Understanding HTTP and REST – part 2

Oxford University

Software Engineering Programme

Sep 2015

REST

- **Roy Fielding**, a principal author of HTTP 1.1
- PhD thesis *Architectural Styles and the Design of Network-based*
- Subsequent article *Principled Design of the Modern Web Architecture* (ACM TOIT 2:2, 2002)
- Richardson & Ruby, *RESTful Web Services* architectural patterns of the web
- *Software Architectures* (2000) more about evaluation than a cookbook
- Taking HTTP seriously as a distributed computing protocol: fixed few verbs, emphasis on the nouns

Client Server (CS)

- Server offers services, listens for requests
- Client sends request, waits for response
- Transient, triggering client; persistent, reactive server
- Separation of concerns: user interface from behaviour
- Improves portability to a new user interface
- Improves scalability by simplifying components
- Improves evolvability by allowing independent evolution of components

Replicated Repository (RR) and Caching ($)

- Replicated repository: multiple servers provide same service
  - Present the illusion of a single, centralized service
  - Improves performance: latency, redundancy
  - Maintaining consistency the primary challenge
- Caching: caching responses for later reuse
  - Effectively a replication of a fragment (typically, potential data set is huge or infinite)
  - Responses explicitly or implicitly labelled cacheable or not
  - Lazy or active replication
  - Less effective than full replication, but cheaper and simpler

Stateless (S)

- Each request from client must carry all necessary context
- No session state stored on server — kept entirely on client
  - ***Resource state is a different matter***
- Improves visibility for monitoring
- Improves reliability by simplifying recovery from partial failure
- Improves scalability by allowing server to free resources quickly
- Improves evolvability by simplifying server, cache
- Decreases performance by increasing overhead

Layered Systems

- Hierarchical arrangement
- Layer provides services to layer above, uses services from layer below
- Improves evolvability and reusability through abstraction
- Decreases performance through overhead, latency
- Layered-client-server (LCS) adds proxy and gateway components to CS
- Proxy acts as shared server for one or more clients, forwarding (maybe translated) requests
- Gateway appears as normal server, but forwards (maybe translated)
- Requests to lower layers: load balancing, security

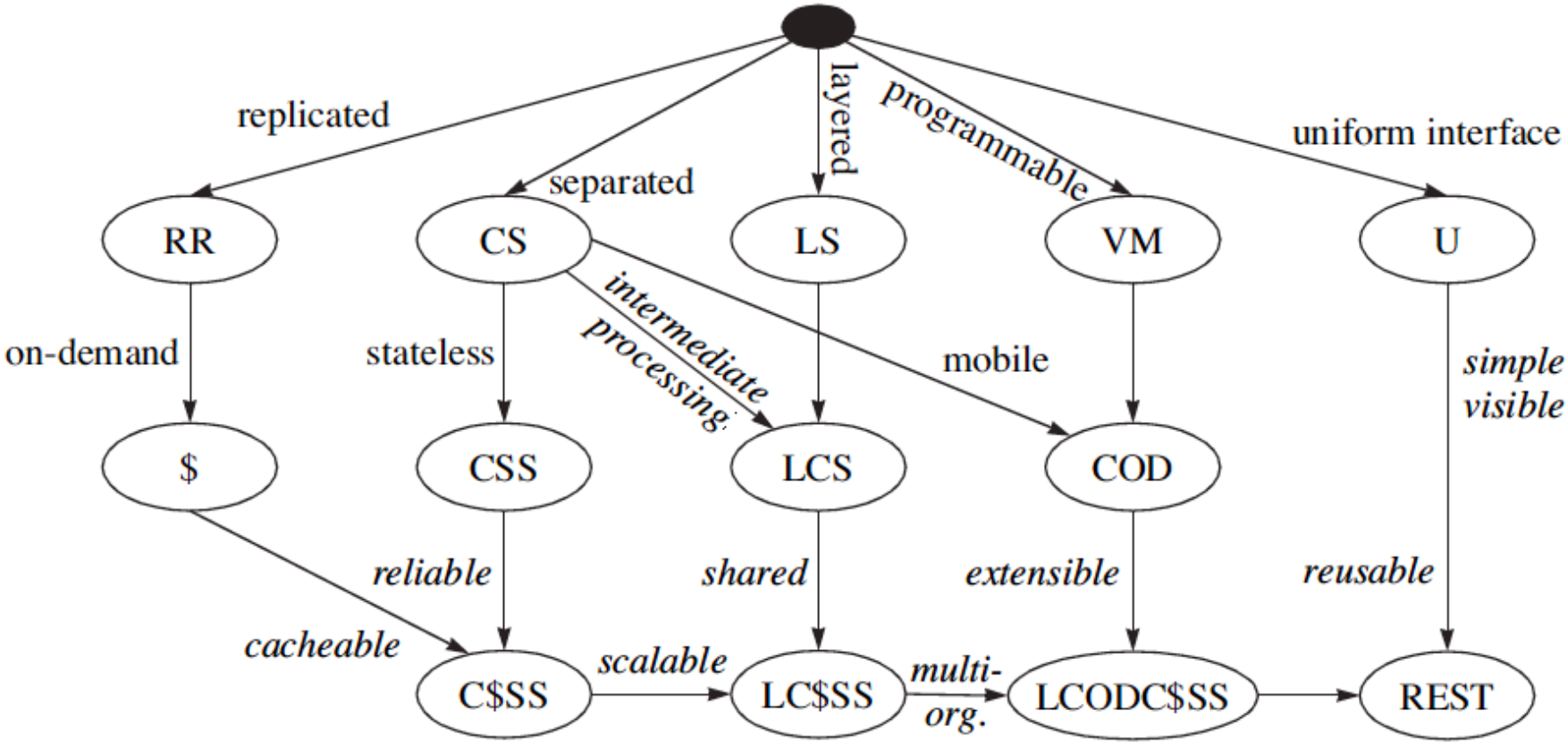
Uniform Interface

- Improves simplicity and visibility
- Decreases efficiency through possible data translations
- For REST, optimized for large-grain hypermedia data transfer
- Identification of resources
- Manipulation of resources through representations
- Self-descriptive messages
- Hypermedia as the engine of application state (more later)

Virtual Machine (VM) and Code-on-Demand (COD)

- Mobile code
- Dynamically relocate processing between data source and destination
- Improves performance by relocating code near data
- Data element must be transformed into component
- Extend client functionality by downloading applets/ scripts
- Virtual machine to provide controlled environment
- Improves simplicity and extensibility of client
- Reduces visibility
- Not a big part of REST-based SOA (yet: cf AJAX)

REST Derivation from Style Constraint



Principles of REST Architecture

- REST isn't protocol specific, but in practice means the RESTful usage of HTTP
- HTTP is actually a very rich application protocol which gives us features like content negotiation and distributed caching.
- HTTP verbs nicely map to CRUD operations of data
- RESTful web services try to leverage HTTP in its entirety using specific architectural principles.

Resources and Uniform Interface

- Addressable Resources. Every "object" on your network should have a unique ID.
- An important aspect is that each "object" or resource has its own specific URI where it can be addressed
- Anything you wish to act upon, reference, annotate, etc
- The URI should have a lifetime equivalent to the resource it represents (e.g. I've had the same bank account for 20+ years)

Representation

- State of resource captured and transferred between components
- Might be current or desired future state
- Represented as data plus metadata (name–value pairs)
- Metadata includes control data, media type
- The **Content-Type** of the resource should be useful and meaningful (self-description)
- One resource might have several representations
- Selected via separate URIs, or via content negotiation

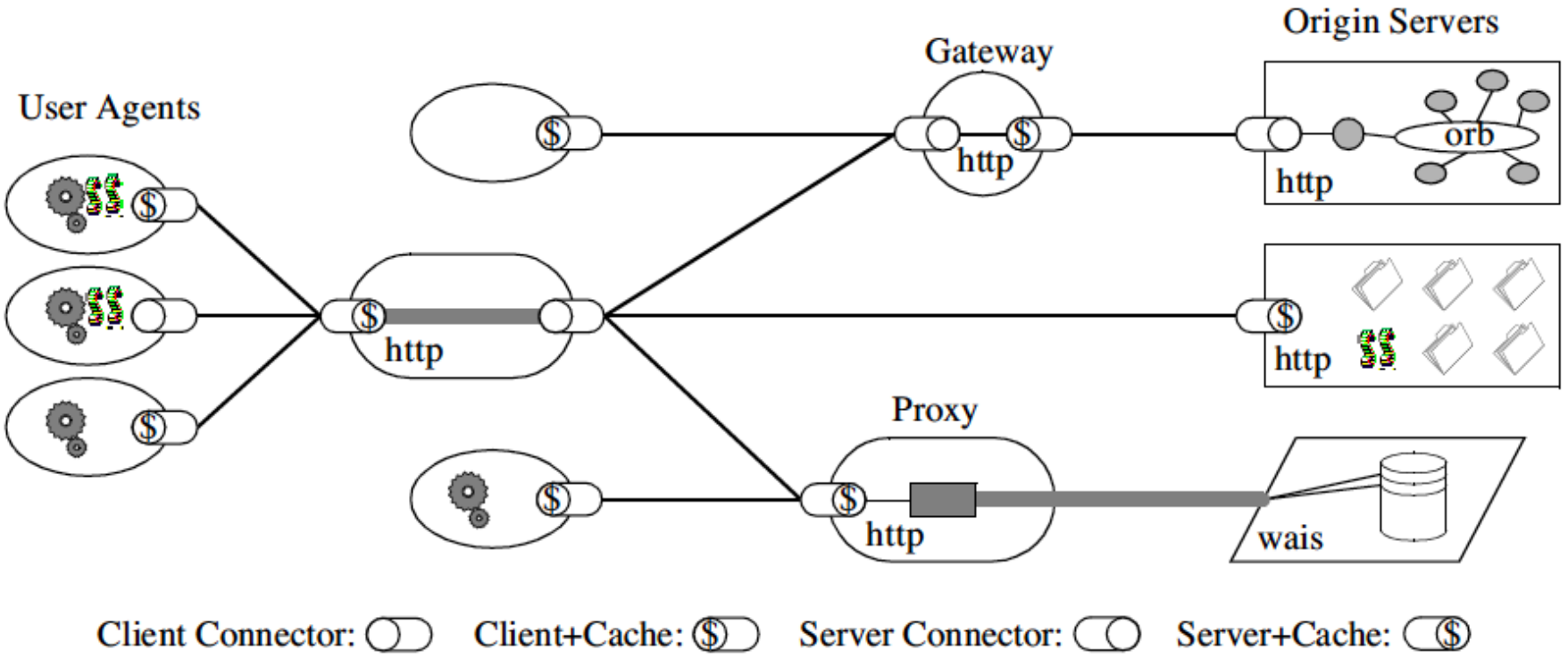
Stateless Interaction

- Abstract interface for component communication
- Stateless interactions:
  - connectors need not retain application state between requests
  - interactions can be processed in parallel, naively
  - intermediary may view and understand request in isolation
  - reusability of cached response can be determined from response itself
- Request parameters: control data, target URI, optional representation
- Response parameters: control data, optional resource metadata, optional representation

Uniform Interface

- A Uniform, Constrained Interface. When applying REST over HTTP, stick to the methods provided by the protocol
  - GET, POST, PUT, and DELETE.
- These should be used properly
  - GET should have no side effects or change on state
  - PUT should update the resource “in-place”

REST Architecture



REST Standards

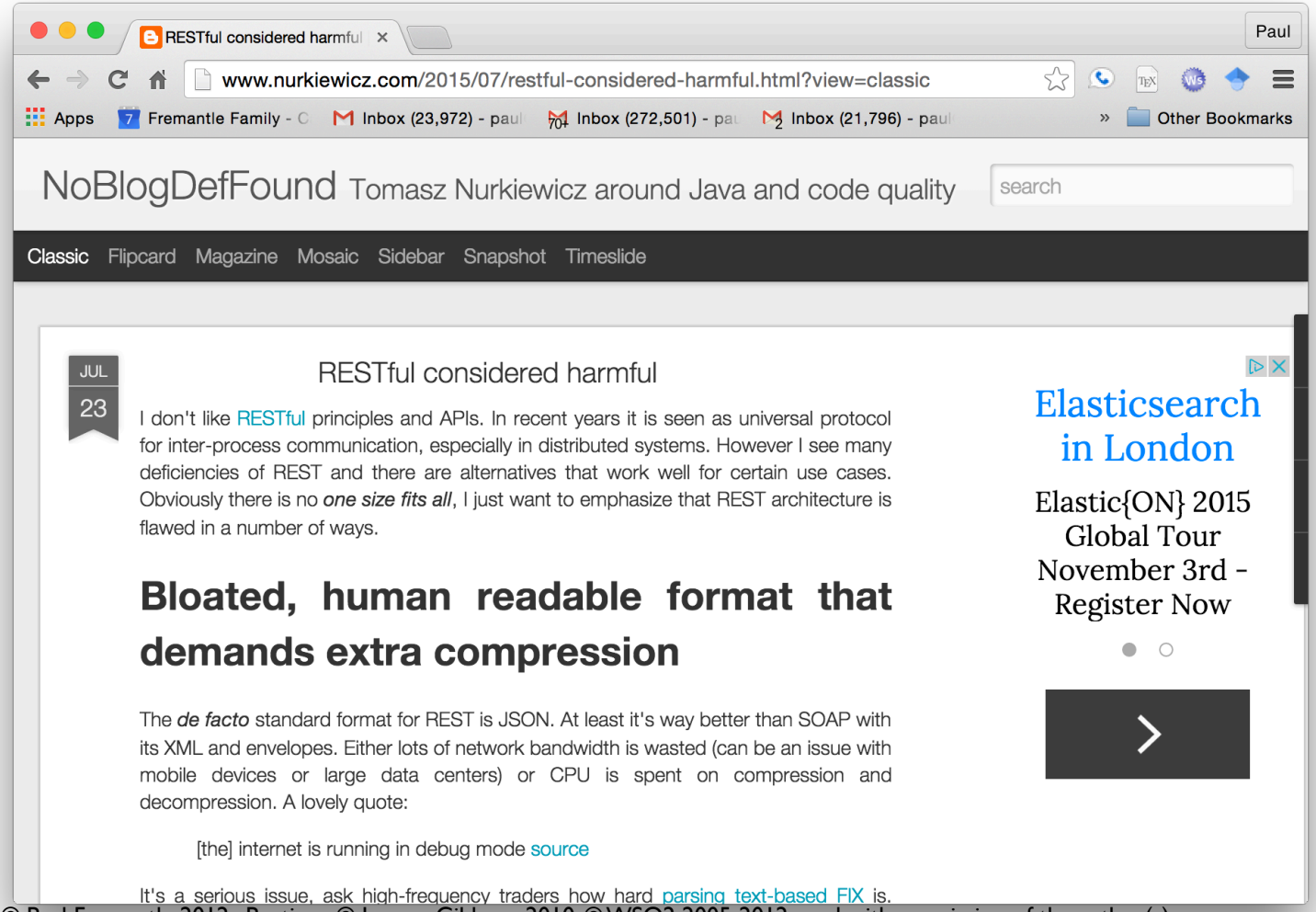
- **HTTP 1.1**
- URI
- URI Template
- **WebSockets**
- **XML, JSON, etc**
- Atom/AtomPub
- OData
- OpenId
- **OAuth 1 / 2**
- **SAML/SAML2**
- JSON Web Tokens
- WADL
- Swagger
- Json Home
- Json Web Encryption
- Json Web Signature
- Json Patch
- SPDY
- HTTPbis
- HTTP Link Header
- Microformats
- RDDL
- …

Async

- HTTP is synchronous: request–response
  - What about long-running requests? deferred synchronous interaction
- Client POSTs request (because not idempotent)
- POST /queue HTTP/1.1
- Host: jobservice.com
  - Please tell me whether 2^43,112,609 - 1 is prime
  - server queues task, returns code 202 'Accepted' with
    - URI Location: http://jobservice.com/queue/job11a4f9
    - 202 Accepted
- Client polls resource:
  - GET /queue/job11a4f9 HTTP/1.1
  - getting either status report or result
- Of course WebSockets could be used to push the response
  - Also see new Push API from W3C

Not everyone agrees:

http://www.nurkiewicz.com/2015/07/restful-considered-harmful.html



Anti-REST concerns

- Bloated formats (equally applies to SOAP)
- Neither Schema nor Contract
- APIs and discovery instead of clear published machine-readable documentation
- No inbuilt batching, paging, sorting, etc
- CRUD only
- HTTP Status codes mixed with business replies
- Temporal Coupling
- Not clear enough what is REST and what isn't!
- Backwards compatibility

Questions?