Java is a trademark of Sun Microsystems, Inc.

JavaOne℠

RESTful Protocol Buffers

Matt O'Keefe/Alex Antonov

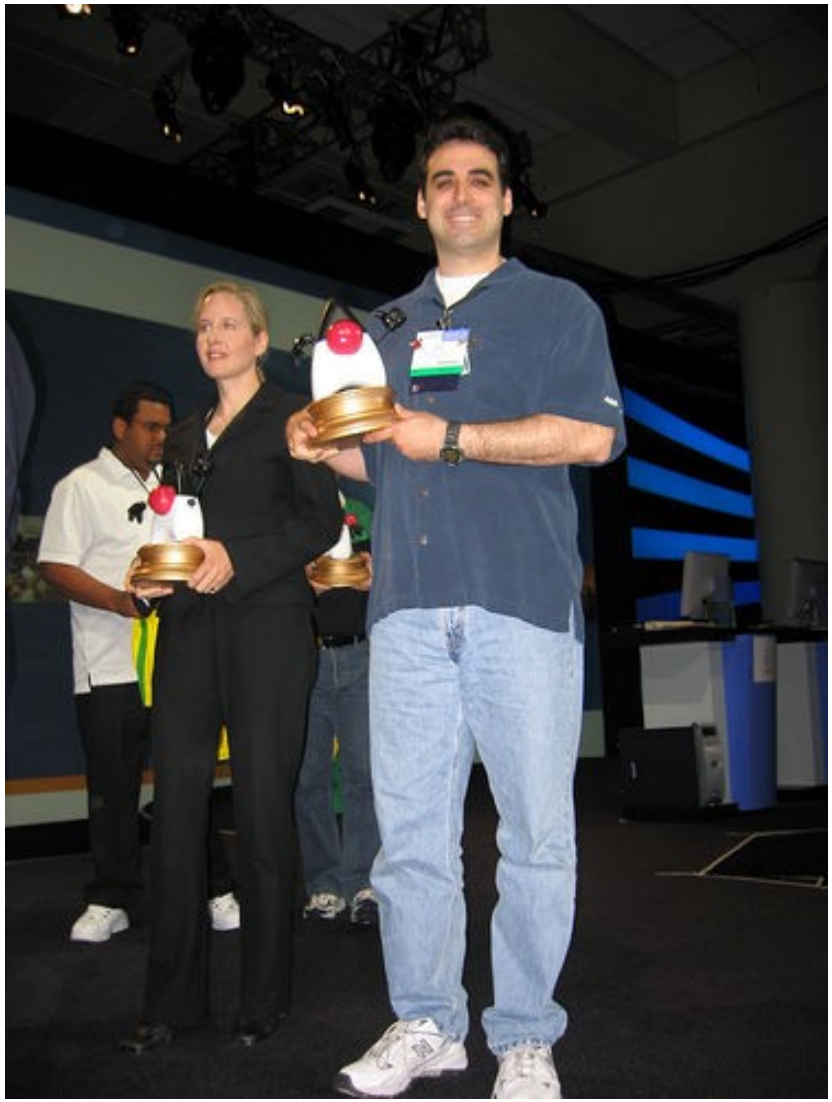
Sears Holdings/Orbitz Worldwide

Agenda

- Problem Statement
- Why REST?
- Why Protocol Buffers?
- Implementation Details
  - Plumbing
  - Data Modeling
- Testing and Release Management
- Results

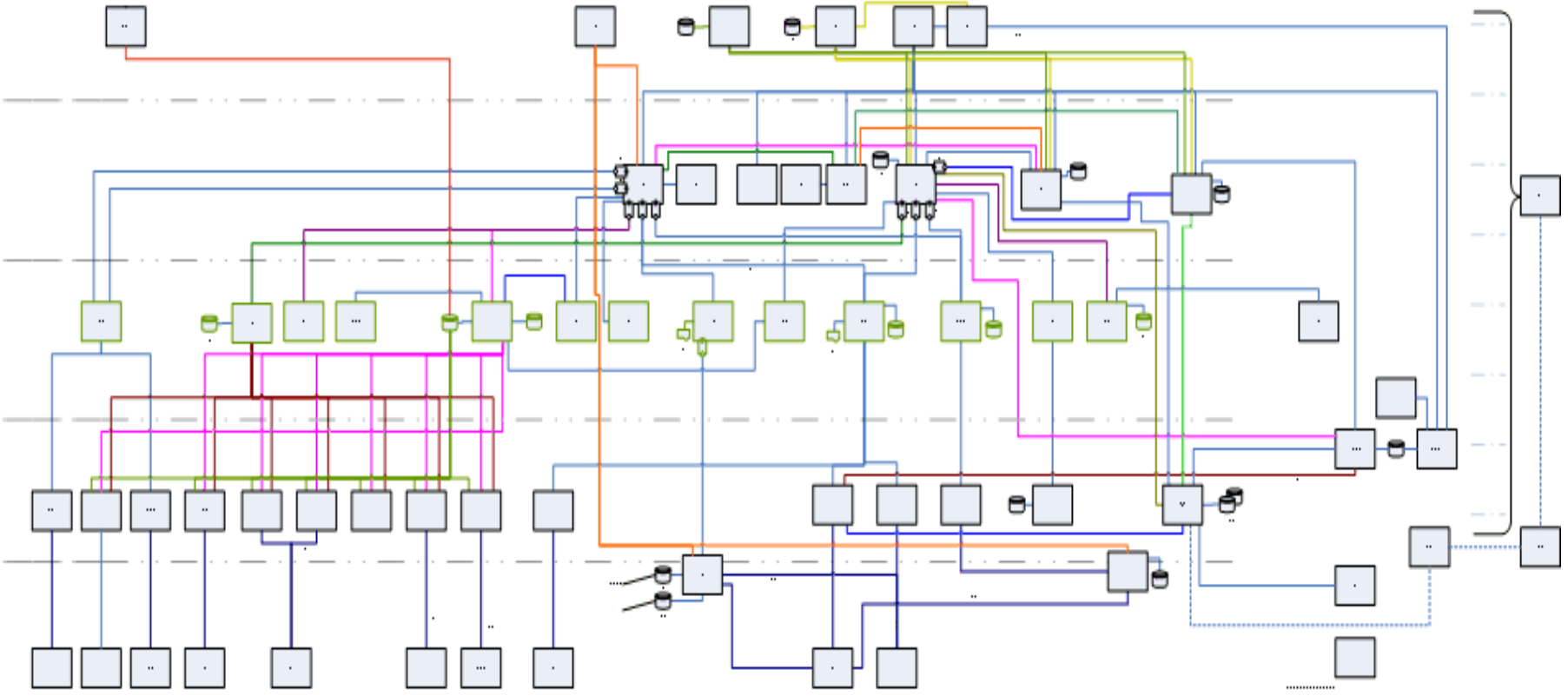
A Little History – JavaOne 2004

- "Miss Con-Jini-ality" Dukie award winner
- SOA before SOA was a buzzword
- Dynamic Service Registration and Discovery
- Inexpensive EJB Alternative



Problem Statement

- Management of Shared, Versioned Dependencies with Serializable Objects is Difficult



Architectural Principles
Our Decision-making Framework

- Loosely Coupled, Contractually Obligated
  - Systems should have few dependencies with freedom to evolve independently, but where dependencies and contracts do exist, they must be rigidly enforced.
- Release Early, Release Often
  - Software should be released when ready, and of high quality. It should be released in small iterations, and frequently, to keep the scope of change to a minimum.

Agenda

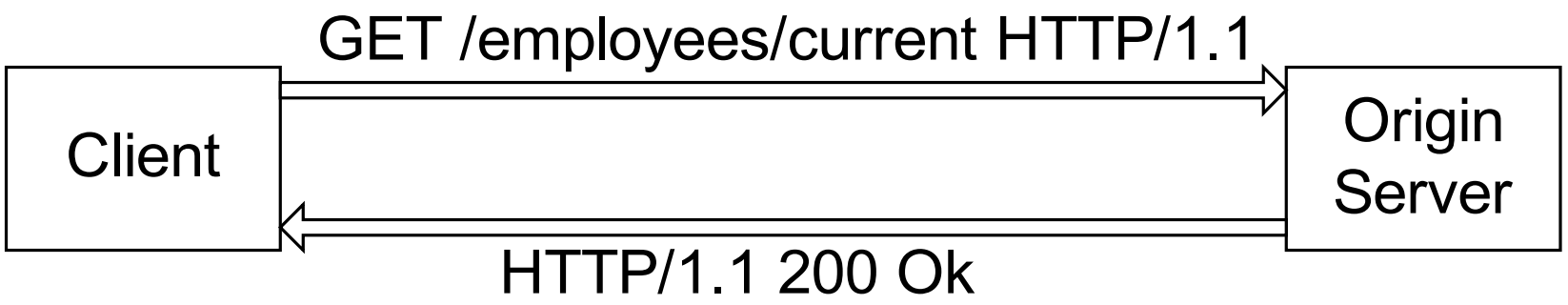
- Problem Statement
- Why REST?
- Why Protocol Buffers?
- Implementation Details
  - Plumbing
  - Data Modeling
- Testing and Release Management
- Results

Why REST?
Uniform Interface

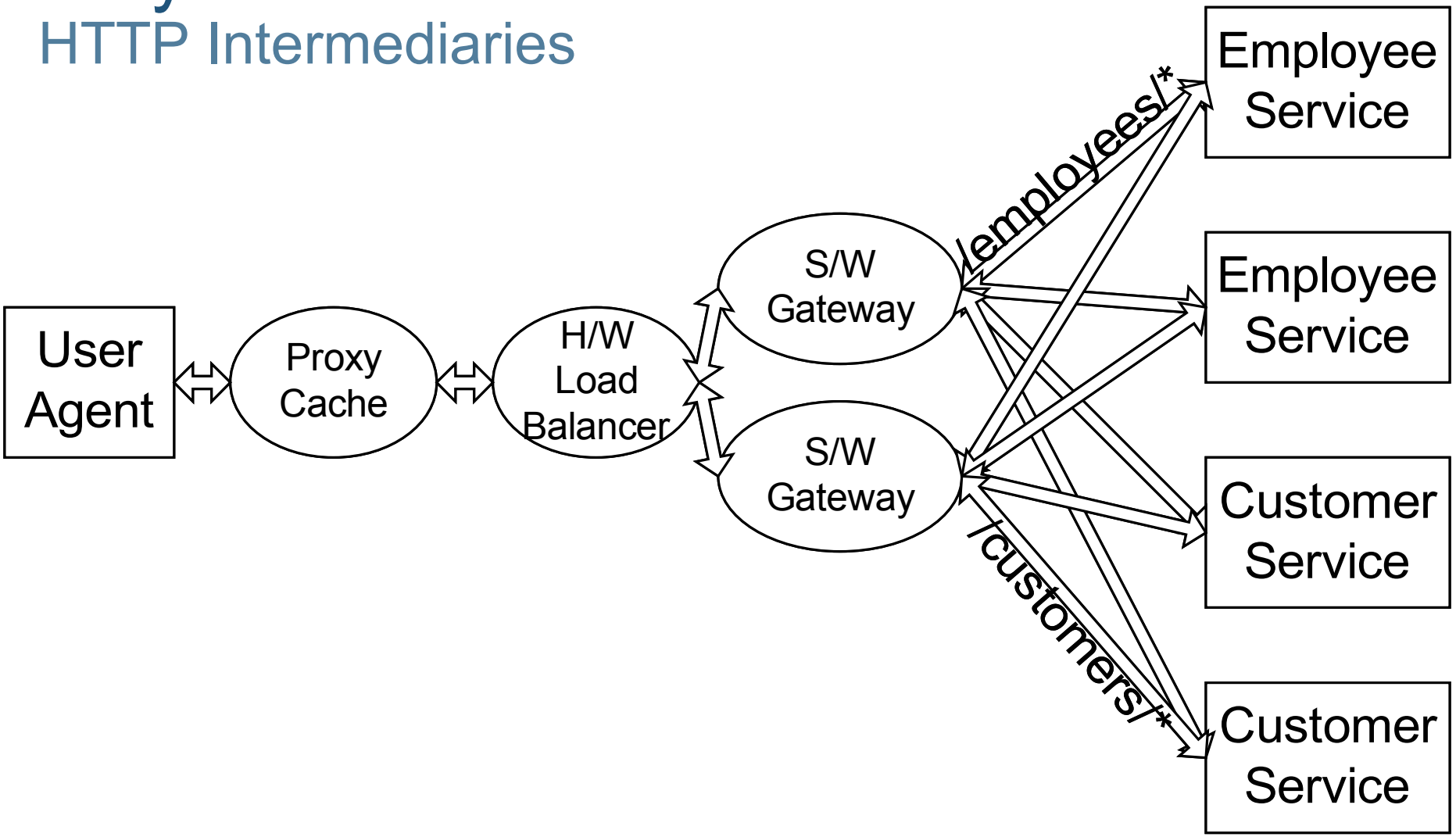
Method	Safe	Idempotent	Visible Semantics	Identifiable Resource	Cacheable
GET	X	X	X	X	X
HEAD	X	X	X	X	X
PUT		X	X	X	
POST					
DELETE		X	X	X	
OPTIONS	X	X	X		

http://rest.blueoxen.net/cgi-bin/wiki.pl?HttpMethods

Why REST?
Simplicity of HTTP



Why REST?
HTTP Intermediaries



Agenda

- Problem Statement
- Why REST?
- Why Protocol Buffers?
- Implementation Details
  - Plumbing
  - Data Modeling
- Testing and Release Management
- Results

Why Protocol Buffers?

Simple Message Format

```
message Person {

  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

    e
```

Why Protocol Buffers?

- Performance
  - Binary encoding
  - 7-10x faster than JSON
  - Even faster than Jini/JERI
- Maturity
  - In use at Google for years
  - Complete toolkit with good docs and community
  - Support for multiple languages
- Evolvability
  - Forward and Backward compatibility
  - Passing of unknown data fields

Why Protocol Buffers?

Evolvability for Agile Oriented Architecture

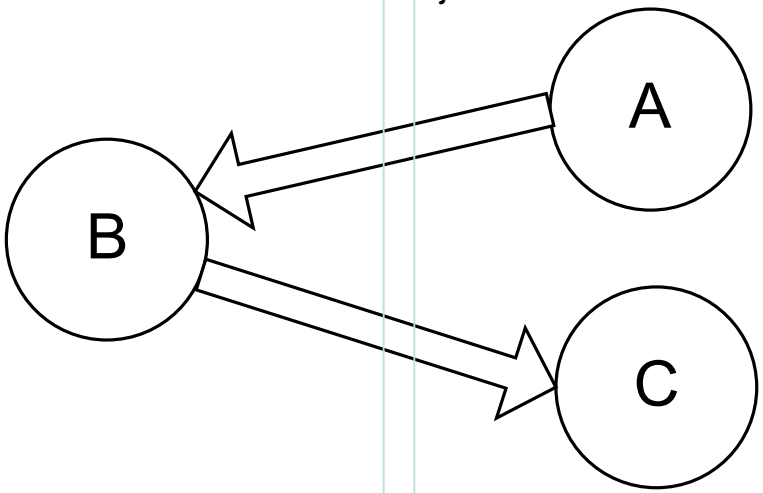
```
message Person {

  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
  repeated PhoneNumber phone = 4;

}
```

```
message Person {

  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
  repeated PhoneNumber phone = 4;
  optional string username = 5;
}
```



Agenda

- Problem Statement
- Why REST?
- Why Protocol Buffers?
- Implementation Details
  - Plumbing
  - Data Modeling
- Testing and Release Management
- Results

Plumbing
What – Where – How

- RESTful Web Services
- URLs
- Standard Methods
- Representations

RESTful Web Services
Best Practices

- Use *Nouns* instead of *Verbs*
- Use proper HTTP methods to indicate your intentions
- REST – No State
- GET is the only immutable operation
- Use HTTP Status codes to communicate the status of your request
- Use '_' to separate the words in your URL
  - Using camelCasing leads to confusion

URLs

- Absolute Path should represent a group of data reducing segments that narrow down data selection
  - GET /employees
    - returns entire set of employees
  - GET /employees/current
    - returns only the ‘current’ subset of employees

URLs
continued…

- GET /employees/{id}
  - returns a specific employee based on its ID
- PUT /employees/{id}
  - updates a specific employee

> Query String should be used for providing optional filtering parameters

- GET /employees/current?last_name=Johnson
  - returns a subset of current employees who's last name is 'Johnson'

Building a Server

```
// EmployeeController.java
@Controller
public class EmployeeController {
    @Resource
    private EmployeeService employeeService;
    @RequestMapping(value = "/employees", method = RequestMethod.GET)
    public Employee getEmployeeById(@RequestParam(value="id")int id) {
            return employeeService.findById(id);
    }
}
```

```
<!– urlrewrite.xml -->
<urlrewrite>
   <rule>
      <from>/employees/([0-9]{1,})?</from>
      <to last="true">/employees?id=$1</to>
   </rule>
</urlrewrite>
```

Building a Client

```
// EmployeeServiceClient .java
public interface EmployeeServiceClient {
    @ServiceName(value=”getEmployeeById")
    public Employee getEmployeeById(@ParamName("id") int employeeId);
}
```

```
// employee.proto
package employee;
message Employee {
   required int32 id = 1;
   optional string name = 2;
   repeated Role role = 3;
   enum Role {
      DEVELOPER = 1;
      MANAGER = 2;
   }
}
```

Representations
Content Encoding

- PROTO should be the default representation for all resources
- Use '*Accept*' header to indicate alternative encodings
  - ***application/json*** - the server will return a JSON (and set the Content-Type to *application/json)*
  - ***application/xml***- the server will return an XML (and set the Content-Type to *application/xml)*
  - **application/x-protobuf** – the server will return the protocol buffers binary (and set the Content-Type to *application/x-protobuf*)

Representations

Content Encoding continued…

- When a Protocol Buffer is returned, the HTTP response should also include an ***X-Protobuf-Schema*** *header containing the URI for the .proto schema file*
- To support random requests using variable output format, i.e. JSON vs. PROTO vs. XML, use *format* parameter on the URL
  - *format parameter on the URL query*
  - *Accept header*
  - PROTO format should be used by default if no format guidance is provided

Agenda

- Problem Statement
- Why REST?
- Why Protocol Buffers?
- Implementation Details
  - Plumbing
  - Data Modeling
- Testing and Release Management
- Results

Data Modeling
Objects vs. Structures

- In distributed services one wants to separate data from behavior
  - Each application has its own object model
  - Contract between applications is defined only by data structure
  - Most business logic should reside in object models which internally operate on the data received as a result of a distributed service call

Data Modeling

Objects vs. Structures continued…

- Protocol Buffers
  - *.proto* files provide language-agnostic structure definition
  - Over-the-wire binary data encoding
    - Decreases payload size
  - Ability to deal with mismatching structure definitions
  - Ability to convert into alternative formats (xml, json)
- Objects
  - Provide behavior
  - Should not be serializable

Data Modeling

Objects vs. Structures continued…

```
// Writing a message
Employee.Builder builder = Employee.newBuilder();
builder.setId(1).setName("John").setRole(Employee.Role.DEVELOPER);
Employee employee = builder.build();
employee.writeTo(response.getOutputStream());
```

```
// Reading a message
Employee employee = Employee.parseFrom(request.getInputStream());
```

```
// Merging a message
builder.mergeFrom(request.getInputStream());
Employee employee = builder.setId(2).build();
```

Data Modeling
Domain Model

- Java Object Models
  - Should provide behavioral logic
  - Should encapsulate the data
    - Wrap the Proto Message with Java Object Model providing the behavior which uses data from the Proto

- “Services should be very thin, most logic should reside in objects” -- Martin Fowler
  - Services should only coordinate interactions between objects

Agenda

- Problem Statement
- Why REST?
- Why Protocol Buffers?
- Implementation Details
  - Plumbing
  - Data Modeling
- Testing and Release Management
- Results

Testing

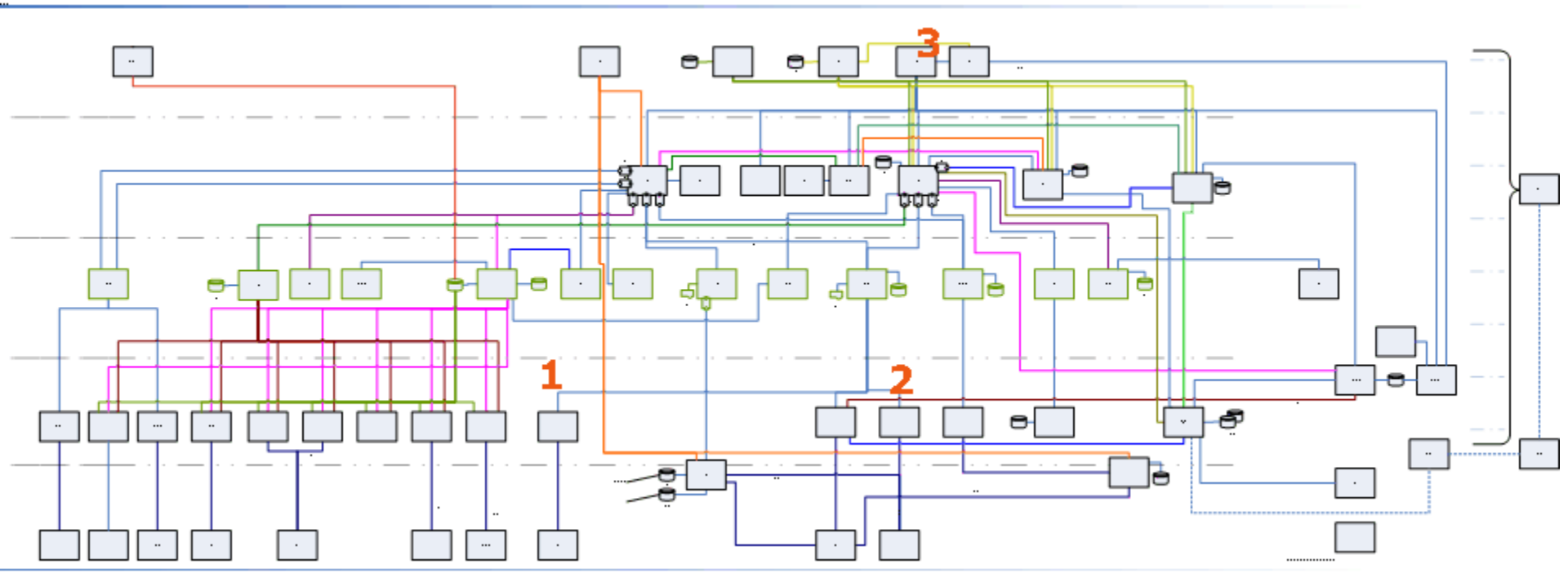
Why Service-Layer Testing is Important

- In order to safely release components independently, Certification Criteria must be defined
- Forward and backward compatibility must be tested and verified
- Test automation is key due to the increased number of releases
- Testing is easier with RESTful services
  - Existing webapp testing tools can be leveraged
  - Browsers can be used for manual tests

Release Management

No More Monolithic Builds

- Focus shifted to higher-order dependencies
- Sequencing is important, but not simultaneity



Results
Smaller, More Frequent Releases

- Loosely coupled components, released independently
- Greater agility and speed with smaller iterations
- Less risk; no more monolithic build
- Stronger contracts and better test coverage
- Greater adoption of commonly used standards and infrastructure

References

- REST – Roy Fielding's Dissertation - http://www.ics.uci.edu/~fielding/pubs/dissertation/tc
- RESTwiki - http://rest.blueoxen.net/cgi-bin/wiki.pl
- Google Protocol Buffers - http://code.google.com/apis/protocolbuffers/docs/o
- Spring MVC - http://static.springframework.org/spring/docs/2.5.x/
- UrlRewrite Filter - http://tuckey.org/urlrewrite/

Thank You

Matt O'Keefe/Alex Antonov
mokeefe@searshc.com
aantonov@orbitz.com

Appendix - Building a Server

- *@Controller* annotation is being used to indicate to Spring that this object class represents a controller
- *@Resource* annotation is being used to indicate to Spring that the annotated field should be injected with a bean corresponding to the type and/or name from the Spring Container

Appendix - Building a Server
continued…

- *@RequestMapping* annotation is being used to indicate to Spring that the annotated method responds to a request pattern
- *@RequestParam* annotation can be used to indicate to Spring what URL parameter name the annotated argument should be mapped to with an optional required attribute to indicate if the parameter is required to be present in the URL (true by default).

Appendix - Building a Client

- InvocationHandler logic would contain the following things:
  - Read the Annotations and construct HTTP request based on them
  - Convert method arguments into URL parameters or RESTful URL patterns
  - Encode the data passed as arguments into format acceptable by HTTP standards
  - Provide abstraction for different possible communication protocols such as JMS, HTTP, RMI, etc.

Appendix - Building a Client
continued…

- *@ServiceName* annotation is being used to provide instructions to the InvocationHandler on the following things:
  - The name of the service which is indicated by the value attribute of the annotation
  - The argument-based url pattern (if one exists)
  - The content type of the desired result output which is indicated by an optional contentType attribute of the annotation (default is protobuf)
    - Possible values: xml, json, text, protobuf

Appendix - Building a Client
continued…

- *@ParamName* annotation is being used to provide instructions to the InvocationHandler on the following things:
  - The name of the parameter, as it is being identified in the pattern of the service call in *@ServiceName* annotation
  - The required status of the argument which is indicated by an optional required attribute of the annotation (default is false)
  - If the name of the parameter is NOT part of the pattern defined in *@ServiceName*, it is being appended to the URL as part of the query string.