Resource-Oriented Architecture

Dominic Duggan
Stevens Institute of Technology

1

1

REPRESENTATIONAL STATE TRANSFER (REST)

2

2

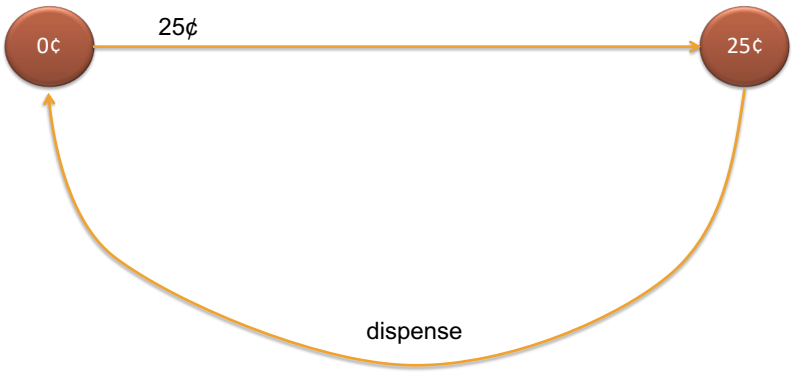
Representational State Transfer

- Software architecture for the Web
- Web browsing as navigation of hypermedia network

3

3

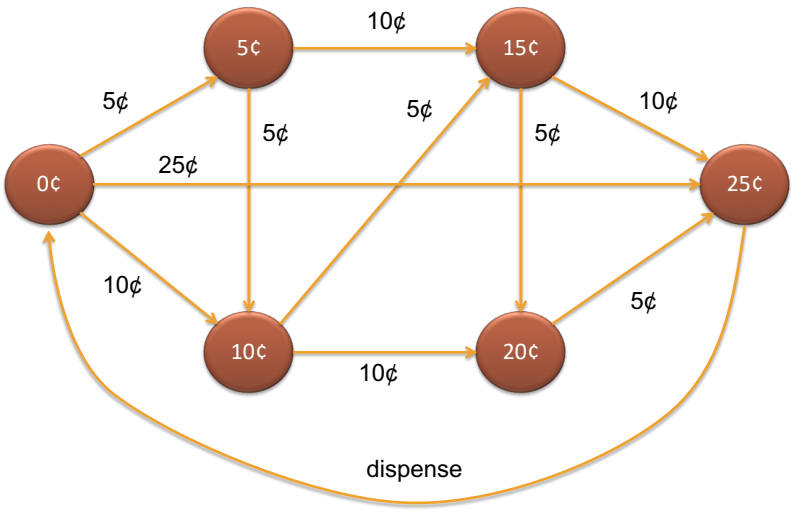
State Machine



4

4

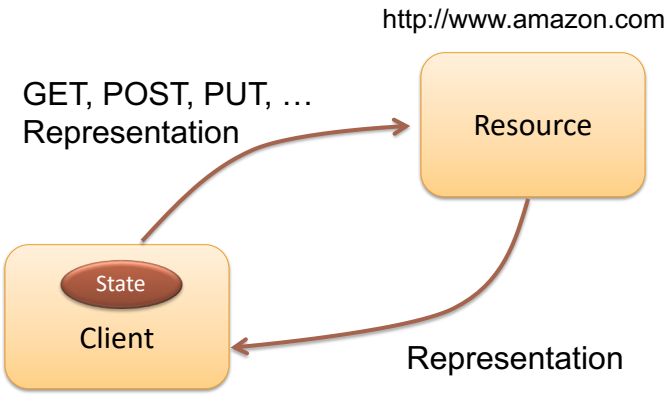
State Machine



5

5

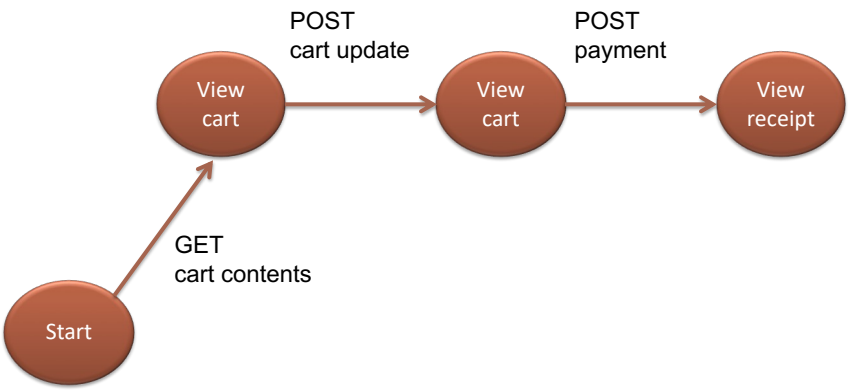
Representational State Transfer



6

6

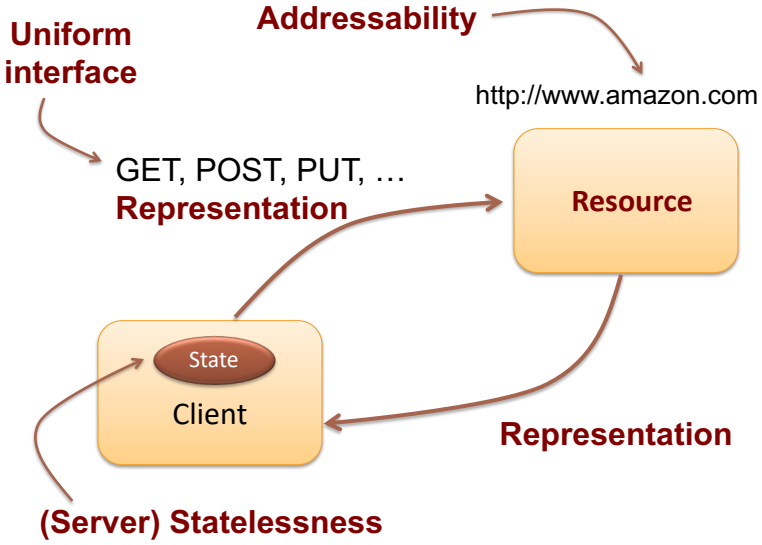
Client as State Machine



7

7

Representational State Transfer



8

8

Representational State Transfer

- Software architecture for the Web
- Web browsing as navigation of hypermedia network
  - Resources
  - Addressability: identified by URIs
    - Canonical URI, Content-Location header in HTTP
  - Representations: unit of data exchange
  - Uniform interface
  - Statelessness: all interaction state in the client
  - *Connectedness*

9

9

Representational State Transfer

SOAP / WSDL	**REST**
Service (operation) oriented	Resource oriented
One endpoint URL	URL for each individual resource
Application-defined verbs	Fixed set of HTTP verbs

- Originally a software architecture for the Web
- Emerged as an alternative architecture for Web services
  - Resource-oriented architecture

10

10

REST Verbs

- Retrieve: HTTP GET
- Create:
  - HTTP PUT for new URI or
  - HTTP POST for existing URI (server decides result URI)
- Modify: HTTP PUT, PATCH to existing URI
- Delete: HTTP DELETE

- Retrieve metadata only: HTTP HEAD
- Check which methods are supported: HTTP OPTIONS

- No other operations besides these

11

11

REST MATURITY MODEL

12

12

Level Zero: POX

- Plain Old XML
- Eschew SOAP, WSDL
- Advantage: Tunneling through firewalls
- Examples: Amazon Web Services, Flickr

13

13

Flickr API

- Flickr SOAP API

```
<env:Envelope
xmls:env="http://www.w3.org/2003/05/soap-
envelope">
  <env:Body>
    <flickr:FlickrRequest
         xmlns:flickr="urn:flickr">
      <method>flickr.photos.delete</method>
    ...
    </flickr:FlickrRequest>
  </env:Body>
</env:Envelope>
```

- Flickr “REST” API: next slide

14

14

Purpose	API calls
Get list of photos in a gallery	GET http://api.flickr.com/services/rest/? method=flickr.galleries.getPhotos &api_key=... &gallery_id=... &api_sig=...
Add a photo to a gallery	POST http://api.flickr.com/services/rest/? method=flickr.galleries.addPhoto &api_key=... &gallery_id=... &photo_id=... &api_sig=...
Create a gallery	POST http://api.flickr.com/services/rest/? method=flickr.galleries.create &api_key=... &title=... &description=... &api_sig=...
Delete a photo	POST http://api.flickr.com/services/rest/? method=flickr.photos.delete &api_key=... &photo_id=... &api_sig=...

15

15

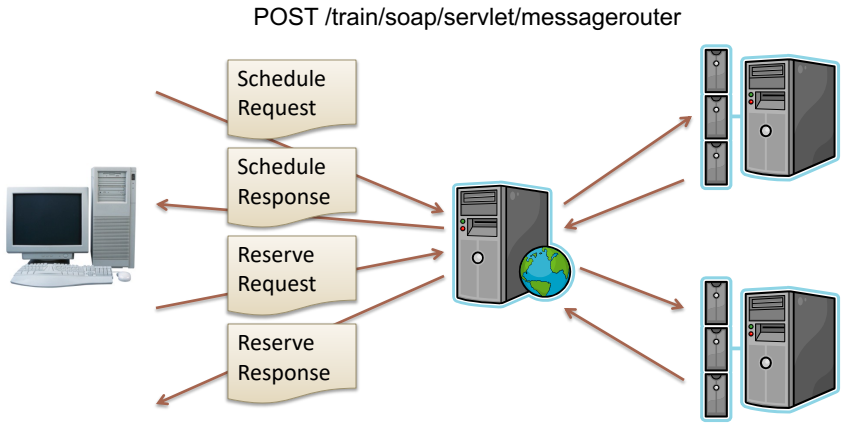
Level One: Resources

- Identify resources and their URIs
- Example: Train Reservation System
- Danger: URI tunneling

16

16

SOAP Web Service



17

SOAP Request Message

```
<soap:envelope xmlns:soap=...
  xmlns:tr=
  "http://www.example.org/schemas/train">
  <soap:body>
    <tr:ScheduleRequest>
      <tr:departure>tr:NYP</tr:departure>
      <tr:destination>tr:WAS</tr:destination>
      <tr:date>05-02-2011</tr:date>
    </tr:ScheduleRequest>
  </soap:body>
</soap:envelope>
```

18

18

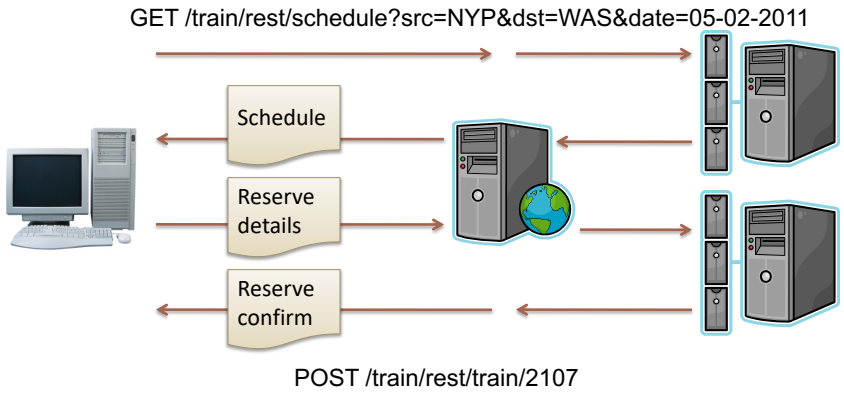
SOAP Response Message

```
<soap:envelope
    xmlns:soap=...
    xmlns:tr=
      http://www.example.org
      /schemas/train>
<soap:body>
  <tr:ScheduleResponse>
    <tr:train>
      <tr:tid>2103</tr:tid>
      <tr:time>0600</tr:time>
    </tr:train>
    <tr:train>
      <tr:tid>2107</tr:tid>
      <tr:time>0700</tr:time>
    </tr:train>
    <tr:train>
      <tr:tid>183</tr:tid>
      <tr:time>0717</tr:time>
    </tr:train>
    <tr:train>
      <tr:tid>2109</tr:tid>
      <tr:time>0800</tr:time>
    </tr:train>
    ...
  </tr:ScheduleResponse>
</soap:body>
</soap:envelope>
```

19

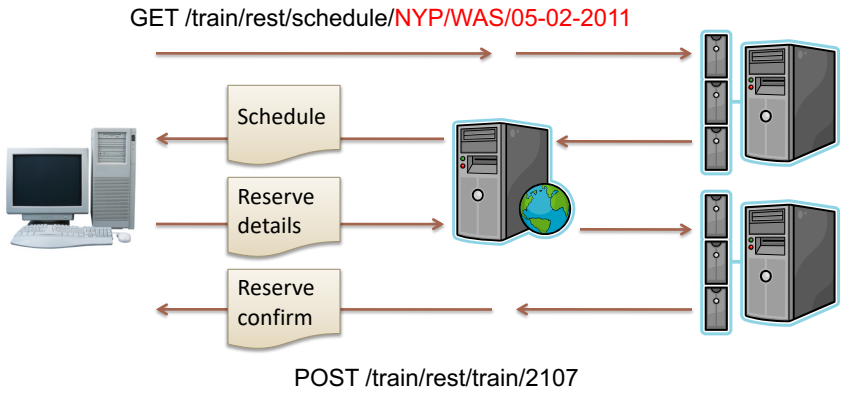
19

REST Web Service



20

Anti-Pattern: URI Tunneling



21

Level Two: Uniform Interface

- Use e.g. HTTP verbs
  - Not the only possible choice!
- CRUD interface is typical

22

22

Example: Amazon Simple Storage Service (S3)

- S3 is based on two concepts
  - Buckets
    - Named container
  - Objects
    - Named piece of data, with metadata
    - Stored in buckets

23

23

S3 RPC Interface

- Object-oriented interface to S3
  - `CreateBucket`
  - `ListAllMyBuckets`
- Getter/setter methods on bucket and object "objects"
  - `S3Object.name()`
  - `S3Object.setValue()`
  - `S3Bucket.getObjects()`

24

24

S3 REST Interface

- Three types of resources
  - List of your buckets
    `https://s3.amazonaws.com`
  - A particular bucket (virtual host)
    `https://`*`name-of-bucket`*`.s3.amazonaws.com`
  - A particular s3 object inside a bucket
    `https://`*`name-of-bucket`*`.s3.amazonaws.com/`*`name-of-object`*

25

25

S3 REST Interface

- Example:
  - A particular bucket
    `https://jeddak.s3.amazonaws.com`
  - A particular s3 object inside a bucket
    - Object names:
      docs/manual.pdf, docs/security.pdf, talks/snt.pdf
    - Resource URIs:
      `https://jeddak.s3.amazonaws.com/docs/manual.pdf`
      `https://jeddak.s3.amazonaws.com/docs/security.pdf`
      `https://jeddak.s3.amazonaws.com/talks/snt.pdf`

26

26

S3 REST Interface

- Use HTTP methods as verbs

Verb	Bucket list	Bucket	Object
GET	List buckets		
HEAD			
PUT			
DELETE			

27

27

S3 REST Interface

- Use HTTP methods as verbs

Verb	Bucket list	Bucket	Object
GET	List buckets	List bucket objects	
HEAD			
PUT		Create bucket	
DELETE		Delete bucket	

28

28

S3 REST Interface

- Use HTTP methods as verbs

Verb	Bucket list	Bucket	Object
GET	List buckets	List bucket objects	Get value and metadata
HEAD			Get metadata
PUT		Create bucket	Set object value and metadata
DELETE		Delete bucket	Delete object

29

29

Level Three: Connectedness

- Hypermedia as the Engine of Application State (HATEOAS)

30

30

Amazon S3

```
HTTP/1.1 200 OK
x-amz-id-2: ...
x-amz-request-id: ...
Date: ...
Content-Type: application/xml
Content-Length: ...
Connection: close
Server: AmazonS3
<?xml version="1.0" encoding="UTF-8"?>
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01">
  <Name>jeddak</Name>
  <Prefix>docs/</Prefix>
  <IsTruncated>false</IsTruncated>
  <Contents>
    <Key>docs/manual.pdf</Key>
    <LastModified>2006-01-01T12:00:00.000Z</LastModified>
    <ETag>&quot;...&quot;</ETag>
    <Size>20356</Size>
    <StorageClass>STANDARD</StorageClass>
    <Owner>...</Owner>
  </Contents>
</ListBucketResult>
```

31

31

SOAP Response Message

```
<soap:envelope
    xmlns:soap=...
    xmlns:tr=
      http://www.example.org
      /schemas/train>
<soap:body>
  <tr:ScheduleResponse>
    <tr:train>
      <tr:tid>2103</tr:tid>
      <tr:time>0600</tr:time>
    </tr:train>
    <tr:train>
      <tr:tid>2107</tr:tid>
      <tr:time>0700</tr:time>
    </tr:train>
    <tr:train>
      <tr:tid>183</tr:tid>
      <tr:time>0717</tr:time>
    </tr:train>
    <tr:train>
      <tr:tid>2109</tr:tid>
      <tr:time>0800</tr:time>
    </tr:train>
    ...
  </tr:ScheduleResponse>
</soap:body>
</soap:envelope>
```

32

32

REST Response Message

```
[
 {
  "href":
    "http://www.example.org/train/
     rest/train/2103",
  "time":0600
 },
 {
  "href":
    "http://www.example.org/train/
     rest/train/2107",
  "time": 0700
 },
 {
  "href":
    "http://www.example.org/train/
     rest/train/183",
  "time": 0717
 },
 {
  "href":
    "http://www.example.org/train/
     rest/train/2109",
  "time": 0800
 }
]
```

33

33

REST Response Message

```
[
 {
  "href":
    "http://www.example.org/train/
     rest/train/2103",
  "rel": ".../reserve",
  "time":0600
 },
 {
  "href":
    "http://www.example.org/train/
     rest/train/2107",
  "rel": ".../reserve",
  "time": 0700
 },
 {
  "href":
    "http://www.example.org/train/
     rest/train/183",
  "rel": ".../reserve",
  "time": 0717
 },
 {
  "href":
    "http://www.example.org/train/
     rest/train/2109",
  "rel": ".../reserve",
  "time": 0800
 }
]
```

34

34

RESOURCE ORIENTED ARCHITECTURE

35

35

Principle #1: Loose Coupling via Explicit State

- Keep application state on the client
- Violation: Session state shared between client and server
  - Send state to server on every operation
  - Identify server-side state as a resource
  - Anti-pattern: session identifier in cookie

36

36

Principle #2: Data Abstraction via Addressable Resources

- Representations decouple clients from internal details of resources
  - Expose `http://www.example.org/resource/`

    rather than

    `http://www.example.org/resource/Default.aspx`
  - URIs should also be abstract!
  - `http://www.example.org/purchases/rest/purchase/67890`

37

37

Principle #3: Canonical Expression via a Uniform Interface

- Not just a CRUD interface
  - POST on invoice resource: payment
  - DELETE on purchase resource: cancellation
- Not just HTTP!

38

38

Principle #4: Workflow Logic as Hypermedia Networks

- How can we describe behavioral contract?
- Service-oriented: control-oriented
  - Execute service operation with arguments
- Resource-oriented: data-oriented
  - Data flows between client and server

39