COCOON GT 2007

ROME 3-4-5 OCT

Atomized Cocoon

Ugo Cei, Cocoon GetTogether 2007

sourcesense

What is Atom

- Atom is a syndication format
  - RFC 4287
- Atom is a publishing protocol
  - draft-ietf-atompub-protocol-17.txt

Atom as a Syndication Format

- Analogous to RSS but arguably “better”, as in:
  - Less ambiguous
  - Richer

Problems with RSS 2.0

Hat tip: Dave Johnson

- Spec is too loose and unclear:
  - What fields can be escaped HTML?
  - How many enclosures are allowed per element?
- Content model is weak:
  - No support for summary and content.
  - Content-type and escaping not specified.
- RSS Board not allowed to clarify specification.

Atom Content Types

- Plain text

```
<content type=”text”>Some Text</content>
```

- Escaped

```
<content type=”html”>
  &lt;div>...&lt;/div>
</content>
```

- Well-formed

```
<content type=”xhtml”>
  <div>...</div>
</content>
```

Atom Content Types

- XML
```
<content type=”application/rdf+xml”>
  <rdf:RDF>...</rdf:RDF>
</content>
```
- External
```
<content type=”application/xyz”
  src=”http://example.com/whatever”/>
```

Atom as a Publishing Protocol

- “Application-level protocol for publishing and editing Web resources using HTTP”
- Based on Atom Syndication Format.
- Began as a replacement for the old XML-RPC based blog APIs.
- Embodiment of the REST principles.

Atom Publishing Protocol

```
GET /entries HTTP/1.1

200 OK
Content-Type: application/atom+xml;type=feed


<a:feed>
  <a:entry>
    <a:link rel='self' href='/entries/1' />
    ...
  </a:entry>
  ...
</a:feed>
```

Atom Publishing Protocol

```
GET /entries/1 HTTP/1.1

200 OK
Content-Type: application/atom+xml


<a:entry>
  <a:link rel='self' href='/entries/1' />
  <a:link rel='edit'
      href='/entries/1/edit' />
  ...
</a:entry>
```

Atom Publishing Protocol

```
POST /entries HTTP/1.1
Content-Type: application/atom+xml

<a:entry>
  <a:title>An entry</a:title>
  ...
</a:entry>
```

```
201 Created
Location: http://example.com/entries/2
```

Atom Publishing Protocol

```
PUT /entries/2;edit HTTP/1.1
Content-Type: application/atom+xml

<a:entry>
  <a:title>Updated entry</a:title>
  ...
</a:entry>
```

```
200 OK
<a:entry>
  ...
</a:entry>
```

Atom Publishing Protocol

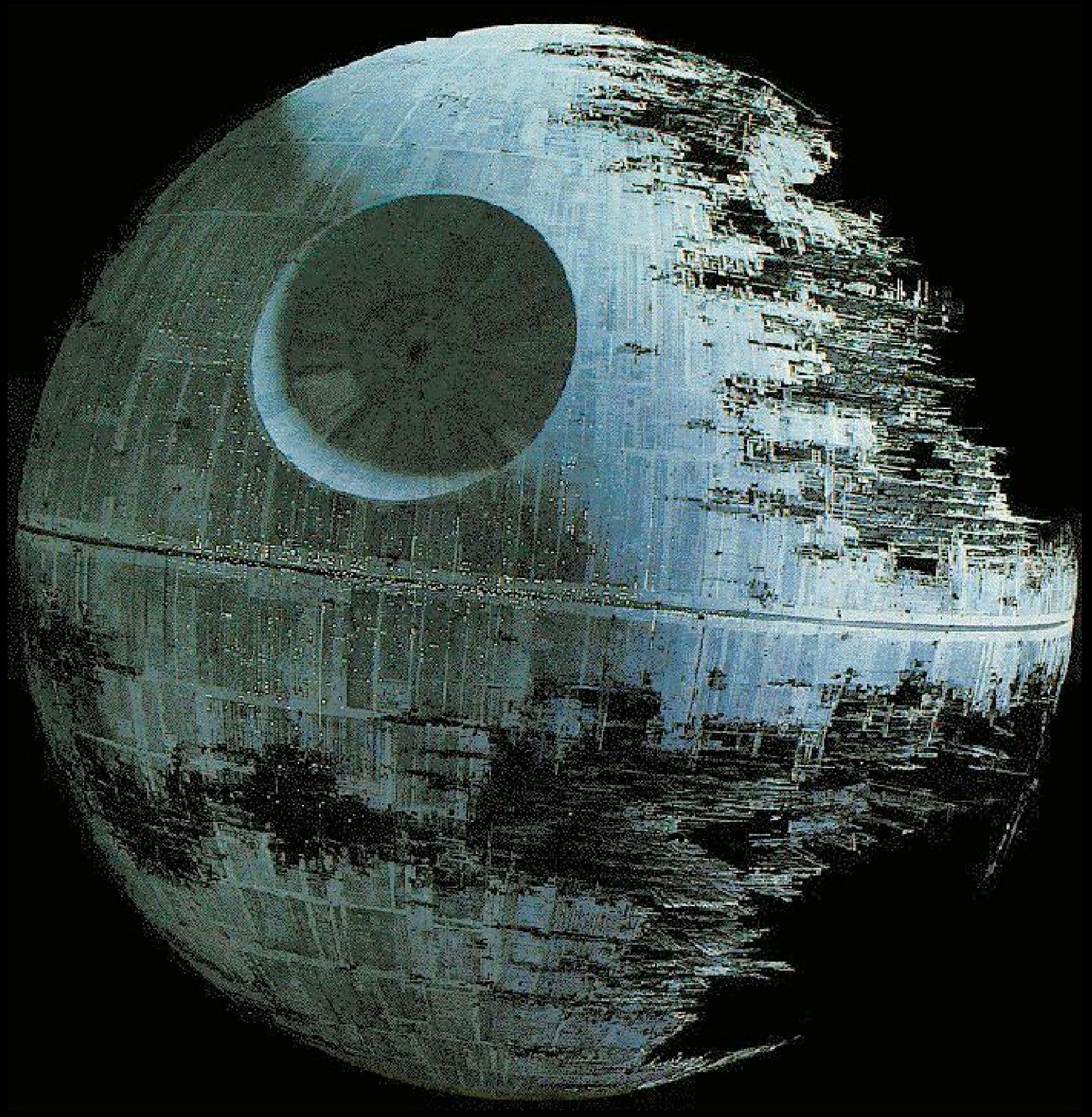
```
DELETE /entries/1 HTTP/1.1

200 OK
```

The Service Document

```
<service>
  <workspace>
    <atom:title>Main Site</atom:title>
    <collection href="http://e.org/entries">
      <atom:title>Blog Entries</atom:title>
      <accept>application/atom+xml;type=entry</accept>
    </collection>
    <collection href="http://e.org/pix">
      <atom:title>My Pictures</atom:title>
      <accept>image/*</accept>
    </collection>
  </workspace>
</service>
```

Why Atom Matters

WS-



Why Atom Matters

Web Services for the Real World
RESTful Web Services
Leonard Richardson & Sam Ruby
Foreword by David Heinemeier Hansson
O'REILLY

Why Atom Matters

- Blogger and Wordpress support Atom and APP, but blogging is just part of the story.
- A new foundation for REST based web services:
  - Google DATA API.
  - Lucene Web Services.
  - Ning.
  - Lotus Connections.

Cocoon and Atom

- Is Cocoon a good choice for an Atom Server implementation?

Implementation

- Uses Apache Abdera.
- Custom generators for serving feeds and entries.
- Custom transformer for handling submitted entries.
- Partial implementation available for Cocoon 2.2 in the whiteboard.

Apache Abdera

- "The goal of the Apache Abdera project is to build a functionally-complete, high-performance implementation of the Atom Syndication Format and Atom Publishing Protocol specifications."

Apache incubator

incubator.apache.org/abdera

Generating an Atom Document

```
<map:generator name="atomsvc"
  src="com.sourcesense.gt07.generators.AtomServiceGenerator"/>

<map:match pattern="">
  <map:generate type="atomsvc" src="demo"/>
  <map:serialize type="atomsvc"/>
</map:match>
```

SAX vs. StAX

- SAX is *push*, StAX is *pull*.
- Abdera uses AXIOM, which sits on top of StAX.
- AXIOM is somewhat limited, e.g. it only supports serializing to *XMLStreamWriter*, not *XMLEventWriter*.
- Need adapter from *XMLStreamWriter* to *ContentHandler*.

Parsing an Atom Document

```
<map:transformer name="create-entry"
    src="com.sourcesense.gt07.transformers.
        CreateAtomEntryTransformer"/>

<map:match pattern="*/entries/">
  <map:select type="method">
    <map:when test="POST">
      <map:generate type="stream"/>
      <map:transform type="create-entry" src="demo/{0}"/>
      <map:serialize type="atomentry" status-code="201"/>
    </map:when>
    ...
```

SAX vs. StAX

- For parsing submitted documents we need to buffer SAX events using SaxBuffer.
- Then we need an adapter from SaxBuffer to *XMLStreamReader*.

Testing it

```
curl -v -H 'Content-Type: application/xml'
  -d @test-data/entry.xml
  http://localhost:8888/GT07/blog1/entries/
```

Content type should really be:

application/atom.xml;type=entry

But *StreamGenerator* does not like it.

Updating an Atom Entry

```
<map:match pattern="*/entries/*">
  <map:select type="method">
    <map:when test="PUT">
      <map:generate type="stream"/>
      <map:transform type="create-entry" src="demo/{0}"/>
      <map:serialize type="atomentry" status-code="200"/>
    </map:when>
    ...
```

Can we do better?

- Abdera defines an abstract Atom Object Model that is not tied to AXIOM.
- The AXIOM-based one just happens to be the only available implementation.
- Nothing forbids anyone from writing a SAX-based implementation of it.

Pull-based Cocoon?

"BTW, I started to sketch a proof of concept pull-based pipeline API and will commit it to whiteboard/cong once I've managed to set up a basic file/xslt/html pipeline with some content inspection in the middle."

Sylvain Wallez on cocoon-dev, Thu, 08 Dec 2005

Resources

- Entry Resource - Members of a Collection that are represented as Atom Entry Documents, as defined in [RFC4287].
- Media Resource - Members of a Collection that have representations other than Atom Entry Documents.

Media Resources

“A client can POST Media Resources as well as Entry Resources to a Collection. If a server accepts such a request, then it MUST create two new Resources - one that corresponds to the entity sent in the request, called the Media Resource, and an associated Member Entry, called the Media Link Entry. Media Link Entries are represented as Atom Entries, and appear in the Collection.”

draft-ietf-atompub-protocol-17.txt

Media Resources

- No easy way to accept a POSTed media resource, create the corresponding entry and returning it.
- Cocoon Reader lacks a symmetric (Writer?) counterpart.

Conclusions

- In order to implement a conformant Atom server with Cocoon we need:
  - A small fix to StreamGenerator.
  - A component for reading POSTed non-XML content.
  - A SAX-based Atom Protocol implementation.

More Conclusions

- In order for Cocoon to be a better REST framework it needs to:
  - embrace HTTP
  - be more symmetrical
- Sample code can be found at:

  http://svn.apache.org/repos/asf/cocoon/whiteboard/gt07/ugocei-atomized/