Reflections on the REST Architectural Style and “Principled Design of the Modern Web Architecture”

Roy T. Fielding
Richard N. Taylor
Justin R. Erenkrantz
Michael M. Gorlick
Jim Whitehead
Rohit Khare
Peyman Oreizy

Adobe
University of California, Irvine
Bloomberg
University of California, Irvine
University of California, Santa Cruz
Google
Dynamic Variable LLC
Outline

1. The Story of REST
   - Early history of the Web
   - What REST is (and is not)
   - Contemporary influences

2. Work inspired by REST
   - Decentralization
   - Generalization
   - Secure computation

3. Reflections on REST
   - Investing in entrepreneurial students
   - Role of Software Engineering research
Original proposal for the World Wide Web

[Hypertext] describes [Comms ACM]

This document

A Proposal "Mesh"

Linked information

"Hypertext"

Hypermedia

Hyper Card

ENQUIRE

CERNDOC

C.E.R.N

IBM GroupTalk

uucp News

Hierarchical systems

for example

unifies

describes

includes

wrote

refers to

describes

includes

Tim Berners-Lee

[Berners-Lee, 1989]
The Web is an application integration system

Linked information

A Proposal "Mesh"

Hyper Card

ENQUIRE

Computer conferencing

IBM GroupTalk

Hierarchical systems

for example

VAX/NOTES

uucp News

for example

Linked information

CERNDOC

describes

includes

[Berners-Lee, 1989]
A bit of context

- Jun 93: 130
- Dec 93: 623
- Jun 94: 2,738
- Dec 94: 10,022
- Jun 95: 23,517

Aug 2017: 1,800,566,882 (76,564x)
A bit of context

- Using the Web
- Conditional GET
- Relative URLs
- HTTP editor
- wwwstat
- 2nd WWW
- 1st WWW
- SJ IETF
- HTTP Object Model

PUBLIC WWW servers [Matthew Gray]

- Jun 93
- Dec 93
- Jun 94
- Dec 94
- Jun 95

Aug 2017: 1,800,566,882 (76,564x)

Using the Web

wwwstat

Conditional GET

Relative URLs

HTTP editor

HTTP Object Model
Three (very different) perspectives of the Web

Browsers

Information

Protocols
Web Implementation (user view)
Web Implementation (origin view)

User Agents

Webservers/Gateways

Accelerator Cache

Application Servers
Dynamic Content

Intermediary
Proxy Cache

Centralized Data
RDBMS, NFS, SAN
Web Architecture

Architecture is a vertical abstraction on implementation
Web protocols define that vertical abstraction on implementation
So, which one is the Web?

Browsers

Protocols

Information
So, which one is the Web?

All of them!

The Web is a World-Wide System: constantly running, always changing, anarchically accessed, and independently deployed.
Outline

1. The Story of REST
   • Early history of the Web
   • What REST is (and is not)
   • Contemporary influences

2. Work inspired by REST
   • Decentralization
   • Generalization
   • Secure computation

3. Reflections on REST
   • Investing in entrepreneurial students
   • Role of Software Engineering research

---


Why talk about my definition of REST?

Because

REST has become a BUZZWORD

There's nothing particularly wrong with that… unless you happen to be me… or working with me

What is REST Anyway?

• Representational State Transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web

• Roy Fielding looked at the Web and saw that it was good
Architectural Styles

- A horizontal abstraction across multiple architectures (vertical abstractions)
  - names a repeated architectural pattern
  - defined by its design constraints
  - chosen for the properties they induce

- REST is an architectural style
  - for network-based applications
  - to induce a specific set of architectural properties
  - that were desired for the World Wide Web
REST is an accumulation of design constraints
REST is an accumulation of design constraints

Constraint
REST is an accumulation of design constraints

- Replicated
- Separated
- Layered
- Programmable
- Uniform interface
- On-demand
- Stateless
- Intermediate processing
- Mobile
- Simple visible
- Reliable
- Local
- Scalable
- Multi-organizational
- Extensible
- Reusable
- Cacheable
- Shared

Properties:
- $property$
REST’s Five Uniform Interface Constraints

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

2. Access methods have the same semantics for all resources
   - induces visible, scalable, available through layered system, cacheable, and shared caches

3. Resources are manipulated through the exchange of representations
   - induces simple, visible, reusable, cacheable, and evolvable (information hiding)

4. Representations are exchanged via self-descriptive messages
   - induces visible, scalable, available through layered system, cacheable, and shared caches
   - induces evolvable via extensible communication

5. Hypertext as the engine of application state
   - induces simple, visible, reusable, and cacheable through data-oriented integration
   - induces evolvable (loose coupling) via late binding of application transitions
Deep dive into the hypermedia constraint

Hypertext as the Engine of Application State

- Each state can be dynamic
- Each transition can be redirected
The client only needs to know one state and its transitions!

Follow Your Nose
The client only needs to know one state and its transitions!

Follow Your Nose

ESEC/FSE'17, September 8, 2017, Paderborn, Germany
The client only needs to know one state and its transitions!

Follow Your Nose
The client only needs to know one state and its transitions!

Follow Your Nose
Outline

1. The Story of REST
   - Early history of the Web
   - What REST is (and is not)
   - Contemporary influences

2. Work inspired by REST
   - Decentralization
   - Generalization
   - Secure computation

3. Reflections on REST
   - Investing in entrepreneurial students
   - Role of Software Engineering research
WebDAV

- Distributed Authoring and Versioning
  - Returning to the Web's roots
  - Resources vs Representations vs Metadata
  - Stateless Interaction vs Session Locks
- Overwhelmed by commercial demands
  - XML, Locks, remote data model
  - Moved away from REST constraints, but helped enlighten and refine them


Dynamic Software Architectures

- How do you make adaptability easier?
  - Independent post-deployment evolvability
- Expose the application’s architecture
  - Allow third-parties to evolve application by changing architecture
- Verify changes against semantic annotations on the system model
  - with assistance of external analysis modules
  - if change is okay, apply it to the implementation; else, take appropriate action; notify, prevent, …

Outline

1. The Story of REST
   - Early history of the Web
   - What REST is (and is not)
   - Contemporary influences

2. Work inspired by REST
   - Decentralization
   - Generalization
   - Secure computation

3. Reflections on REST
   - Investing in entrepreneurial students
   - Role of Software Engineering research
New challenges for real-time network-based applications

- By 2000, there was a boom in real-time applications:
  - Push, Peer-to-Peer, Publish/Subscribe, Instant Messaging, and Internet-Scale Event Notification…
- REST had gaps for real-time:
  - One-shot: no retry if response lost
  - One-to-one: no concurrent groups
  - One-way: no asynchronous links
- Latency & Agency concerns

---


http://www.ics.uci.edu/~rohit/ARRESTED-ICSE.pdf
Computation exchange: CREST

- Technical triad of the Web:
  1. URLs name information resources
  2. Metadata used for distinguishing representations
  3. HTTP defines exchanges between clients and servers

- What happens when you generalize?
  A. CURLs name *computation resources*
  B. *Metaprogramming* is used for examining and describing computations
  C. Asynchronous protocol for *peer-to-peer* exchanges

- CREST learned about deferring code from ‘living labs’ like Subversion

  Analysis of the essential architectural decisions of the WWW, followed by generalization, opened up an entirely new space of decentralized, Internet-based applications based on computations as the fundamental concept.

CREST with security: COAST

- Capability based security model with computation exchange
- Exchange active computations among peers: Code + run-time state (reified as closures and continuations)
- Novel security mechanism: Capability URL (CURL)
  - Dictates where computations may go
  - Bounds what visiting computations can do
  - Limits resource consumption of computations
- Architectural style: ComputAtional State Transfer (COAST)
  - Build capability security into the architectural style
  - Functional capability: What can a visiting computation do?
  - Communication capability: With whom, when, and how often may that computation communicate?


COAST: Computational State Transfer

- An Architectural Style for the Idiom of Computation Exchange

1. **Service:** All services are computations whose sole means of interaction is the asynchronous messaging of values, closures, continuations and binding environments.

2. **Execution:** Each computation executes within the confines of some execution site \( \langle E, B \rangle \) where \( E \) is an execution engine and \( B \) is a binding environment.

3. **Messaging:** Computation \( x \) may deliver a message to computation \( y \) only if \( x \) holds a CURL \( u \) and \( y \) has the authority to read a message via \( u \)

   - A Capability URL (CURL) conveys the authority to communicate and is a tamper-proof cryptographic structure that cannot be forged or guessed.

4. **Interpretation:** The interpretation of a message delivered to computation \( y \) via CURL \( u \) of \( y \) is \( u \)-dependent.

5. **The Service and Messaging rules confer communication by introduction:**

   - Computation \( x \) can message computation \( y \) only if \( x \) has been introduced to \( y \) beforehand.

6. **Ab initio, endowment, Messaging, or Execution**
Outline

1. The Story of REST
   • Early history of the Web
   • What REST is (and is not)
   • Contemporary influences

2. Work inspired by REST
   • Decentralization
   • Generalization
   • Secure computation

3. Reflections on REST
   • Investing in entrepreneurial students
   • Role of Software Engineering research
Investing in entrepreneurial students, over long periods...
The Role of Software Engineering research

- **This is Software Engineering research**
  - It is about Design
  - It is fundamentally software architecture
  - It is based on reflection
  - This is how styles....good styles... get developed:
    - Experience, evaluation, reflection. Wash Rinse Repeat.

- **The research environment was “unusual”**
  - DARPA funding with a long leash (Thanks Bill and John!)
  - Lots of travel. Lots
  - A (mostly) accommodating university process
    - 9 years to Ph.D after B.S.
  - A highly interactive research community: UCI’s PhD students, W3C, IETF, and close industry links

- **Contrast with today's environment...**
The Paper

- The first version of what eventually became *Principled Design of the Modern Web Architecture* was submitted to FSE99 a year earlier.

- It was rejected, with reviewer comments including “Over all, the originality of the paper is quite low. There is only little to learn from it.” and “- the web is old technolgy [sic] now. - lots of jargon make the paper difficult to understand. ... - I can't find a novel lessons [sic] for software engineers in this paper.”

- The ICSE 2000 paper had:
  - no surveys, no statistical analyses, and essentially no evaluation section. It merely stated:
    - “The REST architectural style has been validated through six years of development of the HTTP/1.0 and HTTP/1.1 standards, elaboration of the URI and relative URL standards, and successful deployment of several dozen independently developed, commercial-grade software systems within the modern Web architecture.”

- That work, in its multiple forms, has now been cited over 8000 times...
Acknowledgments

- Funding
  - DARPA: Bill Scherlis and John Salasin; National Science Foundation; ISR’s corporate sponsors

- The Web
  - Tim Berners-Lee, Henrik Frystyk Nielsen, Dan Connolly, Dave Raggett, and Larry Masinter

- REST advocates
  - Mark Baker, Paul Prescod, Mike Amundsen, Leonard Richardson, Sam Ruby, and Aaron Swartz

- Colleagues
  - Mark Ackerman, Ken Anderson, Greg Bolcer, Eric Dashofy, Nenad Medvidovic, Kari Nies, Jie Ren, Jason Robbins, David Rosenblum, and Girish Suryanarayana.

- Thanks —
  - To the SIGSOFT community for the honor of this Impact Award.