www.acegisecurity.org/
RESTWiki	http://rest.blueoxen.net
Slides	http://www.bosatsu.net/talks/RESTful-Spring.pdf
Examples	http://www.bosatsu.net/talks/examples/RESTful-Spring-Examples.zip
Flickr Picture	http://www.flickr.com/photos/ernstl/132402663/

Questions?

brian@bosatsu.net

菩薩相談