The Rest of REST

Roy T. Fielding, Ph.D.

Chief Scientist, Day Software
V.P., Apache HTTP Server

Overview

Representational State Transfer (REST)

- A little background
  - WWW history + Roy history = REST context
  - Why do we need a Web architectural style?
- A touch of theory
  - Principled design
  - Architectural properties
  - Constraints that induce properties
- What parts of REST are missing from Ruby on Rails?
- Industry reactions to REST
  - and a little bit of Relaxation
Life's race will run, Life's work well done, Life's victory won,
Now cometh REST. [Dr. Edward Hazen Parker]

Why me?

Using XMosaic
www.ics.uci.edu
wwwstat
MOMspider
Conditional GET
libwww-perl
Relative URLs
HTML 2.0
HTTP editor

1st WWW
2nd WWW
SJ IETF
23,517
10,022
2,738
130
623

REST BEGINS AS HTTP OBJECT MODEL

Public WWW servers [Matthew Gray]

Tuesday, September 18, 2007
The Problem (circa 1994)

Early architecture was based on solid principles

• URLs, separation of concerns, simplicity
• lacked architectural description and rationale

Protocols assumed a direct server connection

• no awareness of caching, proxies, or spiders
• many independent extensions

Public awareness of the Web was just beginning

• exponential growth threatened the Internet
• commercialization meant new requirements and new stakeholders

A modern Web architecture was clearly needed

• but how do we avoid breaking the Web in the process?
A software architecture is an abstraction of the run-time elements of a software system during some phase of its operation. A system may be composed of many levels of abstraction and many phases of operation, each with its own software architecture.

- A software architecture is defined by a configuration of architectural elements—components, connectors, and data—constrained in their relationships in order to achieve a desired set of architectural properties.
- A configuration is the structure of architectural relationships among components, connectors, and data during a period of system run-time.
Architectural Styles

An architectural style is a coordinated set of architectural constraints that restricts the roles and features of architectural elements, and the allowed relationships among those elements, within any architecture that conforms to that style.

- A style can be applied to many architectures
- An architecture can consist of many styles

Design at the right level of abstraction

- Styles help architects communicate architecture
- Architecture determines potential system properties
- Implementation determines actual system properties
- Architectural patterns are styles with common recipes
What is the Web, really?

Information

Protocols

Browsers

Oh, some seek bread—no more—life’s mere subsistence, ...

Tuesday, September 18, 2007
... And some seek wealth and ease--the common quest; ...
Web Architecture

One abstraction level above the implementation

Components

- User agents, Intermediaries, Servers
- Browsers, Spiders, Proxies, Gateways, Origin Servers

Connectors

- HTTP: a standard transfer protocol to prefer over many

Data

- URI: one identifier standard for all resources
- HTML, XML, RDF, ...: common representation formats to describe and bind resources
Web Architectural Style

One abstraction level above Architecture

- two abstraction levels above implementation
- that’s one too many for most folks

An architectural style is a set of constraints

- unfortunately, constraints are hard to visualize
  - kind of like gravity or electromagnetism
  - observed only by their effect on others

Constraints induce architectural properties

- both desirable and undesirable properties
  - a.k.a., software qualities
  - a.k.a., design trade-offs
Web Requirements

Low entry barrier

- Hypermedia User Interface
- Simple protocols for authoring and data transfer
  • a.k.a., must be **Simple**, **Reusable**, and **Extensible**

Distributed Hypermedia System

- Large data transfers
- Sensitive to user-perceived latency
  • a.k.a., must be **Data-driven**, **Streamable**, and **Cacheable**

Multiple organizational boundaries

- Anarchic scalability
- Gradual and fragmented change (deployment)
  • a.k.a, must be **Scalable**, **Evolvable**, **Visible**, **Reliable**, ...
Sometimes the most urgent and vital thing you can possibly do is take a complete REST. [Ashleigh Brilliant]
Style = nil

Starting from a condition of no constraints...

How beautiful it is to do nothing, and then REST afterward. [Spanish Proverb]
Apply separation of concerns: Client-Server

- improves UI portability
- simplifies server
- enables multiple organizational domains
Style += Stateless

Constrain interaction to be stateless...

- simplifies server
- improves scalability
- improves reliability
- degrades efficiency

... and to lie sometimes on the grass ...

Tuesday, September 18, 2007
Style += Caching

Add optional non-shared caching

- reduces average latency
- improves efficiency
- improves scalability
- degrades reliability

... under the trees on a summer's day, ...

Tuesday, September 18, 2007
Style += Uniform Interface

Apply generality: uniform interface constraint

- improves visibility
- independent evolvability
- decouples implementation
- degrades efficiency
- improves visibility
Style += Layered System

Apply info hiding: layered system constraints

- adds latency
- shared caching
- legacy encapsulation
- simplifies clients
- improves scalability
- load balancing

Tuesday, September 18, 2007
Finally, allow code-on-demand (applets/js)

simplifies clients  improves extensibility  reduces visibility
REST Uniform Interface

All important resources are identified by one resource identifier mechanism
- simple, visible, reusable, stateless communication

Access methods (actions) mean the same for all resources (universal semantics)
- layered system, cacheable, and shared caches

Resources are manipulated through the exchange of representations
- simple, visible, reusable, cacheable, and stateless communication

Exchanges occur in self-descriptive messages
- layered system, cacheable, and shared caches
Hypertext as the engine of application state

- A successful response indicates (or contains) a current representation of the state of the identified resource; the resource remains hidden behind the server interface.

- Some representations contain links to potential next application states, including direction on how to transition to those states when a transition is selected.

- Each steady-state (Web page) embodies the current application state
  - simple, visible, scalable, reliable, reusable, and cacheable network-based applications

- All application state (not resource state) is kept on client
- All shared state (not session state) is kept on origin server
Hypertext Clarification

Hypertext has many (old) definitions

• "By 'hypertext,' I mean non-sequential writing — text that branches and allows choices to the reader, best read at an interactive screen. As popularly conceived, this is a series of text chunks connected by links which offer the reader different pathways" [Theodor H. Nelson]

• “Hypertext is a computer-supported medium for information in which many interlinked documents are displayed with their links on a high-resolution computer screen.” [Jeffrey Conklin]

When I say Hypertext, I mean ...

• The simultaneous presentation of information and controls such that the information becomes the affordance through which the user obtains choices and selects actions.

• Hypertext does not need to be HTML on a browser — machines can follow links when they understand the data format and relationship types
REST Rationale

Maximizes reuse
• uniform resources having identifiers = Bigger WWW
• visibility results in serendipity

Minimizes coupling to enable evolution
• uniform interface hides all implementation details
• hypertext allows late-binding of application control-flow
• gradual and fragmented change across organizations

Eliminates partial failure conditions
• server failure does not befuddle client state
• shared state is recoverable as a resource

Scales without bound
• services can be layered, clustered, and cached

Simplifies, simplifies, simplifies
What is missing from Rails?

*Just newbie speculation, without looking at edge:*

**Uniform method semantics?**
- Rails support (via CRUD) is outstanding
- but what happens when I add a new HTTP method?

**Resource identifiers for important resources?**
- Route configs are good, but code-structure dependent
- URI templates would be better, IMO

**Resources manipulated as representations?**
- Rails has excellent support for alternative data formats

**Hypertext as the engine of application state?**
- Is this just assumed? Can it be guided by Rails?
A little relaxation

Roy T. Fielding, Ph.D.

Chief Scientist, Day Software
V.P., Apache HTTP Server

Industry Practice

Meanwhile, in a parallel universe ...

- http://www.youtube.com/watch?v=-RxhkWLJH4Y
- Microsoft was selling COM+/DCOM
- IBM and friends were selling CORBA
- Sun was selling RMI
- W3C was developing XML

Then SOAP was dropped on the shower floor as an Internet Draft

- and quickly laughed out of the IETF
- only to be picked up by IBM and renamed “Web Services”

and REST became the only counter-argument to multi-billions in advertising
Industry Reaction?

Not very constructive

- proponents labeled as RESTafarians
- arguments derided as a “religion”
- excused as “too simple for real services”

Service-Oriented Architecture (SOA)

- a direct response to REST
- attempt at an architectural style for WS
  - without any constraints
- What is SOA?
  - Wardrobe, Musical Notes, or Legos?
  - http://www.youtube.com/profile_videos?user=richneckyogi
Industry Acceptance

Something has changed ...

• People started to talk about the value of URIs (reusable resources)
• Google maps decided to encourage reuse (Mashups)
• O’Reilly began talking about Web 2.0
• Rails reminded people that frameworks can be simple

and REST(ful) became the next industry buzzword

Yikes!

REST is sweet after strife. [Lord Edward Robert Bulwer Lytton]
Relaxation

Clearly, it’s time to start messing with minds

- REST is not the only architectural style
- My dissertation is about Principled Design, not the one true architecture

What do constraints really mean?

- codify a design choice at the level of architecture
- to induce certain (good) architectural properties
- at the expense of certain (bad) trade-offs

What happens when we relax a given constraint?

- Is it really the end of the world?
- Should waka (a replacement for HTTP) have its own style?
Relax uniform methods?

What happens when we let the interface be resource-specific?

• URI is no longer sufficient for resource identification
  – lose benefit of URI exchange (assumed GET)
  – require resource description language
• Information becomes segregated by resource type
  – walled into gardens (loss of power laws / pagerank)
  – important information must be replicated
• Intermediaries cannot encapsulate services
  – unable to anticipate resource behavior
  – too complex to cache based on method semantics
• No more serendipity
Relax client/server?

What happens when we let servers make requests?

- lose implementation simplicity due to listening, additional parsing requirements
- potential for confusion with mixed-protocol intermediaries
- unknown: does it impact session state?

Trade-offs aren’t as severe as the first example.

Benefits?

- peer-to-peer applications
- shared cache mesh, triggered expiration

Can we find ways to compensate for the trade-offs?

- Make message syntax more uniform
  - Limit server-initiated requests to same-connection
Conclusion

Use your brains!

• don’t design-by-buzzword
• don’t believe everything you read
• always keep in mind that change is inevitable
• use principled design
  – identify desired architectural properties
  – constrain behavior to induce properties
  – compensate for the inevitable design trade-offs

Let the weary at length possess quiet REST.

[Lucius Annaeus Seneca]