



This Presentation Courtesy of the  
**International SOA Symposium**  
October 7-8, 2008 Amsterdam Arena  
[www.soasymposium.com](http://www.soasymposium.com)  
[info@soasymposium.com](mailto:info@soasymposium.com)

Founding Sponsors



Platinum Sponsors



Gold Sponsors



Silver Sponsors



IBM

## REST Design Patterns for SOA

Raj Balasubramanian  
SOA Advanced Technology  
IBM Software Group  
[raj\\_balasubramanian@us.ibm.com](mailto:raj_balasubramanian@us.ibm.com)

09/15/08 © 2008 IBM Corporation

## Agenda

- Introduction
- Context
- Patterns
- Future work
- Q & A

## Introduction

- Customer facing consultant in IBM Software Group
- Experience with J2EE, Portal, SOA and recently Web 2.0 style applications in customer projects
- Interests in OWL and semantics, functional/logic programming for the web
- Built service providers on few different projects using the REST style, for consumption from Portal and other dynamic web apps

## Context - Timeline

- Started working on a Portal/UI chapter for upcoming SOA with Java book
- Wrote a paper for my Masters Report on REST as an implementation style for a traditional SOA style system
- Started working on a REST chapter for the same book - SOA with Java



## Context - Timeline

- Approached by Thomas Erl to review SOA Design Patterns book
- As part of review suggested we look at few REST-inspired patterns that are unique from the rest (no pun)
- Had a short time frame to draft the initial list of patterns.
- Leveraged work from past to look at some key contributions REST could provide



## Context

- Challenges
  - Keep these patterns at the same level as the other patterns in the book (SOA Design Patterns)
  - Introduce the patterns with no real orientation to REST, since the book assumed background knowledge on WS/SOAP/HTTP etc.
  - Time frame

## Context

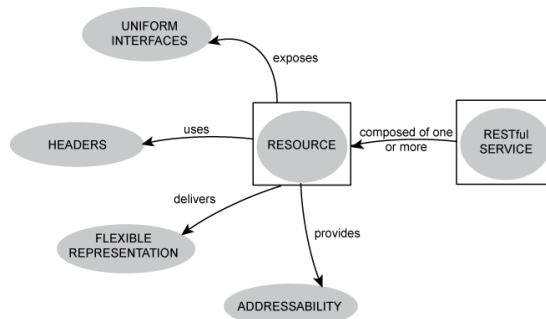
- Thanks to
  - Thomas Erl
  - Jim Webber
  - Kevin Davis
  - Clemens Utschig

## Context

- Disclaimer
  - This is not SOAP/WS\* vs HTTP/REST
  - This is about REST as a design and realization paradigm for services
  - Strived to be unbiased by the politics

## Context

- Concept of a resource and RESTful services
  - Underpinnings of REST (as established by Roy T. Fielding's thesis) and HTTP architecture form the basis of the following 5 basic patterns
- First batch of REST-inspired patterns
- More to come over time



## Context

## Patterns

- Uniform Interface
  - How can services share a common, standardized contract?
- Entity Linking
  - How can services maintain and expose the inherent linkage between business entities?
- Transport Caching
  - How can some of the temporal activity-specific state data be efficiently persisted?

## Patterns

- Layered Redirect
  - How can a service be moved without impacting its consumers?
- Alternative Format
  - How can services exchange data based on less common or non-standard formats?

## Patterns

- Uniform Interface
  - How can services share a common, standardized contract?

## Pattern - Uniform Interface

### Problem

- Custom contract design can be burdensome and risky when the contract content is subject to change
  - Designing customized service contracts that contain validation logic can lead to a variety of challenges:
    - Developers may be required to learn new technology languages.
    - The overall development effort of services increases due to the extra stage of having to custom build the contract content.
    - The contract design and validation logic can become outdated or can be subject to change, thereby risking dependencies formed on the contract by service consumers.
    - Furthermore, once changes to already-implemented service contracts are made, versioning responsibilities enter the picture, thereby increasing the governance burden of services.

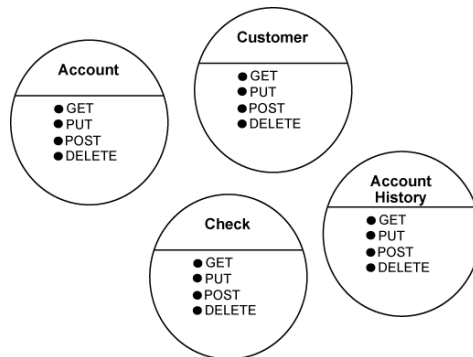
## Pattern - Uniform Interface

### Solution

- A simple and highly generic contract is established and shared by a collection of services, thereby placing the design emphasis on location and structure of the information exchange
  - An existing, pre-defined, and generic service contract is used by all services. This simplifies the overall service inventory architecture and alleviates service designers from having to create individual contracts for services. It further shifts the customization to the data being passed to and from the Uniform Contract.

## Pattern - Uniform Interface

### Sample



## Pattern - Uniform Interface

### Application

- The most common implementation of this pattern is existing practices around REST pattern that enables services to share the common HTTP method contract provided by the Web server
  - The primary application option for this pattern is HTTP method interface provided by all Web servers. Using this generic and prevalent contract, HTTP listeners can receive messages containing one of a number of pre-specified method verbs, such as GET, PUT, POST, DELETE, and HEAD.
  - The notion of Uniform Contract is a core tenet of the REST architectural style and therefore, when applying this pattern using REST, services must support some or all of the possible HTTP methods:
  - *GET method* - performs a read-only function that requires the service to be in the same state after the function is completed
  - *PUT method* - creates new data or data records within the service using the content provided in the HTTP Body field
  - *HEAD method* - carries out an inquiry about a given service without providing the entire response data
  - *DELETE method* - removes a specific body of data
  - *POST method* - can be used to selectively modify or create new data records
  - NOTE: For the traditional web service developer looking to use a single canonical WSDL to enforce this Uniform Contract pattern (by defining key service operations and using a generic schema for message type), the application of this pattern to web services is not appropriate. Since applying the Uniform Contract doesn't leverage the intended richness of web services platform and related technology. The best use case of application of this pattern is by applying the REST design principles using HTTP. HTTP provides a global understanding and accepted standardization around the use of the uniform contract as enforced by the HTTP specification.

## Pattern - Uniform Interface

### Impacts

- Validation logic that is commonly located in the contract layer must now be moved into the core service logic, and an absence of customized contract content introduces a reliance on supplementary human-readable content in order to discover and understand available services
  - Due to the simplicity of the multi-service contract established by this pattern, the richness of defining custom capabilities that semantically define related business tasks as desired by some service developers is lost. Because the single interface established by Uniform Contract is rigid, it can lead to
    - creative use of service definitions - by allocating service functionality to inappropriate operations
    - extensions of intended service operations - by creating variants of service operations from the Uniform Contract (as illustrated by the invention of new verbs in the WebDAV protocol)
    - both of which might undermine the goals of this Uniform Contract pattern.

## Patterns

- Entity Linking
  - How can services maintain and expose the inherent linkage between business entities?

## Pattern - Entity Linking

### Problem

- Business entities are naturally inter-related, yet entity-centric services are commonly designed autonomously with no indication of these relationships
  - In any business environment there are entities that represent the primary and secondary artifacts involved in carrying out the business functions. These business entities can have numerous relationships with each other, many of which may even be dependencies. Some of these relations are physical in nature; some are implicit, while others may even be abstract. Understanding these relationships provides a useful insight as to the inner workings of an organization.
  - However, common service analysis and design approaches result in the delivery of services that are implemented independently with little to no indication of cross-entity relationships communicated to consumers. This can inhibit the consumer designers' understanding of the business entity relationships

## Pattern - Entity Linking

### Solution

- Relationships between services are expressed via the association between entity-related data sets
  - The relationship between business entities is expressed through the service contract or by inter-linking data sets and making these data-level relationships the basis of service-consumer data exchanges .

## Pattern - Entity Linking

### Application

- The most common implementation of this pattern is in the RESTful service that enables business entities to be linked at the information level using URIs in the form of anchor tags or XLink/XPointer in various content types.
- There are two approaches to explicitly associating entities together at the service level:
  - The service contract can provide specific capabilities with names that indicate entity relationships.
  - Uniform Contract can be applied, requiring that entity relationships be defined at the data level and within the corresponding HTTP response or request representation

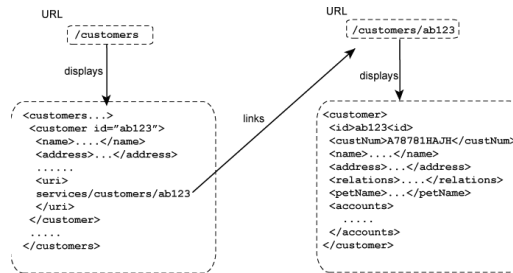
## Pattern - Entity Linking

### Application

- The first approach is simply associated with how service capabilities are named. A Web service contract for a Customer service may expose an operation called “GetCustomerAccounts” to produce a listing of accounts for the given customer. In this case, the operation name itself communicates that the Customer and Accounts entities are related
- The second method relies on physically linking data sets at the level of representation in the request and response message together so that they can be accessed via the content of consumer request messages that express the same data entity relationships. In this case, Entity Linking is commonly applied by implementing a REST framework that positions a Uniform Contract via the standard HTTP method interface

## Pattern - Entity Linking

### Application



```
<xlink:type="simple" xlink:href=
  "http://alleywoodcorp.com/services/customers/ab123">
```

Or

```
<a href="http://alleywoodcorp.com/services/customers/ab123">
```

## Pattern - Entity Linking

### Impacts

- Cross-business entity relationships tends to be more prone to change than the entities themselves, making established links also subject to change, which will impact service consumers
  - Although an extent of entity relationship information can be expressed through customized service contracts, it is almost never a complete representation as it pertains only to the functionality offered by the service at any given point in time. Furthermore, different naming conventions used to determine service and capability names may be too vague and not conducive to accurately describing the nature of these relationships.
  - When this pattern is applied to REST services, XML documents representing entity-centric data sets need to be physically linked. This can tightly bind the underlying data architecture and future changes to these links can have significant impacts because consumers are required to couple to the manner in which these links are established. In essence, this design pattern can result in negative consumer-to-implementation coupling because it simulates the type of negative coupling that occurs when consumer programs are bound to service contracts that mirror underlying physical data models.
  - Furthermore, the primary benefit of navigability is lost when content types used to represent data sets have no standard means of accommodating links

## Patterns

- Transport Caching
  - How can some of the temporal activity-specific state data be efficiently persisted?

## Pattern - Transport Caching

### Problem

- Services required to repeatedly exchange larger amounts of the same data can impose unnecessary runtime performance and bandwidth burden
  - Typical services are dynamic in nature, but there are instances when a subset of the information processed seldom changes. Repeatedly transmitting relatively static, state-specific data can be wasteful.
  - Then there are instances when the information is stable for a given period of time. In this case, the service is required to keep the data in memory while it perhaps waits for other processing to complete. This can be equally wasteful and can impede the service's scalability

## Pattern - Transport Caching

### Solution

- Services are designed to leverage built-in caching features provided by the transport communications framework in order to defer and persist some forms of state data
  - A transport communications framework with built-in caching features can be leveraged as a runtime state deferral mechanism. This allows the service to temporarily off-load static or idle state data until it needs to retrieve it at a later point. The result is that less data needs to be repeatedly transmitted and the service's overall memory consumption is reduced .

## Pattern - Transport Caching

### Application

- Available caching features of an application protocol (most commonly HTTP) are incorporated into service and composition designs
  - Within contemporary service inventory architectures, this pattern is most commonly applied using the caching features provided by HTTP.
  - HTTP supplies a set of headers and associated rules that can be used to identify a cache, how (and how long) to cache state data and when it should be cached. By using these features, there are three common cache types that can be employed:
    - *Gateway Caching* - The service can use the centralized HTTP caching facility that is built into any Web server. This caching model is based on a reverse proxy that makes the decision as to whether to forward a request from the consumer to the service or to look it up in its cache. There are several variations of this caching model available in supplementary proxy products and appliances.
    - *Intermediary Proxy Caching* - Service consumers can use an intermediary facility based on a standard proxy server that provides a shared HTTP cache. Consumers will have to point to this proxy server before making any requests to the service itself.

## Pattern - Transport Caching

### Application (contd.)

- *Client-side Caching* - With this approach the cache resides in the presentation layer, usually as part of a rich application interface or Web browser. This option only pertains to services with which client-side programs interact directly. A common example is the use of REST services that help assemble presentation-centric mashups.
- In addition to the caching options or strategies mentioned above, there are specific directives that the HTTP specification makes available to developers to implement caching, specifically leveraging HTTP headers, as follows:
  - The Response header can be used to dictate whether or not to cache a body of state data. In HTTP 1.1 it is represented by the Expires and Cache-Control header elements.
  - The Last-Modified response header and the If-Modified-Since request headers can be used to implement caching logic depending on when the state data was altered. The corresponding headers in HTTP 1.1 are the ETag response header and the If-None-Match request header. The decision to use a given header is made based on the benefits to the overall service architecture

## Pattern - Transport Caching

### Impacts

- Additional infrastructure may be required to effectively apply this pattern in high usage environments
  - The caching functionality provided by a transport communications framework can be convenient and efficient, but it may require additional infrastructure and associated caching configurations. Furthermore, there may be security implications to having data reside on the transport level, rather than behind the service contract as when State Repository or Stateful Services are applied .

## Patterns

- Layered Redirect
  - How can a service be moved without impacting its consumers?

## Pattern - Layered Redirect

### Problem

- Various logistical circumstances may require a service to be physically relocated, thereby requiring configuration or code changes on the consumer end. There are numerous reasons as to why a service may need to be relocated:
  - IT departments may be subject to restructuring due to corporate acquisitions or company-wide re-organizations.
  - Infrastructure may need to be scaled requiring the introduction of new servers and the redistribution of previously deployed services.
  - New security requirements may demand that a service be isolated or relocated to a different enterprise domain

## Pattern - Layered Redirect

### Solution

- Application and Transport-level redirect features are used to preserve consumer-to-service connectivity
  - There are two ways to address the problem:
    - Redirection at the Transport Protocol layer - This approach ensures the redirection occurs at the network transport protocol (TCP) layer and addressed by use of network devices.
    - Redirection at the Application Protocol layer- This approach ensures the redirection occurs at the application transport protocol (HTTP) layer and addressed by use of application proxies and intermediaries.
  - In addition to the above approaches, there can be two types of service relocation-
    - *Transparent Relocation* - This approach ensures that the consumer is unaware of the change and that all service interaction proceeds normally.
    - *Relocation with Referrals* - A mechanism residing at the service's original location responds to consumer requests with an appropriate code indicating the new service location.

## Pattern - Layered Redirect

### Application

- Redirection logic placed at the original service location either transparently redirects consumer requests or responds to consumers with a notification indicating the new service location. This redirection can be applied at the transport protocol level or at the application protocol level
  - Transparent relocation is typically achieved by the application of Intermediate Routing together with specialized redirection logic or by using core networking services (such as DNS). This redirection can be implemented at the Transport or Application Protocol layer. At the transport protocol layer, network appliances are used. This can be applied to both HTTP based REST services as well as SOAP based web services. HTTP specification offers guidance on accomplishing the redirects as indicated below. This is leveraged fully in RESTful services. For SOAP based services, there are additional intermediaries in the form of middleware environments, such as the Enterprise Service Bus, to accomplish routing functionality
  - Implementing relocation by referral can also be achieved via Intermediate Routing logic, but is more commonly built at the application protocol layer. HTTP specification offers guidance on accomplishing the redirects as indicated below. This is leveraged fully in RESTful services.
  - The HTTP specification contains several provisions that address temporary move and permanent moves, primarily due to its origins in Web publishing. Some of the more relevant HTTP codes that are associated with the header fields in the HTTP response include:
    - 301 Moved Permanently
    - 302 Found
    - 303 See Other
    - 307 Temporary Redirect
  - The modified address or the older address of the service is specified in the Location field in the response header. Additional information can be provided in the response body.

## Pattern - Layered Redirect

### Impacts

- Requires additional planning of resource design as well as possible impact to the intermediaries - proxies, network devices etc
  - When following the transparent relocation approach, it may be tempting to never update consumers because service interaction continues to occur as it always has. For anything that requires more than changes at the networking (DNS) level, this pattern can lead to the need for middleware or network appliances and can further compound governance efforts required to keep track of all of the old and updated locations that need to be continually maintained.
  - The relocation by referral method either requires that consumers undergo a change to incorporate the new service location or that they be designed to receive the response codes containing the relocation information. Either way, this will impact consumer programs that were not designed for this type of response.
    - Note: Due to the HTTP-centric nature of REST-based implementations, consumers of REST services may naturally contain the processing logic required to accept HTTP response codes .

## Patterns

- Alternative Format
  - How can services exchange data based on less common or non-standard formats?

## Pattern - Alternative Format

### Problem

- Service consumers may have data exchange requirements based on non-XML formats
  - Service consumers can have diverse requirements, especially when they exist as alternative programs in mobile devices or on desktop applications. Whereas most consumers will tend to expect a standard XML representation, others may require different formats, such as binary-encoded images or portable data format (PDF) based data

## Pattern - Alternative Format

### Solution

- Services can be designed to exchange data in alternative representation formats, thereby accommodating special consumer requirements
  - Services are designed to support data exchange in multiple or alternative data formats. Either service contracts are extended to include additional capabilities with the functionality to receive and respond with messages containing different formats, or processing is added to transform a service's natural data representation format to and from the consumer's desired representation format .

## Pattern - Alternative Format

### Application

- How this pattern is applied is dependent on the endpoint type of the service and technology required to support the desired data format .
- There are two common approaches for supporting Alternative Format in services:
  - Use an attachment technology to associate the alternatively formatted data with an XML message and configure the service to address the processing of these attached content formats.
  - Use any one of the supported content-types or custom representation, available natively to the application protocol (such as HTTP) and configure the service to address the processing of these different content formats.

## Pattern - Alternative Format

### Application (contd.)

- HTTP Mime-type example-For example, to represent XML the media-type value would be text/xml or application/xml, to represent HTML it is text/html, and the Adobe PDF format is application/pdf. When using HTTP functionality with a REST framework (such as Restlet etc.) of choice, the JSON (JavaScript Object Notation) format is also commonly supported. Hence a web application consuming this service from a browser can process this response with little overhead as these are just another JavaScript objects.

- For example, this simple XML fragment:

```

<customers>
  <customer>
    <name>John Smith</name>
    <type>Gold</type>
  </customer>
</customers>

```

...would be represented in JSON as:

```

{customers:{
  customer:[
    {
      name:"John Smith",
      type:"Gold"
    }
  ]
}

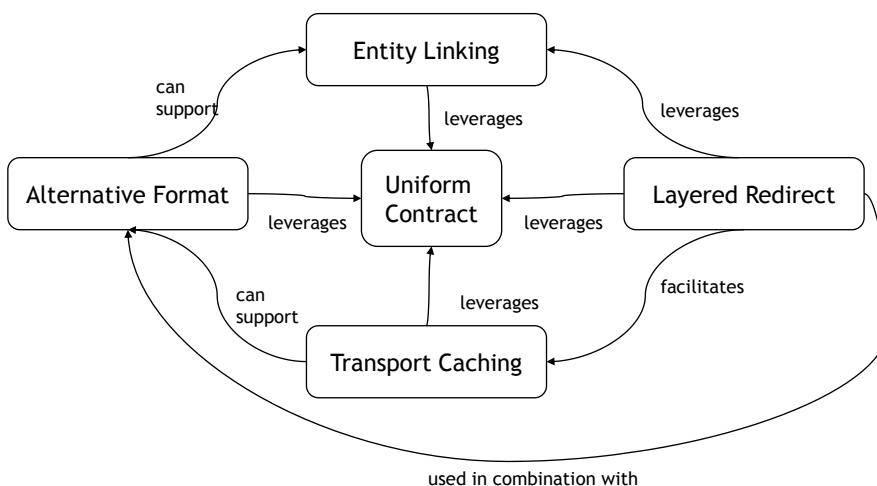
```

## Pattern - Alternative Format

### Impacts

- This pattern can deviate from the goal of establishing Canonical Schema in a single representation format, across an enterprise
  - The primary impact is on the service implementation, since the additional logic to accept the various media-types and respond with unique media-types has to be addressed as part of the implementation. The impact on the consumer is that they have to perform additional calls to the service to inquire about the supported content as part of the content negotiation process .

## Patterns - Relations





IBM



Thank You

09/15/08

© 2008 IBM Corporation