

**Continuations continued:
the REST of the computation**

**Anton van Straaten
appsolutions.com**

REST - The 10-second overview

- "the name given to the architecture of the World Wide Web by Roy Fielding." -- rest-discuss
- Following REST principles leads to systems with desirable properties, like scalability
- Some web systems are more RESTful than others
- RESTful systems avoid maintaining state on the server between requests

First-class Continuations

- Simplifies writing of web applications
- Write web applications in direct style - no need to 'invert control'
- Increasingly popular - in part due to prior LL workshops
- Benefits of continuations largely dependent on their ability to maintain application state on the server between requests
 - (ignore possibility of sending state to the client for now)

REST vs. Continuations

- On the server between requests:
 - REST avoids keeping state
 - Continuations keep state
- Bam! Conflict?
- "Trying to use continuations for a web application just means you don't really understand what the web is for" -- anonymous

Why does it matter?

- REST guidelines well-proven - ignore at own risk
- Continuation-based web systems increasingly popular
- Application spaces overlap
- Worth understanding apparent conflict
- Understand where first-class continuation use might be problematic
- Broaden the dialogue about continuations in practical contexts
- Get new ideas!

REST

- Again: "the name given to the architecture of the World Wide Web by Roy Fielding."
- Some context: "The first edition of REST was developed between October 1994 and August 1995, primarily as a means for communicating Web concepts as we wrote the HTTP/1.0 specification and the initial HTTP/1.1 proposal."
- "The name 'Representational State Transfer' is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through the application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

Major REST components

- Data elements
 - Resource : the intended conceptual target of a hypertext reference
 - Resource identifier : URL, URN
 - Representation : XML or HTML document, JPEG image
- Components
 - origin server : Apache httpd, Microsoft IIS
 - user agents: Mozilla, IE, etc.
 - other client programs
 - gateways, proxies, and caches

REST Benefits

- "...an architecture that separates server implementation from the client's perception of resources, scales well with large numbers of clients, enables transfer of data in streams of unlimited size and type, supports intermediaries (proxies and gateways) as data transformation and caching components, and concentrates the application state within the user agent components."

REST Caveats

- "REST is not intended to capture all possible uses of the Web protocol standards."
- "REST isn't supposed to be a baseball bat; it is supposed to be a guide to understanding the design trade-offs, why they were made, and what properties we lose when an implementation violates them."

Context for this talk

- Focusing on a particular area of REST of interest w.r.t. continuations: the "statelessness" constraint
 - REST servers avoid maintaining state between requests
- Focus on the WWW as a concrete example, although REST is theoretically a broader model.
- REST describes "distributed hypermedia systems". Talk focuses on distributed applications, with an expedient focus on web applications.

Continuation-based web systems

- Lisp - ViaWeb, UnCommon Web
- Scheme - PLT, SISC, Chicken, PS3I
- Smalltalk - Seaside
- Javascript/Java - Apache Cocoon
- Python - Impostor
- Ruby - Borges
- Hacks
 - Java - ATCT, RIFE
 - Perl: Continuity, Contize, Coro

Continuation accolade #1

"Wow!

"Seaside makes writing web apps startlingly simple and straightforward. In comparison, web app frameworks in the Java world, such as JSP or Struts, and even Ruby on Rails look like dinosaurs."

- Nat Pryce, on Avi Bryant's blog

Continuation accolade #2

"Whoever invented the flow, whoever implemented it here, and had anything to do with it and brought it to Cocoon is a F#\$%ING GENIOUS.

"I shall erect an altar in my fireplace and burn incense to his holiness every single day, preaching for my soul, to be, at one day, of comparable intelligence with his...

"PS Can you tell that I started using it? And that it did beat the crap out of me? Now I won't be able to design any web application without it! And that's the beautiful part of it!:-)"

- Pier Fumagalli on xml-cocoon-dev

REST & abstract continuations

- Forget first-class continuations -
- Do abstract continuations apply to REST?
- If not, continuations as an abstraction are suspect.
- Is something else needed - pi calculus?
- Would be unfortunate:
 - Continuations have gained mindshare
 - Already present in real tools

Analyzing REST control flow

- Identify the abstract continuations in REST
- Figure out whether reifying them is of any use

Abstract case is obvious

- Every request involves continuations
- Request is a continuation in action
- Server sends continuation(s) to client in every representation
- Example of continuation embedded in code sent to client:
Widget
- Simple, but key concept

Addresses alone are not enough

- Consider the URI as an address to 'goto'
- What about the continuation's state?
- Stored in HTML forms:

```
<form action="http://www.example.com/widget" method="POST">  
<input type="text" name="item-no" /> <input type="text"  
name="description" /> <input type="hidden" name="internal"  
value="relevant stuff" /> ...</form>
```

Continuations in REST: Case #1

- Continuation-based frameworks already support these
- They synthesize URIs for embedding in representations

```
(send-suspend
  (lambda (k-url)
    (quasiquote
      (html
        (body
          (a (@ (href (unquote k-url))))))))))
```

Analysis of Case #1

- Continuation is serialized in representation
- But not a value of type 'continuation' in host language
 - (could be considered such in HTML)
- Conceptualizing as continuation is useful
- Better abstraction from underlying mechanism

Case #2:

Serialize complex continuations to client

- Serialize complex server continuation, send to client
- Quite viable in some situations; been done
- Size of continuations may be an issue
- Satisfies REST:
 - "each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client."

First-class closures as continuations

- Paul Graham used in ViaWeb
- Continuations capture lexical context, just like closures.
- Difference:
 - "A closure needs to retain information about lexical variables, but not return-address information. A continuation needs both."
-- Guy Steele, II-discuss
- To fake continuations with closures, use CPS
- Last operation in closure: invoke the next operation

Case #3: send/suspend/dispatch

- Idiom & function by Pete Hopkins
- Supports providing multiple URIs to client
- Each URI associated with closure
- See also Waterken's Web Calculus
- Real, REST-compatible, first-class continuations
- But faked via closures - local CPS style

send/suspend/dispatch example

```
(define dialog/ask
  (lambda (title question)
    (send/suspend/dispatch
      (lambda (embed/url)
        (gen-web-page
          title
          (quasiquote
            ((unquote question)
             (p
              (a ((href (unquote
                        (embed/url (lambda _ #t))))))
                 "Yes" )
              (a ((href (unquote
                        (embed/url (lambda _ #f))))))
                 "No" ))))))))))))
```

Multiple control views of a distributed system

(informal)

- Server control flow
- Client control flow
- Client/server control flow
 - Client invokes function on server
 - Function springs to life & executes
 - Terminates with result and continuation(s)

One program, or many programs?

- Can restart server, and clients can continue where they left off
- No surprise - REST continuations in action
- Continuations (URIs identifying resources) glue application components together

Important goal of REST style

- Strip continuations down to simple URIs, or URI + HTML form
- Let client manage these continuations
- Applications become a state machine laid out as a web of interconnected resources
- "Flow" of the application determined by, and stored in, user interface
 - (actually stored in representations)

Real applications may need more

- "Sessions" are allowed in REST, subject to constraints
- Multi-step workflows a classic case

Can we keep continuations on the server?

(between requests)

- Continuations have state, so answer is no!
- "communication must be stateless in nature ... such that each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server."

Can we keep continuations on the server?

(between requests)

- Continuations have state, so answer is no!
- "communication must be stateless in nature ... such that each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server."

Misleading!

Can we keep continuations on the server?

(between requests)

- Continuations have state, so answer is no!
- "communication must be stateless in nature ... such that each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server."

Misleading!

- REST requests exploit stored context on the server all the time

Can we keep continuations on the server?

(between requests)

- Continuations have state, so answer is no!
- "communication must be stateless in nature ... such that each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server."

Misleading!

- REST requests exploit stored context on the server all the time
- Stored context is called "resources"

REST workflow

- Preferred REST approach for workflow is to accumulate state in resources
- Workflow involves creating new resources, and mutating existing ones

So: can continuations be resources?

- "Any concept that might be the target of an author's hypertext reference must fit within the definition of a resource."

So: can continuations be resources?

- "Any concept that might be the target of an author's hypertext reference must fit within the definition of a resource."
- In continuation-based web systems, continuations are the target of a hypertext reference

So: can continuations be resources?

- "Any concept that might be the target of an author's hypertext reference must fit within the definition of a resource."
- In continuation-based web systems, continuations are the target of a hypertext reference
- Typical example (PLT):
 - http://www.example.com/widget.scm;id10*k1-877162241

So, are continuations resources?

- As targets of hypertext reference, they qualify by definition

So, are continuations resources?

- As targets of hypertext reference, they qualify by definition
- QED.

So, are continuations resources?

- As targets of hypertext reference, they qualify by definition
- QED.
- OK, too cute!



So, are continuations resources?

- As targets of hypertext reference, they qualify by definition
- QED.
- OK, too cute!
- REST imposes other constraints.



Why aren't continuations resources?

- Often ephemeral - don't survive session

Why aren't continuations resources?

- Often ephemeral - don't survive session
 - "Easy" solution - persistent continuations

Why aren't continuations resources?

- Often ephemeral - don't survive session
 - "Easy" solution - persistent continuations
- But other issues, too
- E.g. continuations tend to be opaque

Why aren't continuations resources?

- Often ephemeral - don't survive session
 - "Easy" solution - persistent continuations
- But other issues, too
- E.g. continuations tend to be opaque
- Not a good primary resource format

So what are they good for?

- Need a little leap

So what are they good for?

- Need a little leap
- Snapshot program state prior to generation of representation
- Act as cache for the representation of a resource
- Accessed via ordinary URI
 - Cache manager uses continuation if present
 - Otherwise, reifies resource

Why not just cache representation?

- Caching representations is an important aspect of REST
- Not sufficient in complex workflow scenario
 - Workflow steps may need data not present in resource representation
 - Resource representation may not be primary source used by application logic
- Re-reification of resources at each step inefficient
- Slower user experience between pages, heavier server load

Case #4: Continuations as cached representation

generators

- Satisfies most REST constraints
- Conceive as:
 - Resource state
 - Pointer to code which generates representation

Cached representation generator example

```
(define (widget)
  (let* ((request (send/suspend widget-form))
        (item-no (get-value request 'item-no))
        (qty      (get-value request 'description))
        (res-id   (cache-resource-here)))
    (widget-view res-id item-no description)))
```

Low-level framework API: cache-resource-here

```
(define (cache-resource-here)
  (let ((res-id #f))
    (resume-kont
      (call/cc
        (lambda (k)
          (set! res-id (cache-resource k))
          (redirect-to
            (make-kont-uri
              (base-uri (servlet-context-url (ctx)))
              res-id)
              301))))))
  res-id))
```

Higher-level example

```
(define (widget)
  (let ((request (send/suspend widget-form)))
    (let-resource ((widg (make-widget request)))
      (purchase-order-view widg))))
```

Back to high-level view

- Web application as coroutines
- Good model, but too simplistic in reality
- Clients and servers don't always just wait for each other
- Support client operations while server busy

Final example before I can REST

- Example: long-running report, REST style
- Client sends request to server
- Server queues request, responds with resource id
- Later, client requests resource using id
- Server reifies resource, generates representation, sends to client

Case #5: representation generator redux

- Server creates stub resource, returns id to client
- Once final resource has been generated, server snapshots continuation
 - uses **cache-resource-here**
- When client requests resource:
 - Server activates cached continuation
 - Response is immediate

Concerns

- Persistence
- Scalability
- Efficiency
- Cache coherence
 - Framework can help
 - Workflows have well-defined dependencies
 - Server complexity - slightly suspect from REST perspective
 - Then again, cache management is explicit part of REST

Benefit

Cleanly addresses known REST tradeoff:

- "Like most architectural choices, the stateless constraint reflects a design trade-off. The disadvantage is that it may decrease network performance by increasing the repetitive data (per-interaction overhead) sent in a series of requests, since that data cannot be left on the server in a shared context." -- REST 5.1.3
- Address this not by saving "bad" inter-request state, but by caching resource state - more RESTful

Lessons - Conceptual

- Continuations are a good abstract model for web applications, REST or not
- Recognizing the abstract continuations in distributed systems can clarify thinking about control flow, REST applications included
- Supports abstraction of simple REST-style continuations from details of URIs

Lessons - Language Design

- Framework designers need building blocks
- Continuations are an important one
- Being forced to hack these things in is bad, 'mkay?
- Cases in point: Java, Perl
- Frameworks need first-class continuations (or equivalent) at some level
- Desired qualities
 - Persistable
 - Efficient
- Languages must provide these features

Lessons - REST

- Distributed applications are complex (duh)
- REST is a response to this complexity
- Some constraints are fundamental to distributed applications
- Others are a response to existing technology
- Tradeoff of some server complexity for continuation features may be worthwhile
 - Plenty of precedent with server-side caching - e.g. memcached, JCS
 - Storage is cheap

Time to give the puns a REST

Thank you!