Java™ Technology and REST: Implementing the Atom Publishing Protocol

Dave Johnson

Staff Engineer
Sun Microsystems
http://rollerweblogger.org/page/roller

TS-1756

Copyright © 2006, Sun Microsystems Inc., All rights reserved.

Java™ Technology and REST

Implementing the Atom Publishing Protocol

Understand REST by taking a close look at the new IETF Atom Publishing Protocol

Learn how to implement Atom protocol with Java technology and open source tools

Java Technology and REST: Agenda

Introduction to REST

The REST Versus WS-* Debate

Introduction to Atom Protocol

Exploring Atom Protocol

Implementing an Atom Protocol Server

Implementing an Atom Protocol Client

Summary

Lightning Introduction to REST

- Representational State Transfer (REST)
  - From Roy Fielding's doctoral thesis
- It's the inherent architectural style of the web
  - Everything is a resource
  - Every resource is addressable by URI
  - Operations on resources defined by HTTP verbs
- RESTafarians want web services to use and not abuse the architecture of the web

Developer View of REST

- Instead of doing these things:
  - Using SOAP, WSDL and the WS-* stack
  - Modeling web services as remote procedure calls (RPC)
- You do these things:
  - Use HTTP and XML (and other formats) directly
  - Model your web service as URI addressable resources
  - Use HTTP verbs to implement CRUD

Java Technology and REST: Agenda

Introduction to REST

The REST Versus WS-* Debate

Introduction to Atom Protocol

Exploring Atom Protocol

Implementing an Atom Protocol Server

Implementing an Atom Protocol Client

Summary

Advantages of the WS-* Stack

- Industrial strength standards for everything!
  - Security, reliable messaging, transactions, etc.
  - Metadata, business process, management, etc.
- Well known
  - Lots of tool support
  - Many books, articles, and training courses available
- IDE friendly/enables nice tooling
  - For example: proxy generation via WSDL

Disadvantages of the WS-* Stack

- Those specifications are big and complex
  - Requires complex tools and frameworks
  - Perhaps the stack vendors like it that way?
- Default is to abuse/ignore the web's architecture
- RPC encourages too much granularity
  - That's why you should prefer Doc-Literal

Tim Bray's WS-Pagecount

Security	230 pages
Reliable messaging	21 pages
Transactions	39 pages
Metadata	111 pages
Messaging	211 pages
Management	23 pages
Business process	74 pages
Specification profiles	74 pages
Total	**783 pages***

* and that's not counting XML and XML schema specifications, add 599 pages

WS-* Default is to Abuse the Web

- SOAP violates first axiom of web architecture
  - "Any resource anywhere can be given a URI"—T.B.L.
  - Everything goes through one URI
- SOAP operations all use HTTP POST
  - Including those that GET, PUT and DELETE
- SOAP uses HTTP as transport protocol
  - When it's really an application protocol with well-known semantics like SMTP, POP, FTP, IRC, etc.

Pros and Cons of REST

- Pros
  - Simplicity: “just” XML and HTTP
  - Minimizes dependencies (in terms of tools, specs ,etc.)
  - Allows web caches to work properly
  - Easily debugged using simple command line tools
  - Different URIs enable easy distributed processing
- Cons
  - No common service specification format (e.g. WDSL)
  - No common means for service lookup (e.g. UDDI)

Summarizing the REST vs. WS-* Debate

- Is it just simplicity versus complexity?
  - Yes, WS-* has larger/complex dependencies
  - No, good tools make using SOAP very easy
- Is it just RPC versus “just use XML”?
  - Yes, we typically use SOAP for RPC style web services
  - No, with SOAP Doc-Literal you can “just use XML”
- Is it about using HTTP properly?
  - Yes, by default SOAP misuses HTTP
  - No, SOAP can be made RESTful (see JAX-WS and Axis2)

Java Technology and REST: Agenda

Introduction to REST

The REST Versus WS-* Debate

Introduction to Atom Protocol

Exploring Atom Protocol

Implementing an Atom Protocol Server

Implementing an Atom Protocol Client

Summary

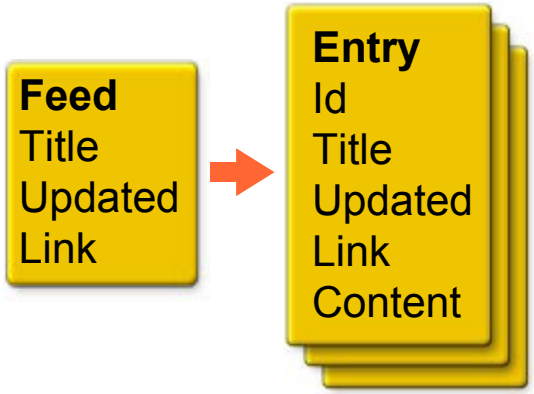
What Is Atom?

- From the Atom WG charter:

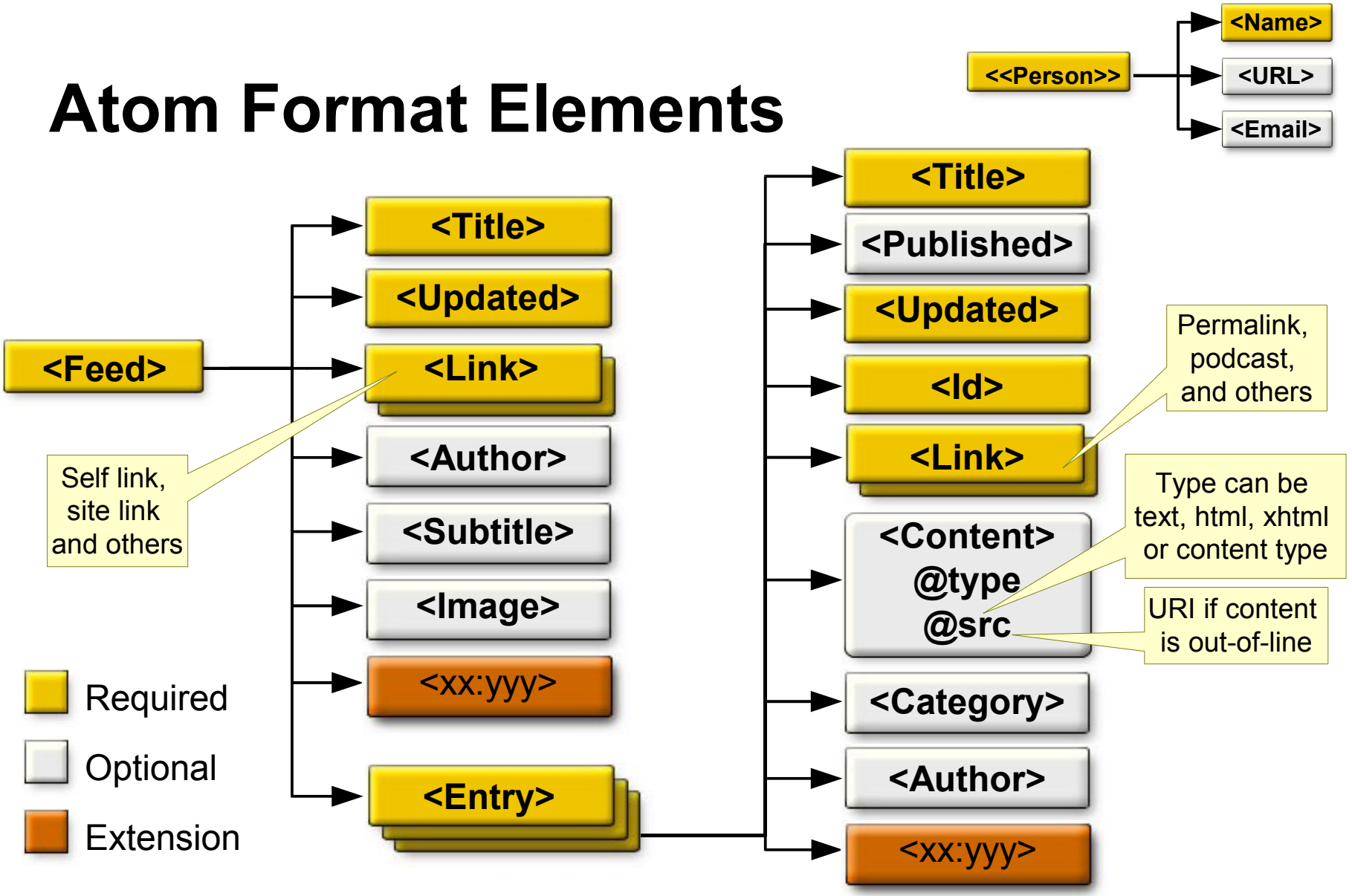
  Atom defines a feed format for representing and a protocol for editing Web resources such as Weblogs, online journals, Wikis, and similar content
- Feed format is now an IETF standard: RFC-4287
- Protocol will be finalized in 2006

Atom Publishing Format

- XML feed format
- A feed contains Entries
  - Entry is a time-stamped, uniquely identified item of data
  - Entry can carry anything
    - In-line content
    - Out-of-line content
    - TEXT, HTML, XHTML, XML
    - Binaries
    - Any content type
- It's generic, not just for blogs



Atom Format Elements



Atom <feed> with One <entry>

```
<?xml version='1.0' encoding='UTF-8'?>
<feed xmlns='http://www.w3.org/2005/Atom'
    xml:lang='en-us'>
    <title>Oh no, Mr. Bill</title>
    <link href='http://nbc.com/sluggo/' />
    <link rel='self' href='http://nbc.com/sluggo/index.atom' />
    <updated>2005-04-06T20:25:05-08:00</updated>
    <author><name>Mr. Bill</name></author>
    <entry>
      <title>A post about stuff</title>
      <link href='http://nbc.com/sluggo/20050420?id=321' />
      <id>http://nbc.com/sluggo/20050420?id=321</id>
      <updated>2005-04-06T20:25:05-08:00</updated>
      <content type='xhtml'>
         <!-- xhtml content -->
      </content>
    </entry>
</feed>
```

The Atom Publishing Protocol (APP)

"application-level protocol for publishing and editing Web resources using HTTP"

Also suitable for Wiki, CMS, search engines, etc.

- For example, Lucene-WS is based on APP

- Based on Atom Publishing Format
- Replaces old XML-RPC based blog APIs
  - Blogger/MetaWeblog API

The Old Blogger/MetaWeblog API

getUserBlogs	Get blogs as array of structures
deletePost	Delete blog post specified by id
newPost	Create new blog post by passing in structure*
editPost	Update existing blog post
getPost	Get blog post by id
getRecentPosts	Get most recent N blog posts
newMediaObject	Upload file to blog (e.g. picture of my cat)
getCategories	Get categories allowed in blog

* A hash-table where (most) keys correspond to RSS element names

What Does Atom Protocol Do?

- Atom WG charter says protocol must enable:
  - Creating, editing, and deleting feed entries
  - Multiple authors for a feed
  - Multiple subjects or categories in a feed
  - User authentication
  - Creating, getting and setting related resources* ~~such as comments, templates, etc.~~
  - ~~Adding, editing, and deleting users~~
  - ~~Setting and getting user preferences~~

*** Grey items won't be in first version of specification**

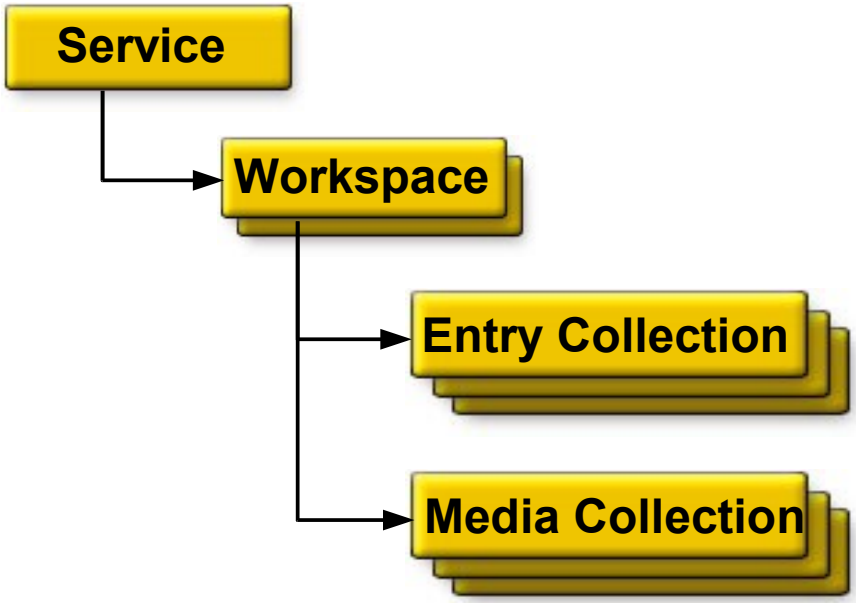
How Does it Do All That?

- The REST way
  - Everything's a resource, addressable by URI
  - HTTP verbs used for all operations

Atom Protocol: Everything's a Resource

- Entry point is an introspection document
- From there you find workspaces
- Workspace has collections
- Collections are resources and contain resources
  - Entry collections
  - Media collections
- Collections are represented as Atom feeds

Atom Protocol: Everything's a Resource



Blog account

Blogs

Collections of Blog entries

Collections of Uploaded files

An Atom Introspection Document

```
<?xml version="1.0" encoding='utf-8'?>
<service xmlns="http://purl.org/atom/app#">
    <workspace title="Main Site" >
        <collection title="My Blog Entries"
            href="http://example.org/reilly/main" >
            <member-type>entry</member-type>
        </collection>
        <collection title="Pictures"
            href="http://example.org/reilly/pic" >
            <member-type>media</member-type>
        </collection>
    </workspace>
</service>
```

Atom Protocol: HTTP Defines Operations

- POST to create Atom entries and media files
- GET to retrieve
  - Introspection document
  - Collections (represented as Atom feeds)
  - Individual resources (Atom entries or media files)
- PUT to update Atom entries and media files
- DELETE to delete an entry or a media file

Extending the Atom Protocol

- Servers can add XML elements via namespaces
- Clients that understand can benefit
- Clients that don't understand foreign markup
  - **SHOULD** preserve and return

Java Technology and REST: Agenda

Introduction to REST

The REST Versus WS-* Debate

Introduction to Atom Protocol

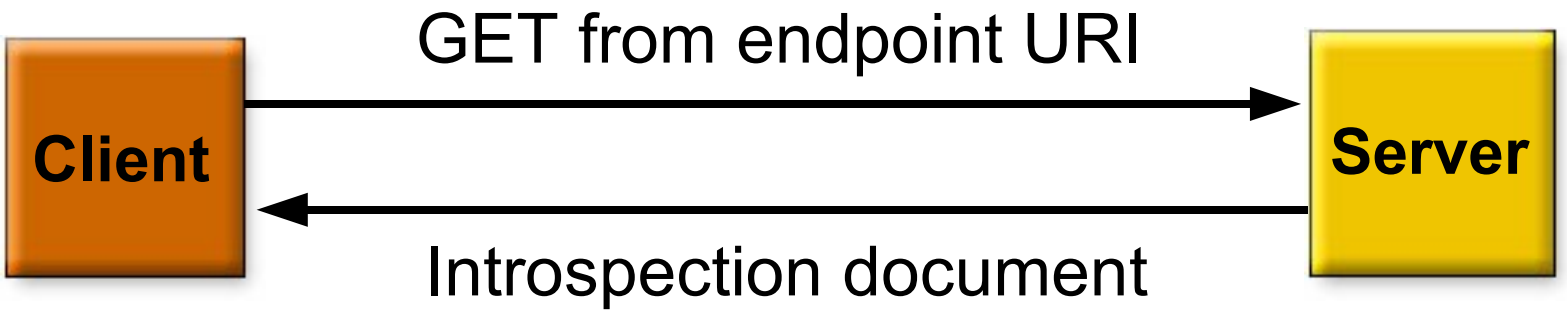
Exploring Atom Protocol

Implementing an Atom Protocol Server

Implementing an Atom Protocol Client

Summary

Introspection



HTTP GET w/authentication

```
// AuthGet.java
import java.io.InputStream;
import org.apache.commons.codec.binary.Base64;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.methods.GetMethod;
// ...define class and main() method...

String url = URL_TO_BE_CALLED;
String creds = USERNAME + ":" + PASSWORD;

HttpClient httpClient = new HttpClient();
GetMethod method = new GetMethod(url);
method.setRequestHeader("Authorization", "Basic "
    + new String(Base64.encodeBase64(creds.getBytes())));
httpClient.executeMethod(method);
InputStream is = method.getResponseBodyAsStream();
```

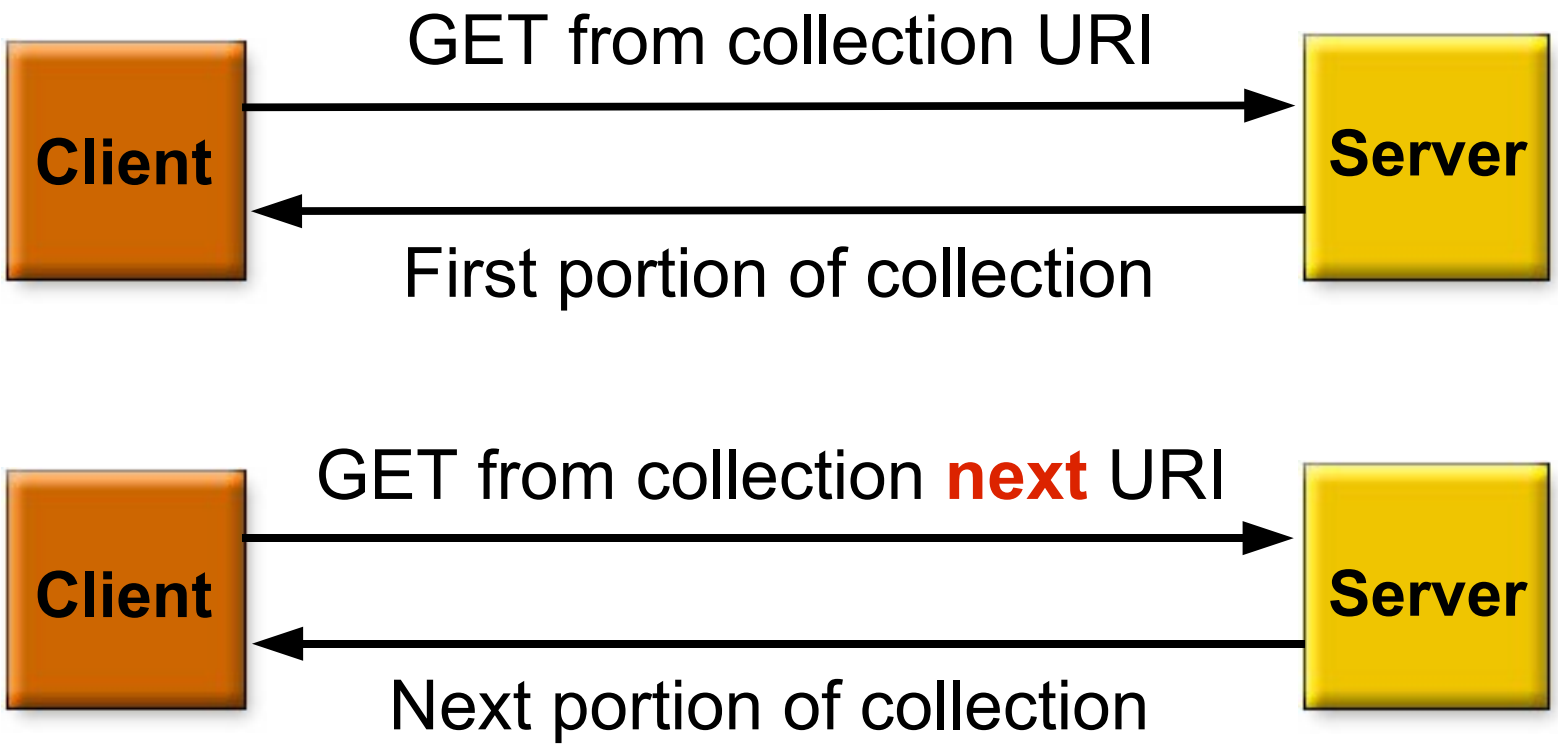
Atom Introspection Document (again)

```
<?xml version="1.0" encoding='utf-8'?>
<service xmlns="http://purl.org/atom/app#">
    <workspace title="Main Site" >
        <collection title="My Blog Entries"
            href="http://example.org/reilly/main" >
            <member-type>entry</member-type>
        </collection>
        <collection title="Pictures"
            href="http://example.org/reilly/pic" >
            <member-type>media</member-type>
        </collection>
    </workspace>
</service>
```

Primary **Entry** Collection URI

Primary **Media** Collection URI

Getting a Collection—With Paging



An Atom Collection <feed>

```
<feed xmlns="http://www.w3.org/2005/Atom">

  <link rel="next"
        href="http://example.org/entries/60" />
  <link rel="previous"
        href="http://example.org/entries/20" />
  ...
  <entry> ... </entry>
  <entry> ... </entry>
  <entry> ... </entry>
  <entry> ... </entry>
  ...
</feed>
```

URIs for next and previous portions of collection

<entry> In a Collection

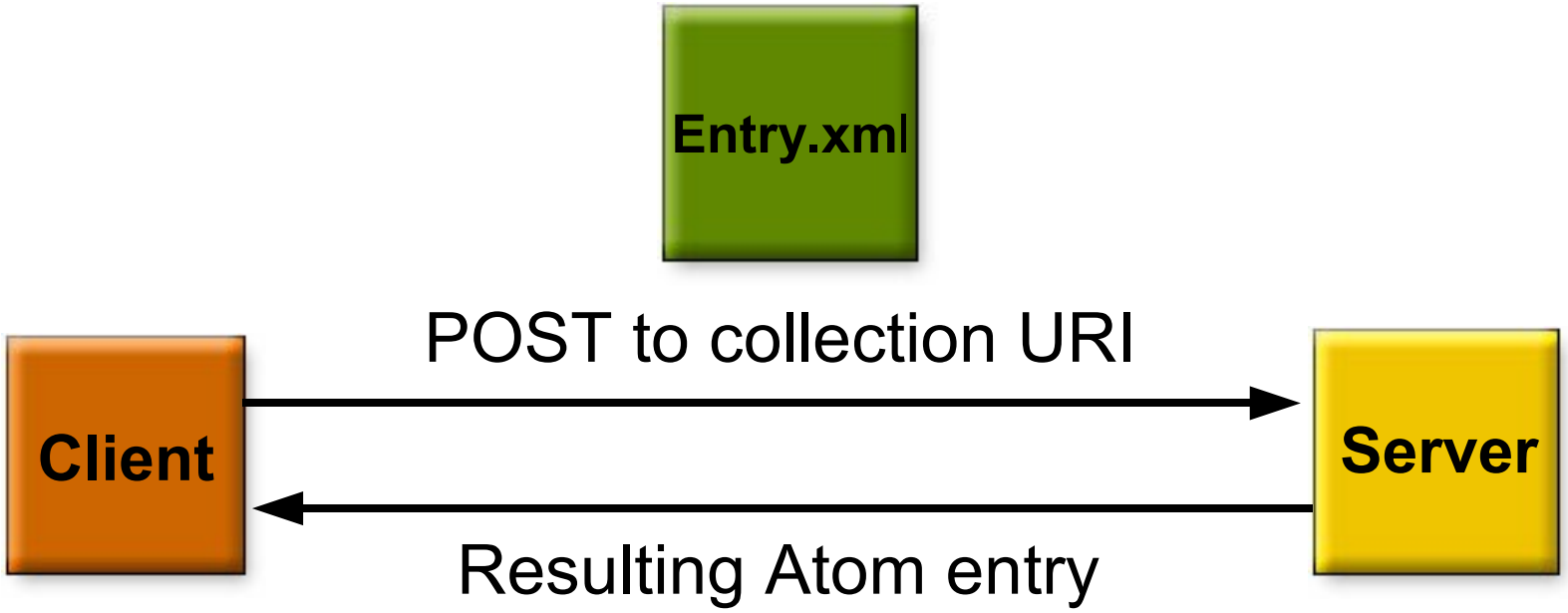
```
<entry>
  <title>First post!</title>
  <link rel="alternate"
     href="http://localhost/roller/page/bill?entry=post1 />
  <link rel="edit"
     href="http://localhost/roller/app/bill/entry/757" />
  <category term="/Sun" />

 <id>http://localhost:8080/roller/page/bill?entry=post1</id>
  <updated>2005-12-27T22:08:03Z</updated>
  <published>2004-10-13T01:07:59Z</published>
  <content type="html">I'm so blogging this</content>
  <app:control>
    <app:draft>no</app:draft>
  </app:control>
 </entry>
</feed>
```

Entry's permalink

Edit URI for entry

APP namespace
to indicate draft status

Creating an Entry



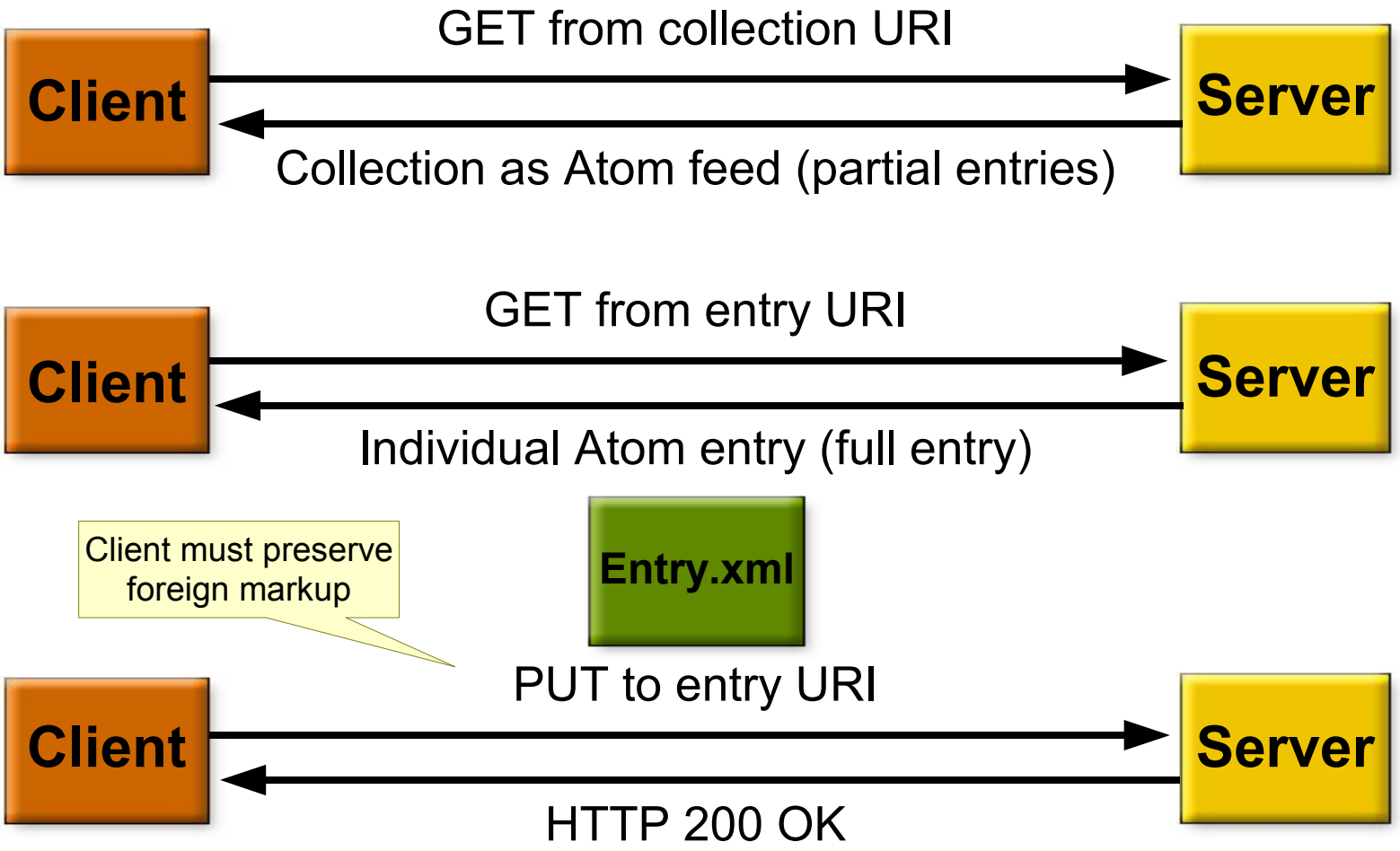
HTTP POST w/authentication

```
// AuthPost.java
import java.io.*;
import org.apache.commons.codec.binary.Base64;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.methods.*;

// ...define class and main() method...

String url = URL_TO_BE_CALLED;
String creds = USERNAME + ":" + PASSWORD;

HttpClient httpClient = new HttpClient();
EntityEnclosingMethod method = new PostMethod(url);
method.setRequestHeader("Authorization", "Basic "
    + new String(Base64.encodeBase64(creds.getBytes())));
```

HTTP POST w/authentication (Cont.)

```
String filePath = FILE_PATH;
String contentType = CONTENT_TYPE;

File upload = new File(filePath);
method.setRequestHeader("name", upload.getName());
method.setRequestHeader("Content-type", contentType);


method.setRequestBody(new FileInputStream(upload));
httpClient.executeMethod(method);

System.out.println(method.getResponseBodyAsString());
```

Updating an Entry



DEMO

Atom protocol in action
With AuthGet.java and AuthPost.java

Java Technology and REST: Agenda

Introduction to REST
The REST Versus WS-* Debate
Introduction to Atom Protocol
Exploring Atom Protocol
Implementing an Atom Protocol Server
Implementing an Atom Protocol Client
Summary

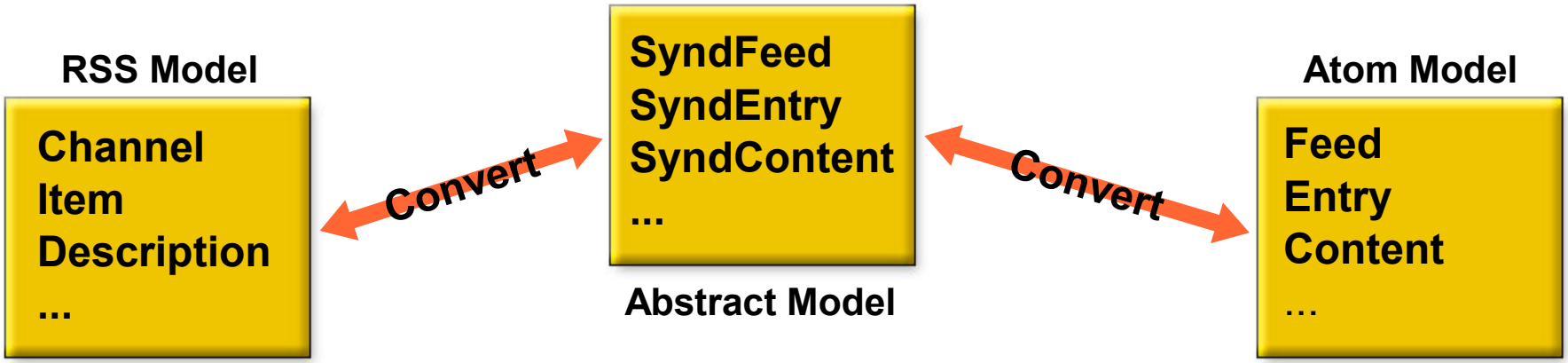
Roller's APP Implementation

- Created for Roller blog server
  - But does not depend on Roller
  - Depends only on Servlet API and ROME
  - Could be used in other servers, perhaps your own

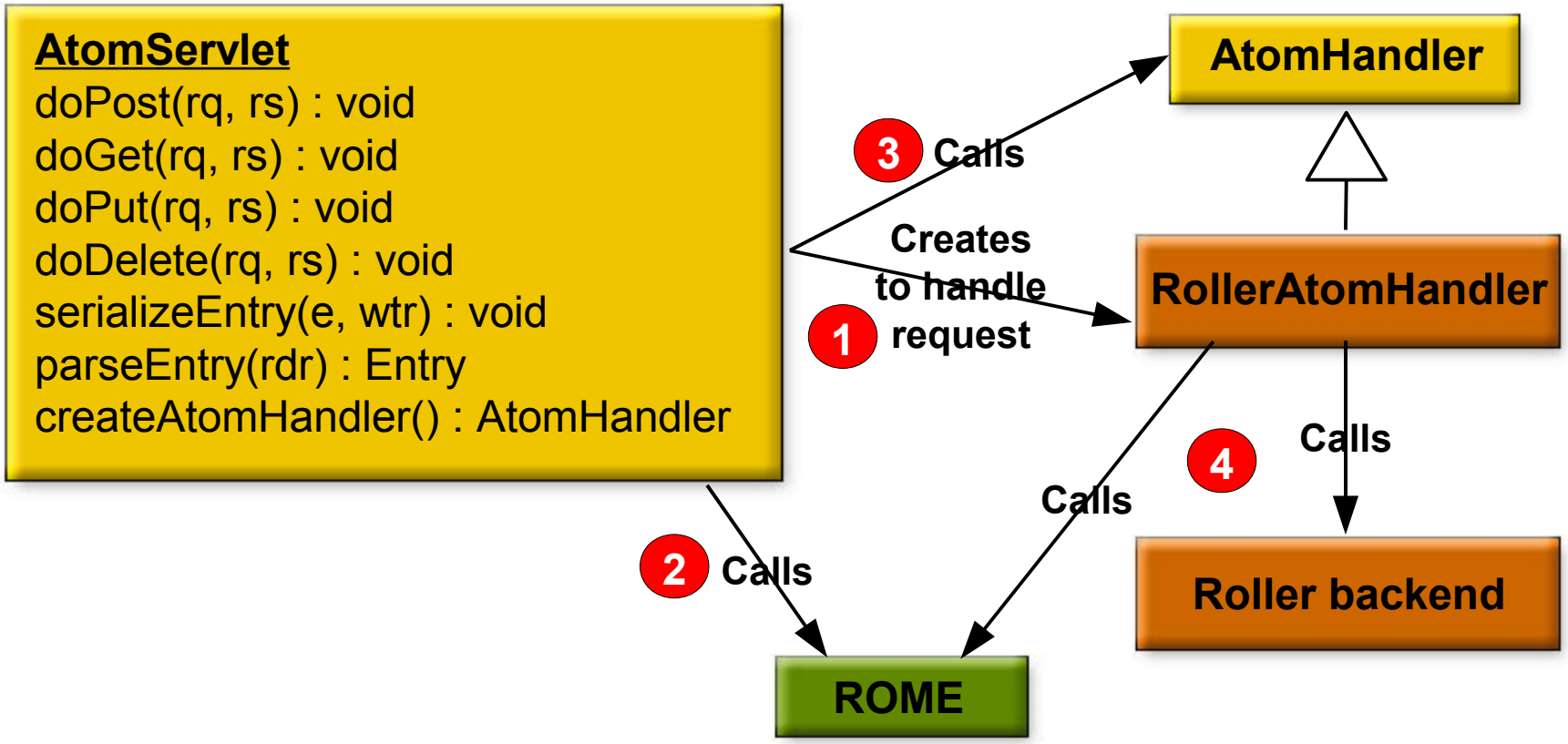
- URI structure
  - `http://localhost:8080/roller/app`
  - `http://localhost:8080/roller/app/<blogname>/entries`
  - `http://localhost:8080/roller/app/<blogname>/entry/<id>`

What Is ROME?

- RSS and Atom format feed parser and generator
- Minimal dependencies (JDOM only)
- Object models for RSS, Atom. And **converters**
- Abstract `SyndFeed` model



Roller's APP Implementation



AtomHandler.java

```
package org.roller.presentation.atomapi;
import java.io.InputStream;
import java.util.Date;
import com.sun.syndication.feed.atom.Entry;
import com.sun.syndication.feed.atom.Feed;

public interface AtomHandler {
    public String getAuthenticatedUsername();

    public AtomService getIntrospection(String[] pathInfo)
        throws Exception;
    public Feed getCollection(String[] pathInfo)
        throws Exception;

    public boolean isIntrospectionURI(String [] pathInfo);
    public boolean isCollectionURI(String [] pathInfo);
    public boolean isEntryCollectionURI(String [] pathInfo);
    public boolean isMediaCollectionURI(String [] pathInfo);
    public boolean isEntryURI(String[] pathInfo);
    public boolean isMediaURI(String[] pathInfo);
```

You use pathInfo to decide which collection to return

You decide how URIs are mapped

AtomHander.java (Cont.)

You use pathInfo to decide which Entry collection

```
    public Entry postEntry(String[] pathInfo, Entry entry)
        throws Exception;
    public Entry getEntry(String[] pathInfo)
        throws Exception;
    public Entry putEntry(String[] pathInfo, Entry entry)
        throws Exception;
    public void deleteEntry(String[] pathInfo)
        throws Exception;

    public Entry postMedia(String[] pathInfo,
        String name, String contentType,
        InputStream is) throws Exception;
    public Entry putMedia(String[] pathInfo, String contentType,
        InputStream is) throws Exception;
    public void deleteMedia(String[] pathInfo)
        throws Exception;
    public Entry getMedia(String[] pathInfo)
        throws Exception;
}
```

You use pathInfo to decide which Media collection

Example Method: `getEntry()`

```
public Entry getEntry(String[] pathInfo) throws Exception {

    if (pathInfo.length == 3) // URI is /blogname/entries/entryid
    {

        WeblogEntryData entry = mRoller.getWeblogManager()
            .retrieveWeblogEntry(pathInfo[2]);

        if (entry != null && !canView(entry)) {
            throw new Exception(
                "ERROR not authorized to view entry");
        } else if (entry != null) {
            return createAtomEntry(entry);
        }
        return null;
    }
    throw new Exception("ERROR: bad URI");
}
```

Example Method: `postEntry()`

```
public Entry postEntry(String[] pathInfo, Entry entry)
    throws Exception {

    // URI is /blogname/entries
    String handle = pathInfo[0];
    WebsiteData website =
       mRoller.getUserManager().getWebsiteByHandle(handle);
    UserData creator =
        mRoller.getUserManager().getUser(mUsername);

    if (canEdit(website)) {
        WeblogEntryData rollerEntry =
            createRollerEntry(website, entry);
        rollerEntry.setCreator(creator);
        rollerEntry.save();
        mRoller.commit();
```

(Continued...)

Example Method: `postEntry()`

(Continued...)

```
            CacheManager.invalidate(website);

            if (rollerEntry.isPublished()) {
                mRoller.getIndexManager()
                    .addEntryReIndexOperation(rollerEntry);
            }
            return createAtomEntry(rollerEntry);
        }
        throw new Exception(
            "ERROR not authorized to edit website");
    }
```

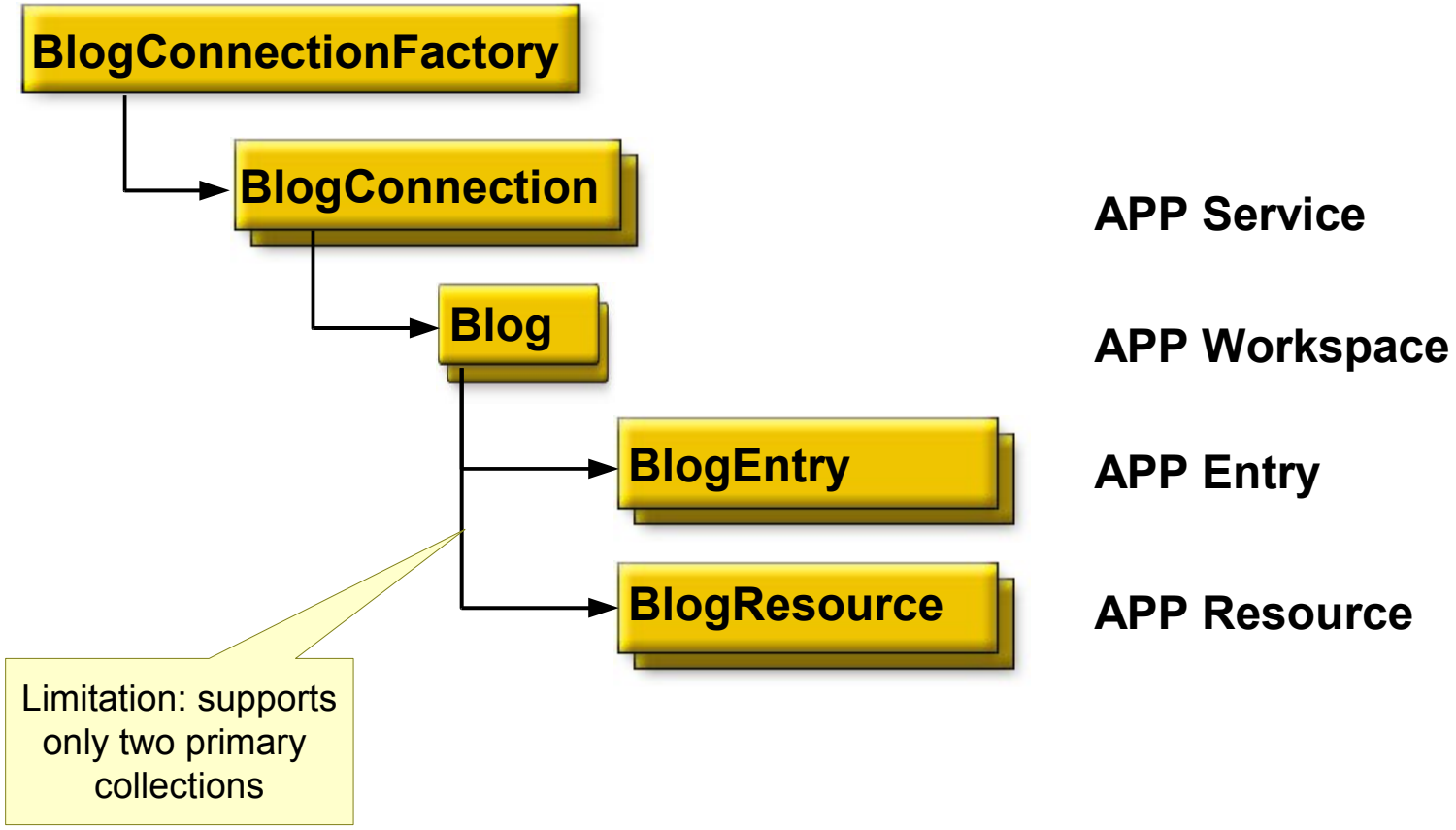
Java Technology and REST: Agenda

Introduction to REST
The REST Versus WS-* Debate
Introduction to Atom Protocol
Exploring Atom Protocol
Implementing an Atom Protocol Server
Implementing an Atom Protocol Client
Summary

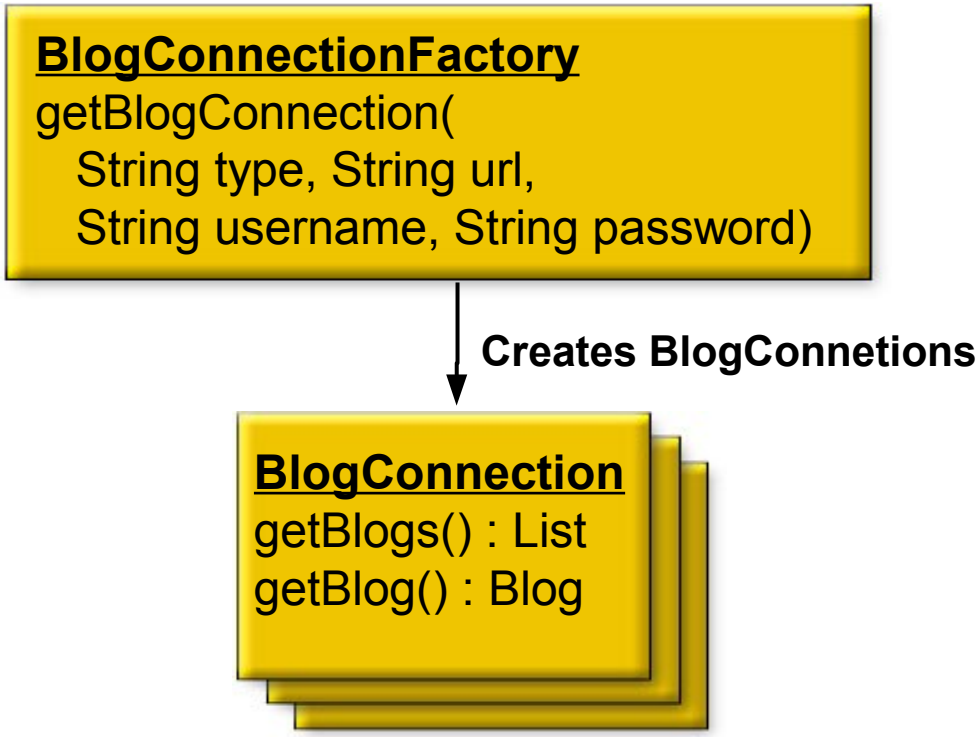
Implementing an Atom Protocol Client

- Approach: **Blog Client Library** abstraction
- Support both Atom protocol and MetaWeblog API
- Not necessarily the only or the best approach
  - Having to support MetaWeblog API holds us back
  - In the long run, pure Atom might be a better approach
  - **APP is generic; it's not just for blogs**

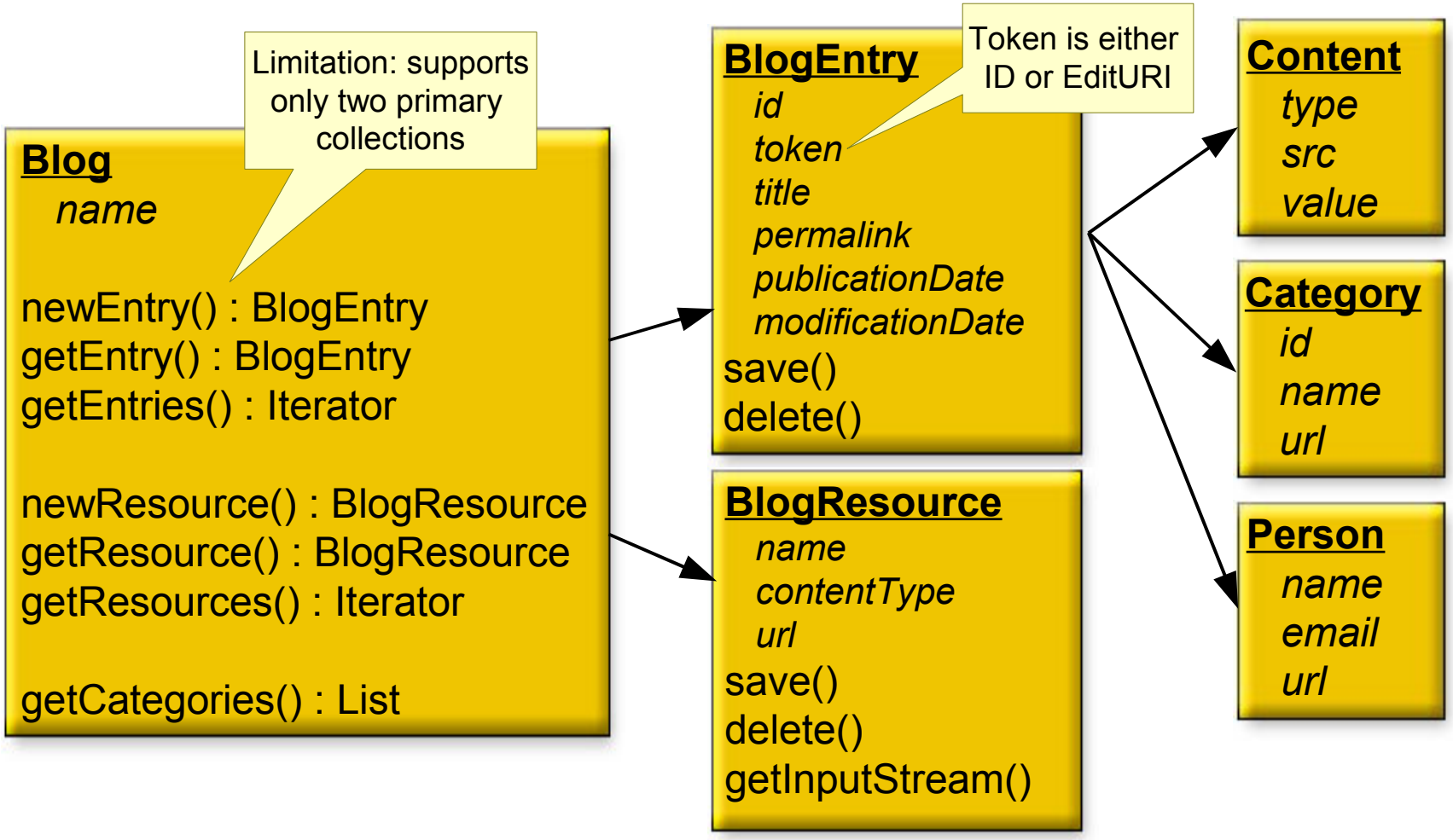
Blog Client Library Interfaces



Blog Client Library: Factory



Blog Client Library: Blog Interfaces



Implementing the Blog Client Library

- Interfaces depend only on core Java libraries

- Implementation will depend on:
  - **ROME** To parse Atom format
  - **JDOM** To extend ROME
  - **Apache HTTPClient** For HTTP operations

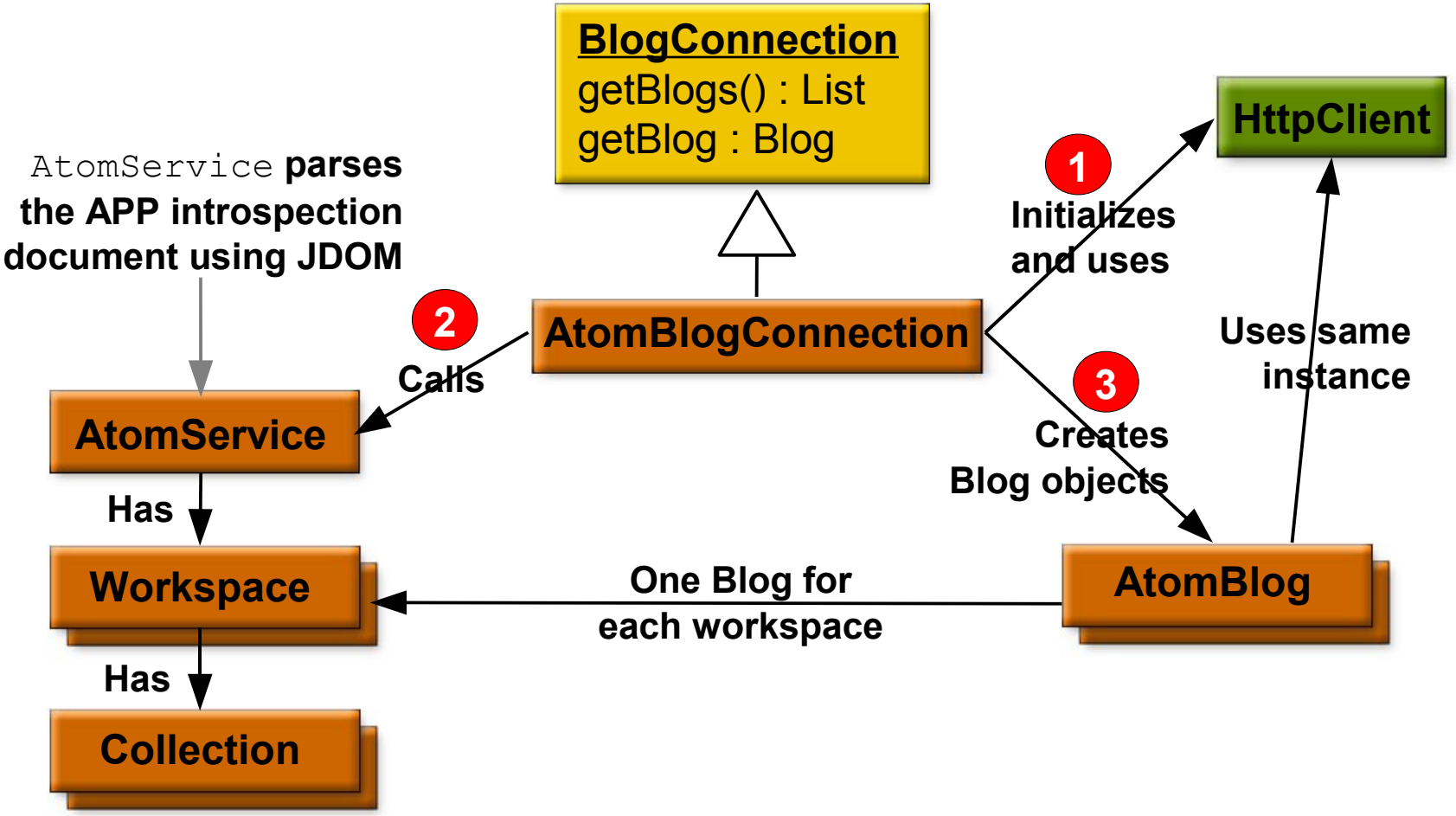
What Is HTTPClient?

- HTTP client library for Java technology
- Replaces `java.net.URLConnection`
  - And is superior in some ways
- Not required for Atom protocol support
  - Everything in this presentation is possible with the old reliable `java.net.URLConnection`
  - (but I prefer HTTPClient)

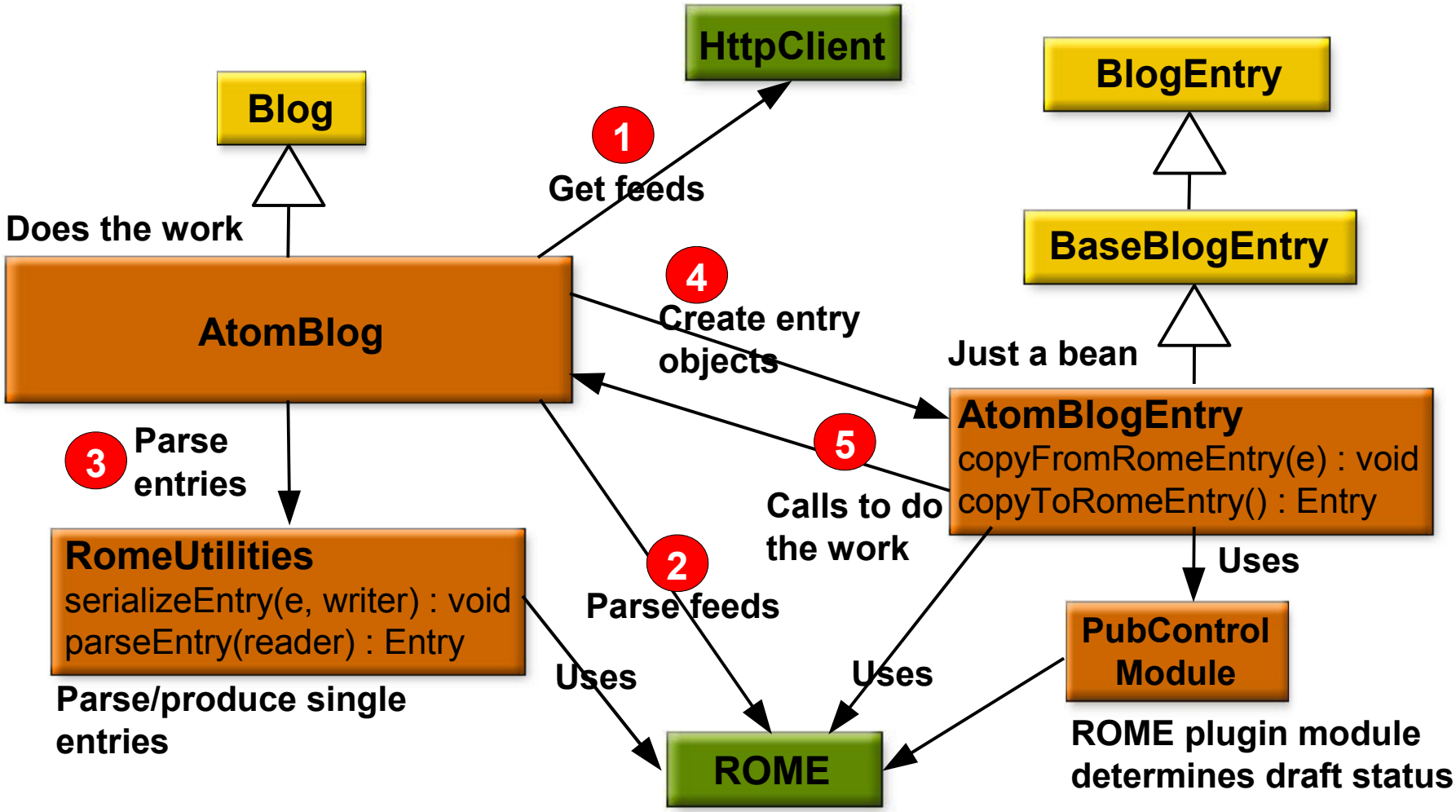
APP Client Implementation

- AtomBlog.java
- AtomBlogConnection.java
- AtomEntry.java
- AtomResource.java
- AtomService.java
- PubControlModule.java
- PubControlModuleGenerator.java
- PubControlModuleImpl.java
- PubControlModuleParser.java
- RomeUtilities.java

AtomBlogConnection.java



AtomBlog.java



Example: AtomBlog.getEntry()

```
public BlogEntry getEntry(String token) throws Exception {

    GetMethod method = new GetMethod(editURI);
    AtomBlogConnection.addAuthentication(
        method, username, password);
    httpClient.executeMethod(method);
    if (method.getStatusCode() != 200) {
        throw new Exception("ERROR HTTP status code="
            + method.getStatusCode());
    }
    String body = method.getResponseBodyAsString();

    Entry romeEntry = RomeUtilities.parseEntry(
        new StringReader(body));

    return new AtomEntry(this, romeEntry, false);
}
```

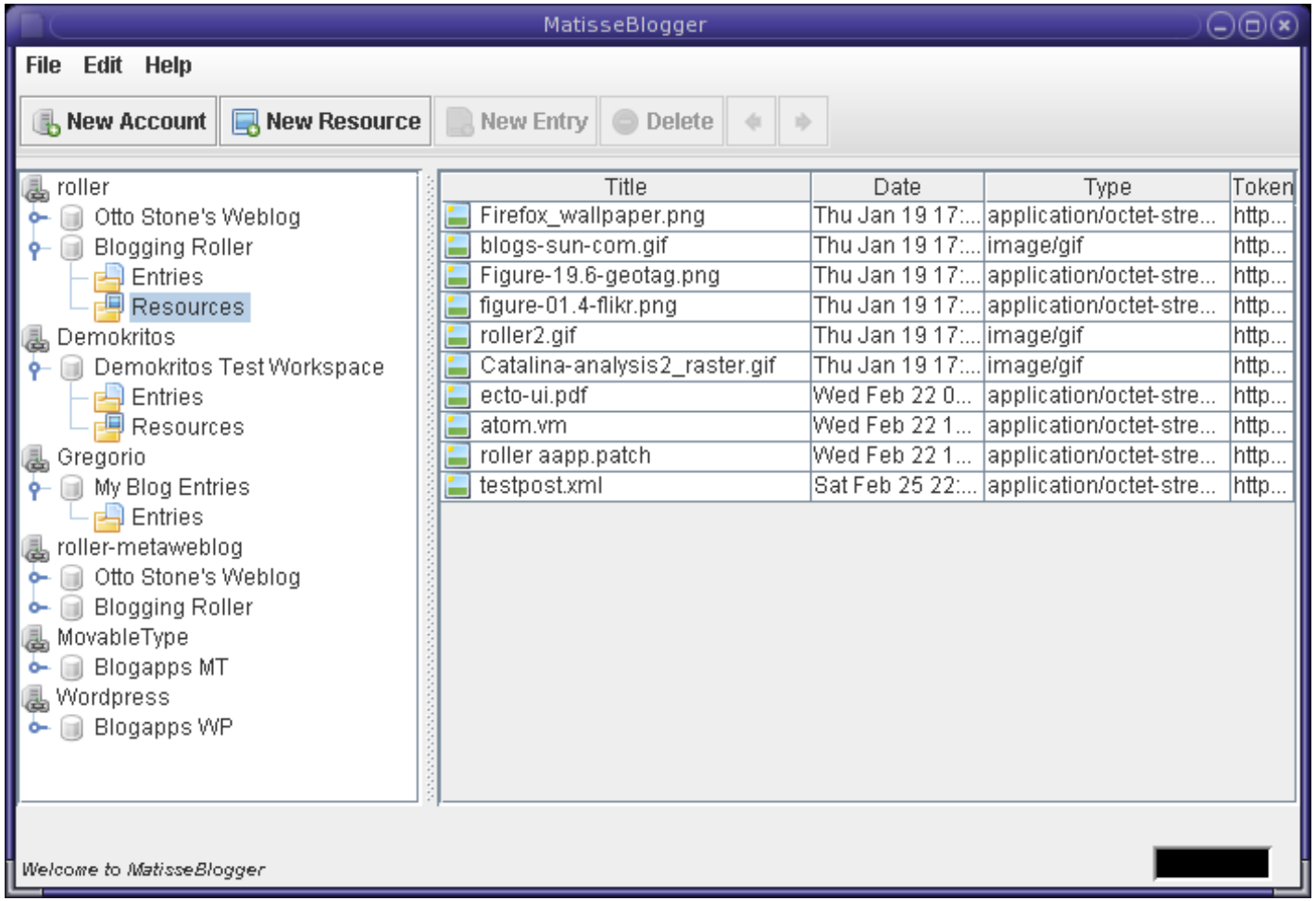
Example: AtomBlog.saveEntry()

```
String saveEntry(BlogEntry entry) throws Exception {

    EntityEnclosingMethod method = null;
    boolean create = (entry.getToken() == null);
    if (create) method = new PostMethod(entriesURI);
    else        method = new PutMethod(entry.getToken());

    StringWriter sw = new StringWriter();
    RomeUtilities.serializeEntry(
        ((AtomEntry)entry).copyToRomeEntry(), sw);
    method.setRequestBody(sw.toString());

    AtomBlogConnection.addAuthentication(
        method, username, password);
    method.setRequestHeader("Content-type",
        "application/atom+xml; charset=utf8");
    httpClient.executeMethod(method);
    ...
```

Blog Client Usage: Simple Example

```
import com.manning.blogapps.chapter08.blogclient.*;
import java.util.*;
import java.io.*;

// get these values from somewhere...
String endpoint, username, password, title, content;

BlogConnection con =
    BlogConnectionFactory.getBlogConnection(
        "atom", endpoint, username, password);
Blog blog = (Blog)con.getBlogs().get(0);

BlogEntry entry = blog.newEntry();
entry.setTitle(title);
entry.setContent(new BlogEntry.Content(content));
entry.save();
```

Blog Client Usage: Swing Example



DEMO

Swing based Atom protocol client

Java Technology and REST: Agenda

Introduction to REST
The REST Versus WS-* Debate
Introduction to Atom Protocol
Exploring Atom Protocol
Implementing an Atom Protocol Server
Implementing an Atom Protocol Client
Summary

Summary

- Those REST folks have a point
  - REST **is** easy
- Don't use WS-* unless you really need it
  - i.e. You need to tie into some existing WS-* service
- Atom is not just for blogs
  - It might be all you need for your REST based WS
  - The format can **carry** almost anything
  - Collections + CRUD can **do** almost anything

For More Information

- IETF Specifications (http://ietf.org)
  - Atom Publishing Format (RFC-4287)
  - Atom Publishing Protocol
- Other resources
  - http://AtomEnabled.org
  - http://FeedValidator.org
  - http://rome.dev.java.net
  - http://jakarta.apache.org/commons/httpclient/
  - http://blogapps.dev.java.net
  - http://rollerweblogger.org

Q&A

Dave Johnson

Java™ Technology and REST: Implementing the Atom Publishing Protocol

Dave Johnson

Staff Engineer
Sun Microsystems
http://rollerweblogger.org/page/roller

TS-1756