© Copyright 2007, Red Hat, Inc.

REST in Peace

Rajith Attapattu
Red Hat, Inc.
rajith@redhat.com

© Copyright 2007, Red Hat, Inc.

About

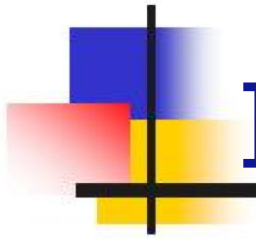
- Actively involved in several open source projects
- Karma on Apache Qpid, Axis2, Tuscany
- Contributed to Apache Synapse, Geronimo
- Involved in AMQP spec group

© Copyright 2007, Red Hat, Inc.

Agenda

- Introduction
- Understand the REST architectural style
- Understand the merits/demerits of REST
- Discuss REST within the context of WS

© Copyright 2007, Red Hat, Inc.

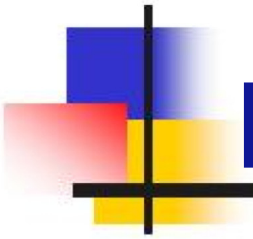


Introduction

- Representational State Transfer
- Introduced in Roy Fielding's PhD thesis

  http://www.ics.uci.edu/~fielding/pubs/dissertat

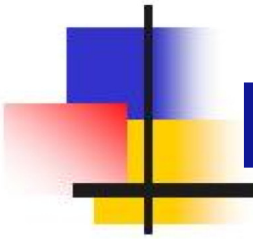
© Copyright 2007, Red Hat, Inc.



REST – Quick Overview

- REST is an architectural style.
- It's not XML/HTTP minus the WS baggage.
- Defines a set of architectural constraints.
- It's the architecture behind the modern Web.

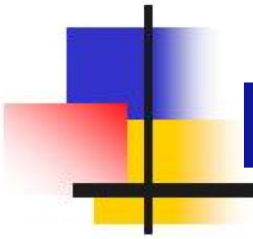
© Copyright 2007, Red Hat, Inc.



Let's keep this in mind

- When applied as a whole REST emphasizes
  - Scalability of component interactions
  - Generality of interfaces
  - Independent deployment of components
  - Intermediary components to reduce interaction latency
  - Enforce security
  - Encapsulate legacy system

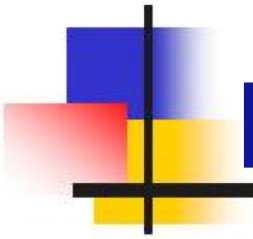
© Copyright 2007, Red Hat, Inc.



Background from the thesis

- Architectural style is a collection of,
- Architectural constrains.
- New styles can be derived by adding,
- Constraints or combining styles.
- Null style – no constraints

© Copyright 2007, Red Hat, Inc.



Defining REST Incrementally

- Start from Null style – no constraints
- No boundaries between components.
- Lets start adding constraints one by one

© Copyright 2007, Red Hat, Inc.

Client Server Style

- Let's add the constraints from Client Server
- Constraint – Separation of concerns
  - Improves Portability of clients
  - Improves Scalability of server components
  - Components can evolve independently

© Copyright 2007, Red Hat, Inc.

Stateless Communication

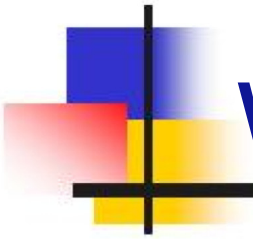
- Let's add another constraint
- Communication must be stateless
  - Visibility – not having to look beyond immediate request
  - Reliability – easy to recover from partial failures
  - Scalability – not having to store state between requests
- Let's discuss some drawbacks too

© Copyright 2007, Red Hat, Inc.

Cache

- Let's add another constraint – Cache
- If response is tagged cacheable, data will be
- Reused for response in equivalent requests.
  - Completely or partially eliminate some interactions.
  - Reduces average latency of series of interactions
  - Improves scalability and user perceived performance

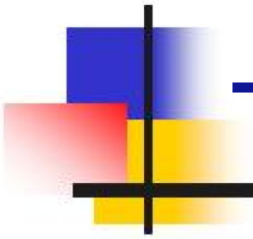
© Copyright 2007, Red Hat, Inc.



Web before 1994

- Web < 1994 was defined by these constraints.
- Expectations exceeded the early design
  - Dynamically generated responses – server side scripts
  - Intermediary components – proxies, shared caches

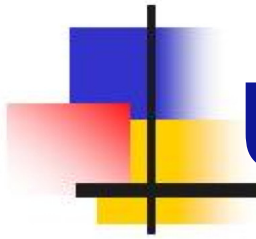
© Copyright 2007, Red Hat, Inc.



The Modern Web

- Constraints were added to guide
- The Modern Web Architecture
- Lets discuss them

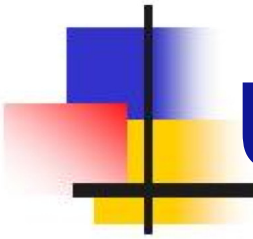
© Copyright 2007, Red Hat, Inc.



Uniform Interface

- Central constraint that distinguishes REST
- From other network architecture styles.
  - Overall System architecture is simplified
  - Visibility of interactions are improved
  - Implementations are decoupled from the service
  - Which means the service can evolve independently

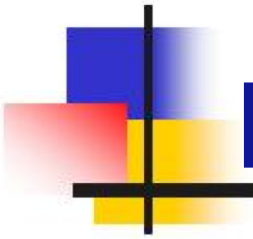
© Copyright 2007, Red Hat, Inc.



Uniform Interface

- Constraints required to satisfy above
  - Identification of resources.
  - Manipulation of resources through representations.
  - Self descriptive messages.
  - Hypermedia as the engine of application state.

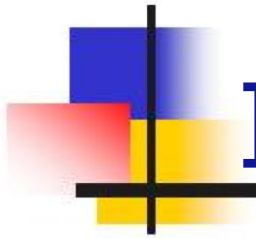
© Copyright 2007, Red Hat, Inc.



Layered System

- Add constraints of a layered system
- Cannot see beyond immediate layer
  - Bound on complexity
  - Substrate independence
  - Layers can encapsulate legacy services/clients
  - Intermediaries – load balancing, enforcing security

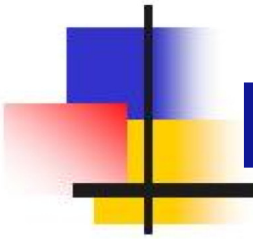
© Copyright 2007, Red Hat, Inc.



Intermediaries

- Uniform Interface + Layered system enables
  - Intermediaries to transform/process messages, As
    - Messages are self descriptive, and their
    - Semantics are visible to intermediaries

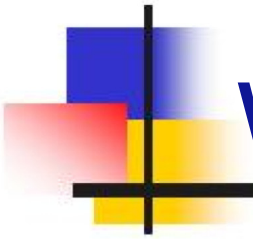
© Copyright 2007, Red Hat, Inc.



REST – Summary

- We looked at the REST constraints that
- Defines the REST architectural style.
- Layered Client Server + Uniform Interface
- Lets dig deeper into the interface constraints

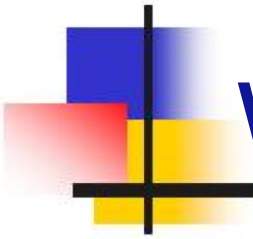
© Copyright 2007, Red Hat, Inc.



Where to process the data

- Let's identify the 3 options for processing data
- And discuss the merits/demerits of each option
- Option 1
  - Process the data where it is located
  - Then send in a fixed format to the recipient
  - This is the traditional client server style

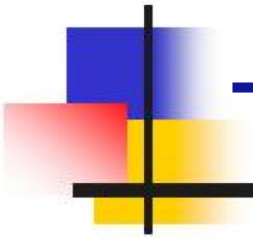
© Copyright 2007, Red Hat, Inc.



Where to process the data...

- Option 2
  - Send the raw data + rendering engine to recipient
  - Mobile object style – ex. Java applets
- Option 3
  - Send the raw data + meta data to recipient
  - Recipient can choose its own rendering engine

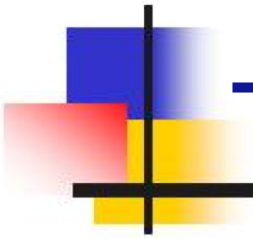
© Copyright 2007, Red Hat, Inc.



Transferring a Representation

- REST provides a hybrid of all 3 options by Focusing on a shared understanding of Data Types and Meta Data.
- REST components communicate by Transferring a representation of the resource.

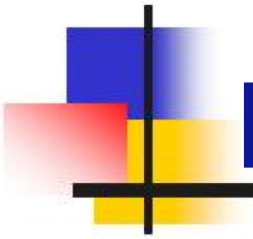
© Copyright 2007, Red Hat, Inc.



Transferring a Representation

- The representation is provided in a format
- Selected dynamically based on the
  - Capabilities/Desires of the recipient
  - Nature of resource
- Whether it is raw or derived from the source
- Remains hidden behind the interface.

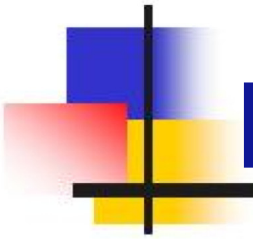
© Copyright 2007, Red Hat, Inc.



Benefits

- The representation is provided in a format
- Selected dynamically based on the
  - Capabilities/Desires of the recipient
  - Nature of resource
- Whether it is raw or derived from the source
- Remains hidden behind the interface.

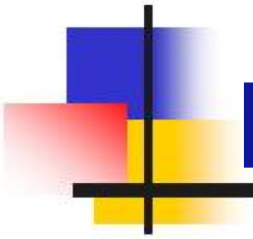
© Copyright 2007, Red Hat, Inc.



Resources & Identifiers

- Key abstraction of information is a resource.
- Conceptual mapping to a set of entities
- Not the entity itself at any given time.
- The value can be static or vary over time.
- Semantics of the mapping needs to be static

© Copyright 2007, Red Hat, Inc.



Resources & Identifiers

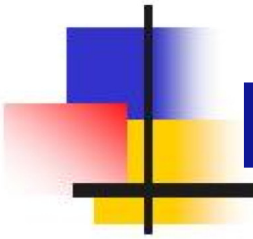
- The abstraction provides following benefits
  - Late binding of reference to a representation
  - Reference the concept not the representation
- A *Resource Identifier* is used to identify the
- Particular resource involved in an interaction

© Copyright 2007, Red Hat, Inc.

Self Describing Messages

- The server or intermediaries does not have
- To Look beyond the current request.
- The message is self descriptive
- And the semantics are visible.

© Copyright 2007, Red Hat, Inc.



Manipulation of Resources

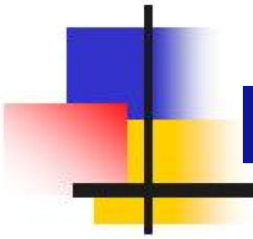
- The REST components performs actions on
- A *resource* by using a representation to
- Capture the current/intended state of that
- Resource and transferring that
- Representation between components.

© Copyright 2007, Red Hat, Inc.

Hypermedia as the engine of application state

- The Next control state of the application,
- Resides in the representation of the current,
- Requested resource.
- Ex. A web page has links to other pages

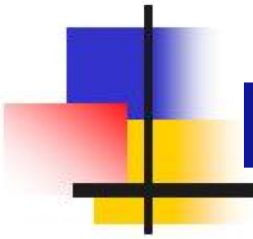
© Copyright 2007, Red Hat, Inc.



Representational State Transfer

- Application starts with a resource identifier
- Then follows a network of states based on
- The analysis of representation provided.
- Each representation will place the
- Application in a different state.

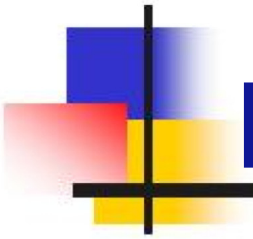
© Copyright 2007, Red Hat, Inc.



Lets revisit the following

- When applied as a whole REST emphasizes
  - Scalability of component interactions
  - Generality of interfaces
  - Independent deployment of components
  - Intermediary components to reduce interaction latency
  - Enforce security
  - Encapsulate legacy system

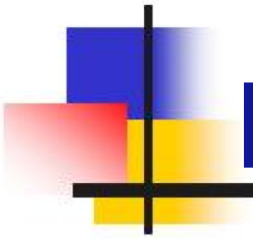
© Copyright 2007, Red Hat, Inc.



REST and the Web

- The Web is a concrete example of REST.
- HTTP verbs provide the Uniform Interface
- URI provides resource identification
- Representations – HTML, XML, JPEG, GIF... *etc*
- Not everything on the Web is RESTful

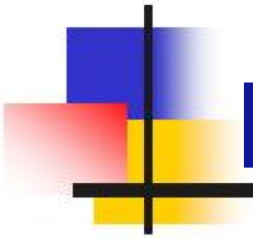
© Copyright 2007, Red Hat, Inc.



REST and the Web

- The Web is a concrete example of REST.
- HTTP verbs provide the Uniform Interface
- URI provides resource identification
- Representations – HTML, XML, JPEG, GIF... *etc*
- Not everything on the Web is RESTful

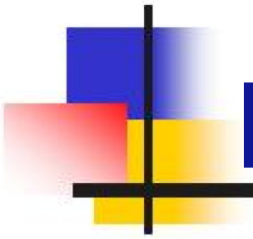
© Copyright 2007, Red Hat, Inc.



Building RESTful Services

- Currently a hot topic
- Everybody claims to support REST
- Lots of misinformation
- Lets discuss what it take to,
- Make your web service RESTful?

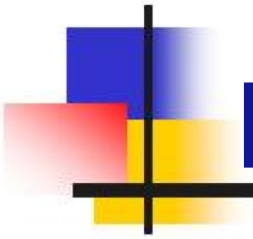
© Copyright 2007, Red Hat, Inc.



Building RESTful Services

- Simply put, your service needs to obey
- The REST constraints.
- Doing POX/HTTP doesn't cut it :)
- REST assumes neither XML nor HTTP.

© Copyright 2007, Red Hat, Inc.



Building RESTful Services

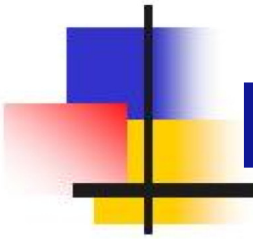
- Representations can be in any format
- Not just XML as many believe.
- As long as it is self describing
- Ex. JPEG or even SOAP :)

© Copyright 2007, Red Hat, Inc.

Can a Framework help

- I doubt a framework can provide magic
- To help make your service RESTful.
- A lot depends on how you design your
- Service and the relationships.

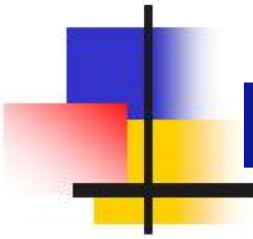
© Copyright 2007, Red Hat, Inc.



REST is easy! – is it really?

- The most common misconception.
- RPC masquerading as XML/HTTP doesn't count.
- Turning a class or a method into a resource is
- Not as easy as u think.

© Copyright 2007, Red Hat, Inc.



Links