Respresentational State Transfer (REST)

Web Architecture and Information Management [./] Spring 2009 — INFO 190-02 (CCN 42509)

Erik Wilde, UC Berkeley School of Information

2009-05-04

Contents

Abstract (2)

The Web is built on an architectural style called *Respresentational State Transfer (REST)*. The main idea of this style is to use a *uniform interface* for all services, which means that each Web site provides the same service. This idea of a uniform interface is apparent in Web documents (browsers can GET documents by following hyperlinks), but also can be extended to cover machine-oriented *Web Services*. REST is a style that supports loose coupling and massive scalability, as opposed to more traditional ways of how enterprise computing attempts to integrate all funcitonality in an attempt to hide distribution.

REST: The Definition

The REST Architectural Style (4)

- A set of constraints that inform an architecture

1. Resource Identification [Resource Identification (1)]
2. Uniform Interface [Uniform Interface (1)]
3. Self-Describing Messages [Self-Describing Messages (1)]
4. Hypermedia Driving Application State [Hypermedia Driving Application State (1)]
5. Stateless Interactions [Stateless Interactions (1)]

- Claims: scalability, mashup-ability, usability, accessibility

Resource Identification (5)

- Name everything that you want to talk about
- "Thing" in this case should refer to *anything*
  - *products* in an online shop
  - *categories* that are used for grouping products
  - *customers* that are expected to buy products
  - *shopping carts* where customers collect products
- *Application state* also is represented as a resource
  - *next* links on multi-page submission processes
  - *paged results* with URIs identifying following pages

Uniform Interface (6)

- The same small set of operations applies to everything [Resource Identification (1)]
- A small set of *verbs* applied to a large set of *nouns*
- verbs are universal and not invented on a per-application base
- if many applications need new verbs, the uniform interface can be extended
- natural language works in the same way (new verbs rarely enter language)
- Identify operations that are candidates for optimization
  - GET and HEAD are *safe operations*
  - PUT and DELETE are *idempotent operations*
  - POST is the catch-all and can have side-effects
- Build functionality based on useful properties of these operations

Self-Describing Messages (7)

- Resources are abstract entities (they cannot be used *per se*)
  - Resource Identification [Resource Identification (1)] guarantees that they are clearly identified
  - they are accessed through a Uniform Interface [Uniform Interface (1)]
- Resources are accessed using *resource representations*
  - resource representations are sufficient to represent a resource
  - it is communicated which kind of representation is used
  - representation formats can be negotiated between peers
- Resource representations can be based on different constraints
  - whatever the representation is, it must support links [Hypermedia Driving Application State (1)]

Hypermedia Driving Application State (8)

- Resource representations [Self-Describing Messages (1)] contain links to identified resources [Resource Identification (1)]
- Resources and state can be used by navigating links
  - links make interconnected resources navigable
  - without navigation, identifying new resources is service-specific
- RESTful applications *navigate* instead of *calling*
  - representations [Self-Describing Messages (1)] contain information about possible traversals
  - the application navigates to the next resource depending on link semantics
  - navigation can be delegated since all links use identifiers [Resource Identification (1)]

Stateless Interactions (9)

- This constraint does not say "Stateless Applications"!
  - for many RESTful applications, state is an essential part
  - the idea of REST is to avoid long-lasting transactions in applications
- Statelessness means to move state to clients or resources
  - the most important consequence: avoid state in server-side applications
- *Resource state* is managed on the server
  - it is the same for every client working with the service
  - when a client changes resource state other clients see this change as well
- *Client state* is managed on the client
  - it is specific for a client and thus has to maintained by each client
  - it may affect *access* to server resources, but not the resources themselves
- *Security issues* usually are important with client state
  - clients can cheat by lying about their state
  - keeping client state on the server is expensive (but may be worth the price)

Web Architecture

What is the Web? (11)

- Web = URI + HTTP + (HTML | XML)
- Imagine your application being used in "10 browsers"
  - resources to interact with should be identified [Resource Identification (1)] and linked [Hypermedia Driving Application State (1)]
  - a user's preferred font size could be modeled as client state
  - what about an access count associated with an API key?
- Imagine your application being used in "10 browser tabs"
  - no difference as long as client state is representation-based
  - cookies are shared across browser windows (different "client scope")

Uniform Resource Identifier (URI)

Identifying Resources on the Web (13)

- Essential for implementing a Resource Identification [Resource Identification (1)]
- URIs are human-readable universal identifiers for "stuff"
  - many identification schemes are not human-readable (binary or hex strings)
  - many RPC-based systems do not have universally identified objects
- Making every thing a universally unique identified thing is important
  - it removes the necessity to *scope* non-universal identifiers
  - it allows to talk about all things in exactly the same way

URI Schemes (14)

```
URI = scheme ":" hier-part [ "?" query ] [ "#" fragment ]
```

- URIs in their general case are very simple
  - the scheme identifies how resources are identified
  - the identification may be hierarchical or non-hierarchical
- Many URI schemes are hierarchical
  - it is then possible to use relative URIs such as in a href="../"
  - the slash character is not just a character, in URIs it has semantics

> […] the URI syntax is a federated and extensible naming system wherein each scheme's specification may further restrict the syntax and semantics of identifiers using that scheme.

"Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, January 2005 [http://dret.net/rfc-index/reference/RFC3986]

Query Information (15)

- Query components specify additional information
  - it is non-hierarchical information further identifying the resource
  - in most cases, it can be regarded as "input" to the resource
- Query components often influence caching
  - successful GET/HEAD requests may be cached
  - only cache query string URIs when explicitly requested (Expires/Cache-Control)

> The query component contains non-hierarchical data that, along with data in the path component […], serves to identify a resource within the scope of the URI's scheme and naming authority […].

"Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, January 2005 [http://dret.net/rfc-index/reference/RFC3986]

Processing URIs (16)

- Processing URIs is not as trivial as it may seem
  - escaping and normalization rules are non-trivial
  - many implementations are broken
  - complain about broken implementations
  - even more complicated when processing an *Internationalized Resource Identifier (IRI)*
- URIs are not just strings
  - URIs are strings with a considerable set of rules attached to them
  - implementing all these rules is non-trivial
  - implementing all these rules is crucial
  - application development environments provide functions for URI handling

Hypertext Transfer Protocol (HTTP)

HTTP Methods (18)

- *Safe methods* can be ignored or repeated without side-effects
  - arithmetically safe: 41 × 1 × 1 × 1 × 1 …
  - in practice, "without side-effects" means "without relevant side-effects"
- *Idempotent methods* can be repeated without side-effects
  - arithmetically idempotent: 41 × 0 × 0 × 0 × 0 …
  - in practice, "without side-effects" means "without relevant side-effects"
- Unsafe and non-idempotent methods should be treated with care
- HTTP has two main *safe methods*: GET HEAD
- HTTP has two main *idempotent methods*: PUT DELETE
- HTTP has one main *overload method*: POST

Cookies (19)

- Cookies are *client site state bound to a domain*
  - they are convenient because they work *without having to use a representation*
  - they are inconvenient because they are *not embedded in representations*
- Cookies are managed by the client
  - they are shared across browser tabs
  - they are not shared across browsers used by the same user
  - essentially, the *client* model of cookies is a bit outdated
- Two major things to look out for when using cookies:
  1. *session IDs* are *application state* (i.e., non-resource state)
  2. cookies break the back button (requests contain a "URI/cookie" combo)
- The ideal RESTful cookie is never sent to the server
  - cookies as *persistent data storage on the client*
  - interactions with the server are only using URIs and representations

Representations

What is a URI? (21)

- Essential for implementing Self-Describing Messages [Self-Describing Messages (1)]
  - also should provide support for Hypermedia Driving Application State [Hypermedia Driving Application State (1)]
- Resource Identification [Resource Identification (1)] only talks about an *abstract resource*
  - resources are never exchanged or otherwise processed directly
  - all interactions use *resource representations*
- Representations depend on various factors
  - the nature of the resource
  - the capabilities of the server
  - the capabilities or the communications medium
  - the capabilities of the client
  - requirements and constraints from the application scenario
  - negotiations to figure out the "best" representation

Extensible Markup Language (XML) (22)

- The language that started it all
  - created as a streamlined version of SGML
  - took over as the first universal language for structured data
- XML is a metalanguage (a language for representing languages)
  - many domain-specific languages are defined as XML vocabularies
  - some metalanguages use XML syntax (RDF [Web Semantics in Practice; Resource Description Framework (RDF) (1)] is a popular example)
- XML is only syntax and has almost zero semantics
  - very minimal built-in semantics (language identification, IDs, relative URIs)
  - semantics are entirely left to the XML vocabularies
- XML is built around a tree model
  - each XML document is a tree and thus limited in structure
  - RESTful XML introduces hypermedia to turn XML data into a graph

Resource Description Framework (RDF) (23)

- Developed around the same time as XML was developed
  - based on the idea of machine-readable/understandable semantics
  - builds the *Semantic Web* as a parallel universe on top of the Web
- RDF uses URIs for *naming things*
  - RDF's data model is based on (URI, property, value) triples
  - triples are combined and inference is used to produce a graph
- RDF is a metalanguage built on the triple-based data model
  - RDF has a number of syntaxes (one of them is XML [Extensible Markup Language (XML) (1)]-based)
  - RDF introduces a number of schema languages (often referred to as *ontology languages*)

Atom (24)

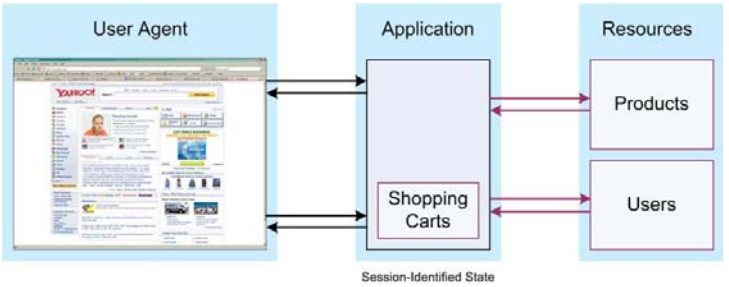
- A language for representing *syndication feeds*
- Much more modest in its goal than XML [Extensible Markup Language (XML) (1)] or RDF [Web Semantics in Practice; Resource Description Framework (RDF) (1)]
  - models feeds as a sets of entries with associated metadata
  - uses an XML vocabulary for representing the data model
  - uses *links* for expressing relationships in the data model
- The foundation for Web-scale Content Syndication [Content Syndication]

RESTful shopping

State Management on the Web (26)

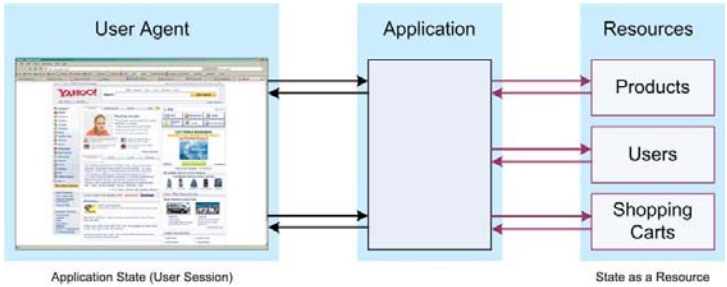
- Essential for supporting Stateless Interactions [Stateless Interactions (1)]
- State Management (Cookies) [State Management (Cookies)] are a frequently used mechanism for managing state
  - in many cases used for maintaining session state (login/logout)
  - more convenient than having to embed the state in every representation
  - some Web frameworks switch automatically between cookies and URI rewriting
- Cookies have two interesting client-side side-effects
  - they are stored persistently independent from any representation
  - they are "shared state" within the context of one browser
- Session ID cookies require expensive server-side tracking
  - not associated with any resource and thus potentially global
  - load-balancing must be cookie-sensitive or cookies must be global
- *Resource-based state* allows RESTful service extensions

State in the Server Application (27)

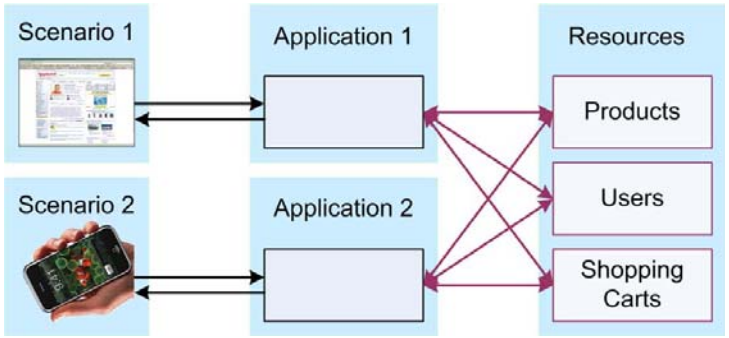


Web Architecture and Information Management [./]
Spring 2009 — INFO 190-02 (CCN 42509)
2009-05-04

State as a Resource (28)



Reusing Resources (29)



Conclusions (30)

- REST is simple to learn and use
- REST and RPC do not mix
  - resource orientation ↔ function orientation
  - cooperation ↔ integration
  - openly distributed ↔ hiding distribution
  - coarse-grained ↔ fine-grained
  - complexity in resources formats ↔ complexity in function set