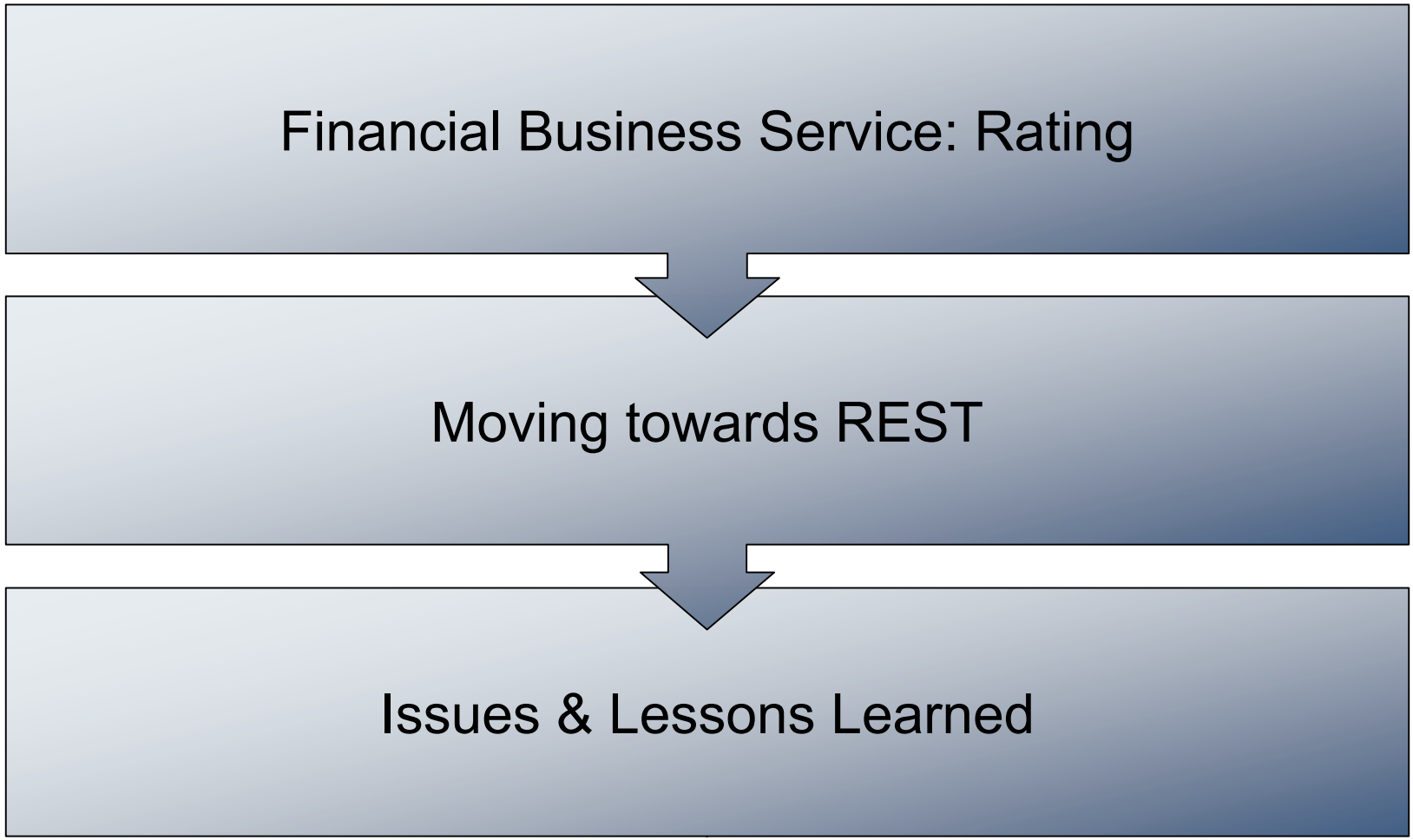
Phillip Ghadir, innoQ

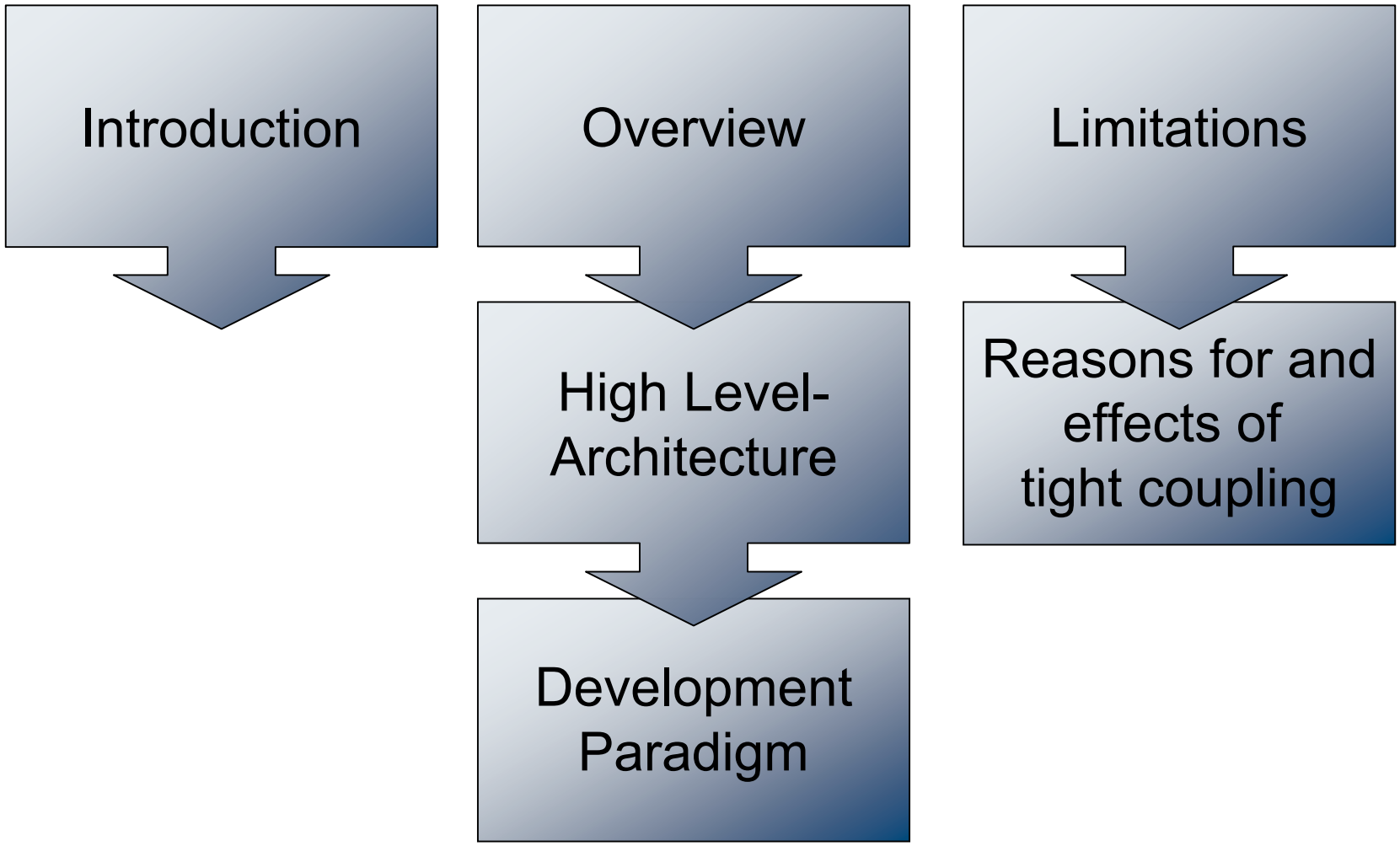
REST-based Integration Architecture for a Financial Business Service

When we started out building a large-scale financial application, we followed all the then-current buzzwords: **SOAP, WSDL, WS-***. Many of the benefits we expected failed to arrive. We finally ended up developing new integration scenarios using an approach based on **REST**, **Atom**, and **AtomPub**, and have since seen a significant improvement in re-use and modularity. I will present the evolution of our customer's rating service, initially available via web services up to the soft migration towards a **REST-based approach**. See which design decisions were made and how they turned out since its release in early 2007. This presentation will end with an explanation of pitfalls and shortcomings and the workarounds we chose.

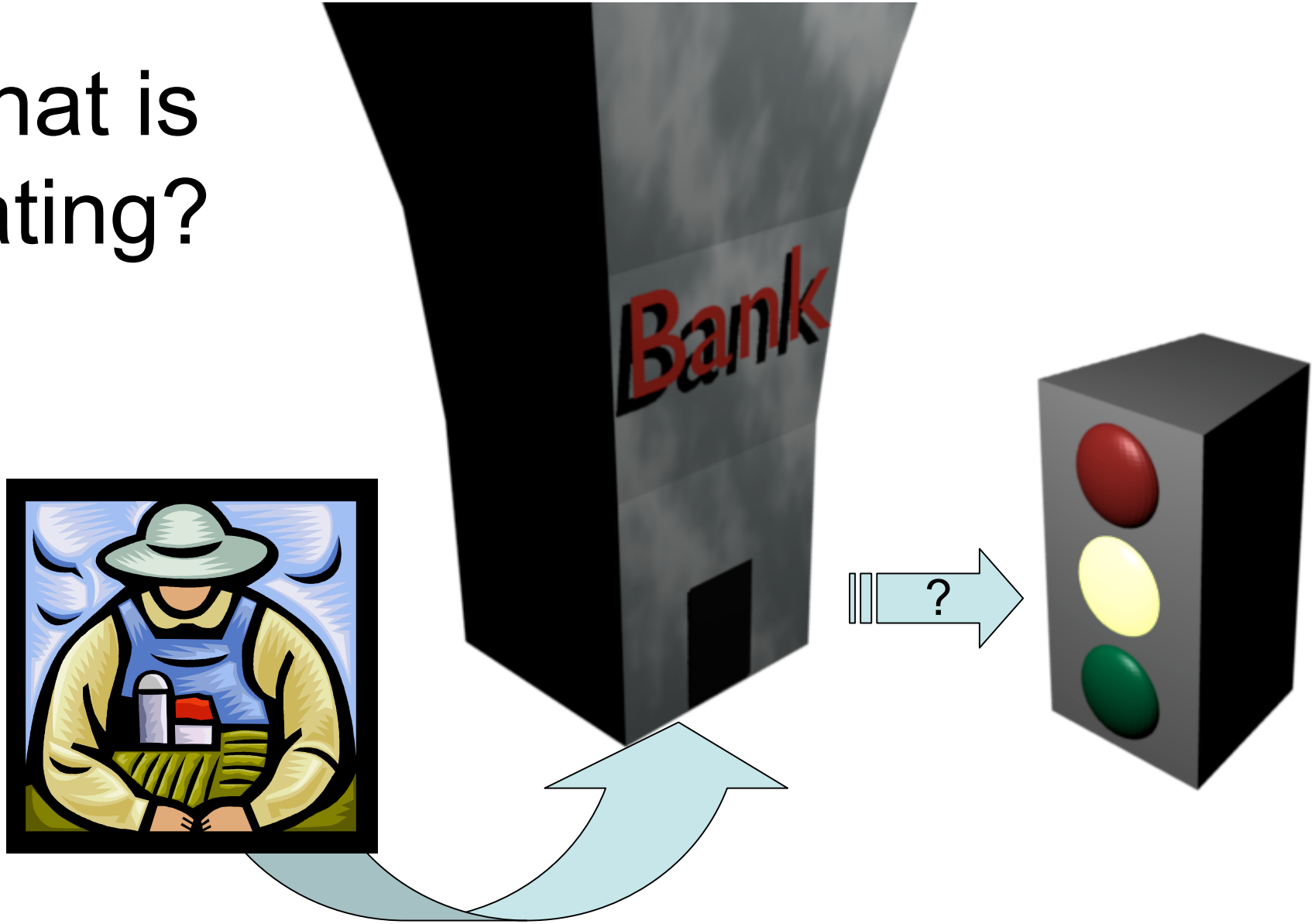
Agenda


Financial Business Service: Rating
Moving towards REST
Issues & Lessons Learned

Rating Business Service



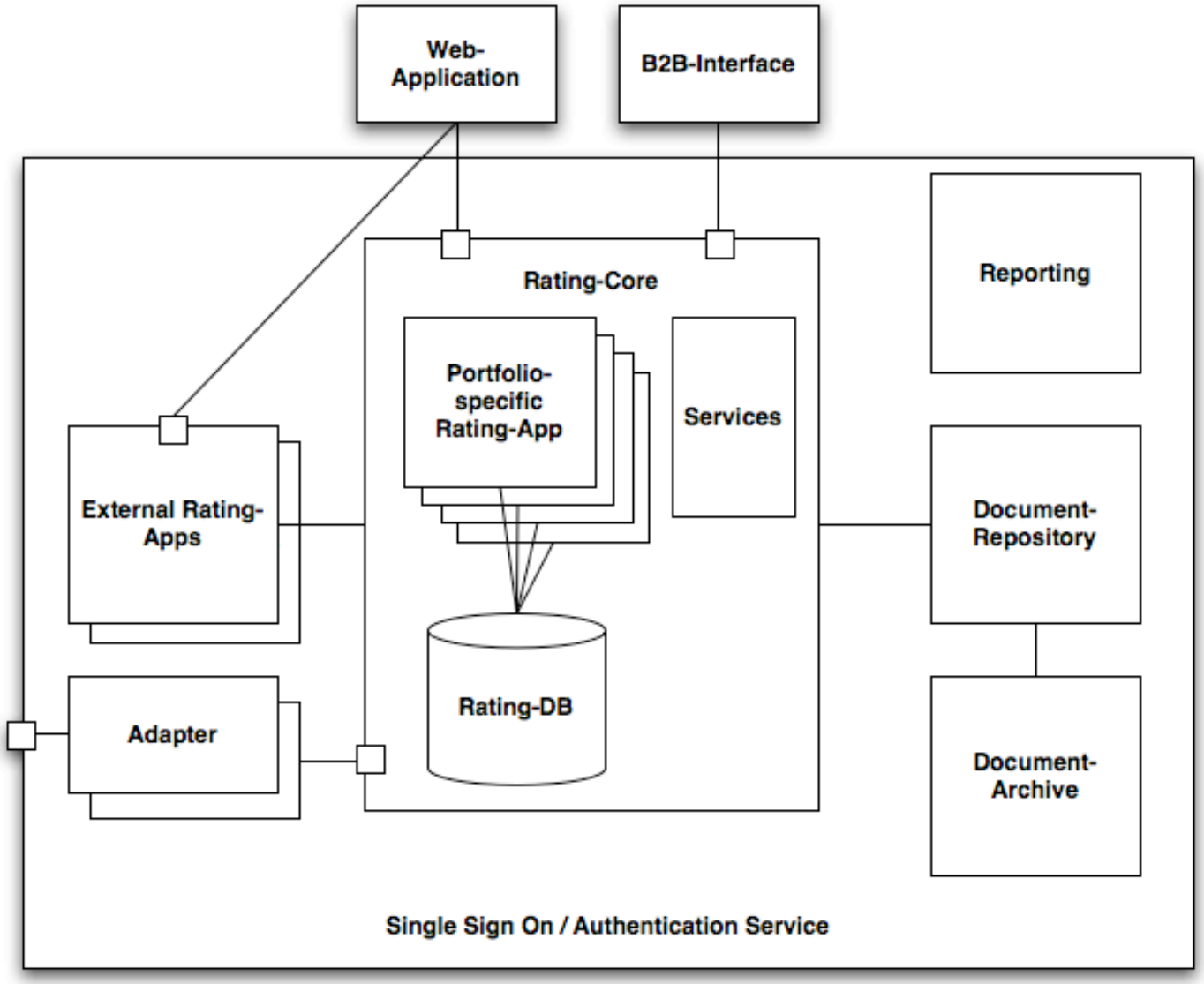
What is Rating?



Rating as a Business Service

- web-application hosted for several banks
- Provided with surrounding business processes (like data backup, validation, auditing, and callibration)
- Integrated into banks‘ own workflows and processes
- Integration into banks‘ applications via web services-based B2B-interface

Architectural Overview



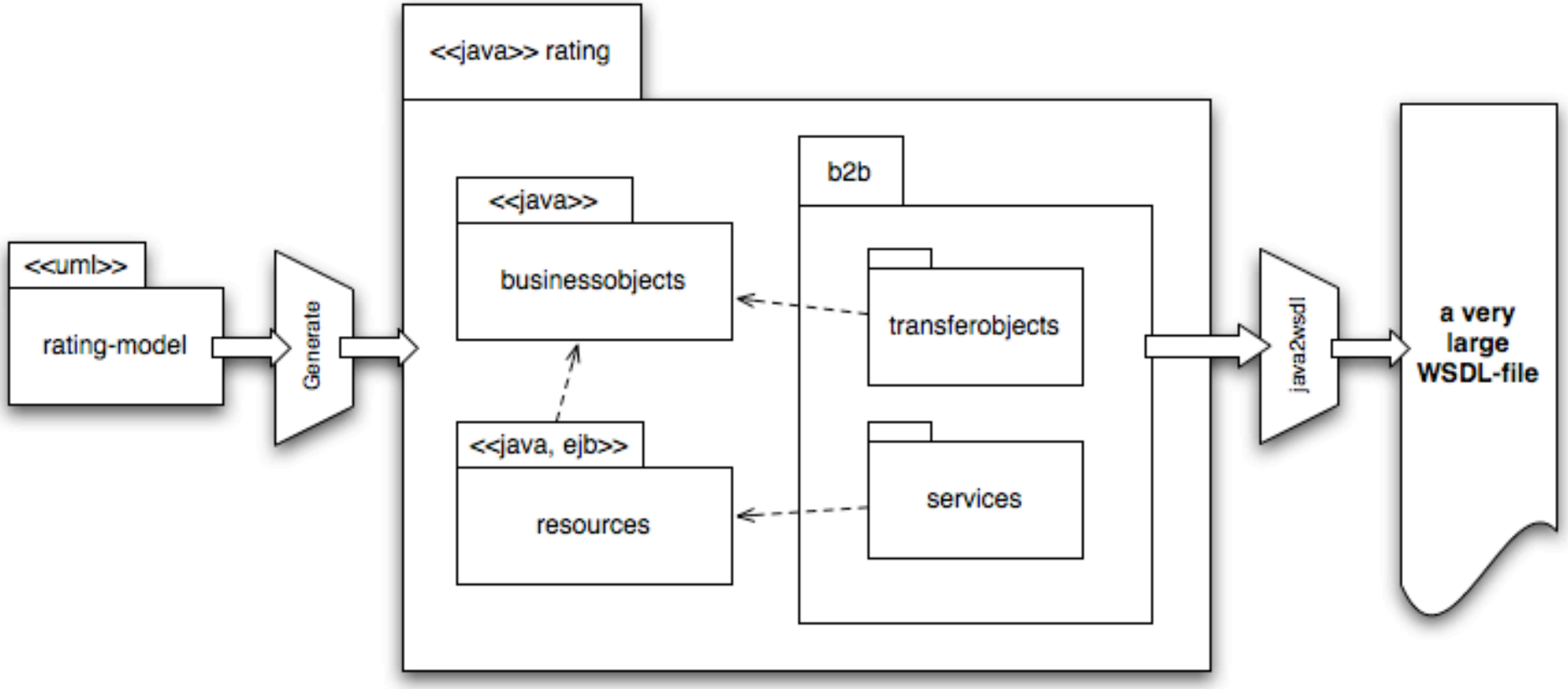
In the beginning ...

- WS-I basic profile wasn‘t available
- Interoperability with Microsoft, Perl, and several Java web service stacks was mandatory
- Systinet Server for Java had the best interoperability

... but

- Use of collections wasn‘t possible
- Had to
  - use arrays
  - integrate schema information into WSDL
  - Have one large WSDL
  - Provide a B2B abstraction layer especially for circumventing the WSDL-constraints

Model driven generative / Code first approach

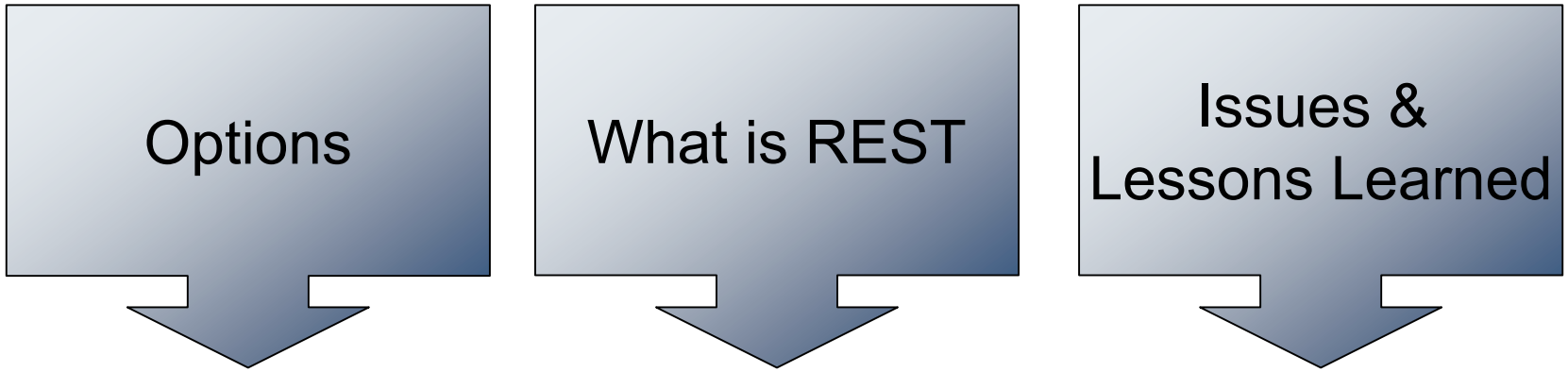


Reasons for and Effects of Tight Coupling

- „One source for all“ approach
- Use of binding generators
- Lack of attention (read: time & budget)
- Established organizational activities
- Fixed time slots for changes

Due to the fixed time slots we were forced to evaluate other options.

Moving towards REST



Our Options

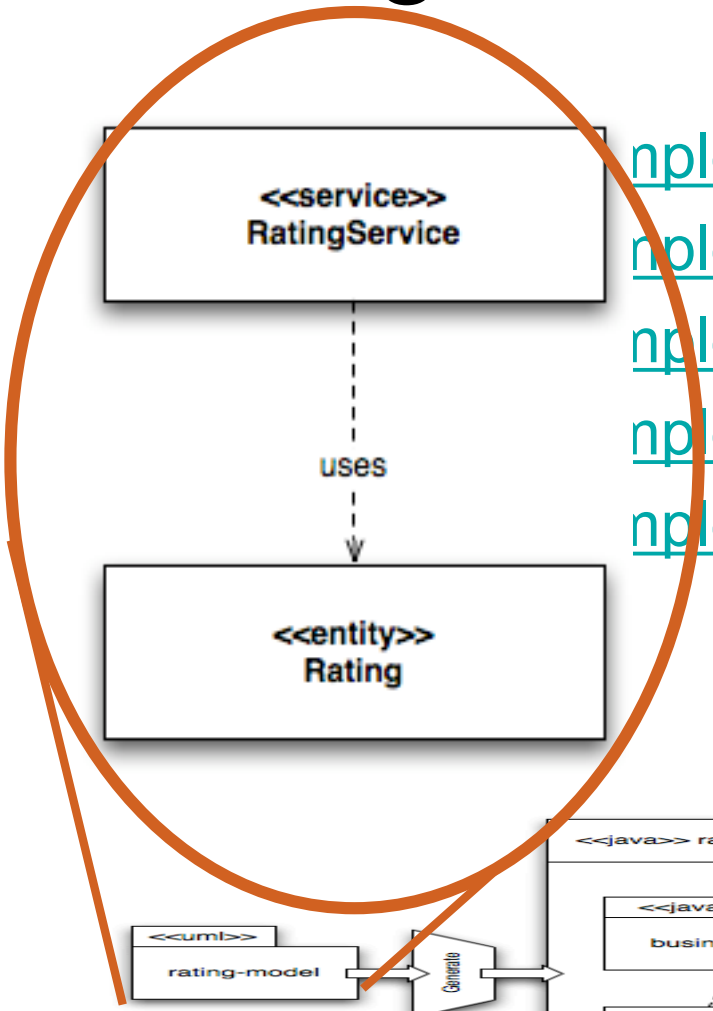
- Making web services „right“
  - Creating an own source for web services definition
  - Building a mediation layer
- Building new application as internal components of the existing system
- Adding new rating applications more loosely coupled

REST foundations

Every resource gets its own ID (=URI)

- http://rating.example.com/
- http://rating.example.com/01/hardfacts/2008/
- http://rating.example.com/01/ratings/2008
- http://rating.example.com/02/customer/4711
- http://rating.example.com/03/customer/0815/ratings

Finding the resources was easy



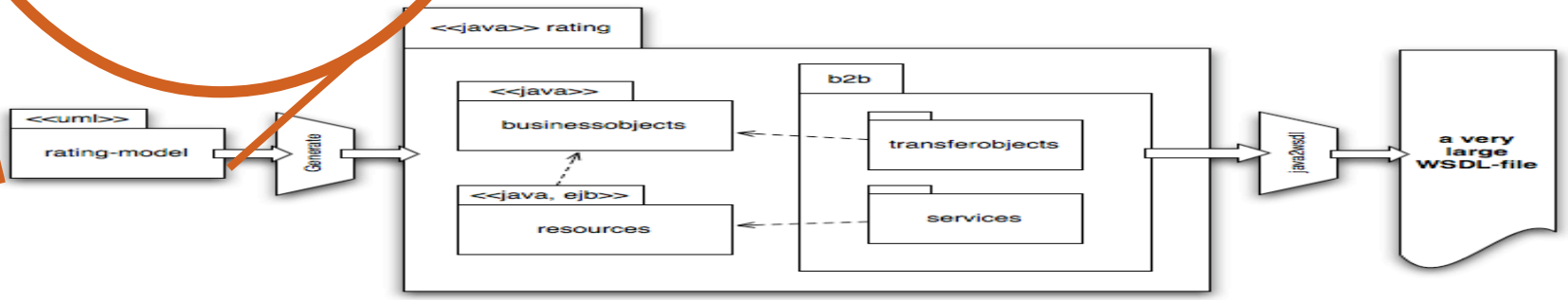
nple.com/

nple.com/01/hardfacts/2008/

nple.com/01/ratings/2008

nple.com/02/customer/4711

nple.com/03/customer/0815/ratings



Hypermedia - Link Everything Together

```
<file>
    <customer id="http://demo/k/4711">
        <lastname>Potter</lastname>
        <firstname>Harry</firstname>
        <customer-no>HP-4711</customer-no>
    </customer>
    <rating id="http://demo/ratings/5"/>
    <rating id="http://demo/ratings/3"/>
    <hardfacts id="http://demo/hf/1"/>
</kundenakte>
```

Adding some URIs was easy ... But ...

```
<file>
    <customer id="http://demo/k/4711">
        <lastname>Potter</lastname>
        <firstname>Harry</firstname>
        <customer-no>HP-4711</customer-no>
    </customer>
    <rating id="http://demo/ratings/5"/>
    <rating id="http://demo/ratings/3"/>
    <hardfacts id="http://demo/hf/1"/>
</kundenakte>
```

Access everything via standard operations

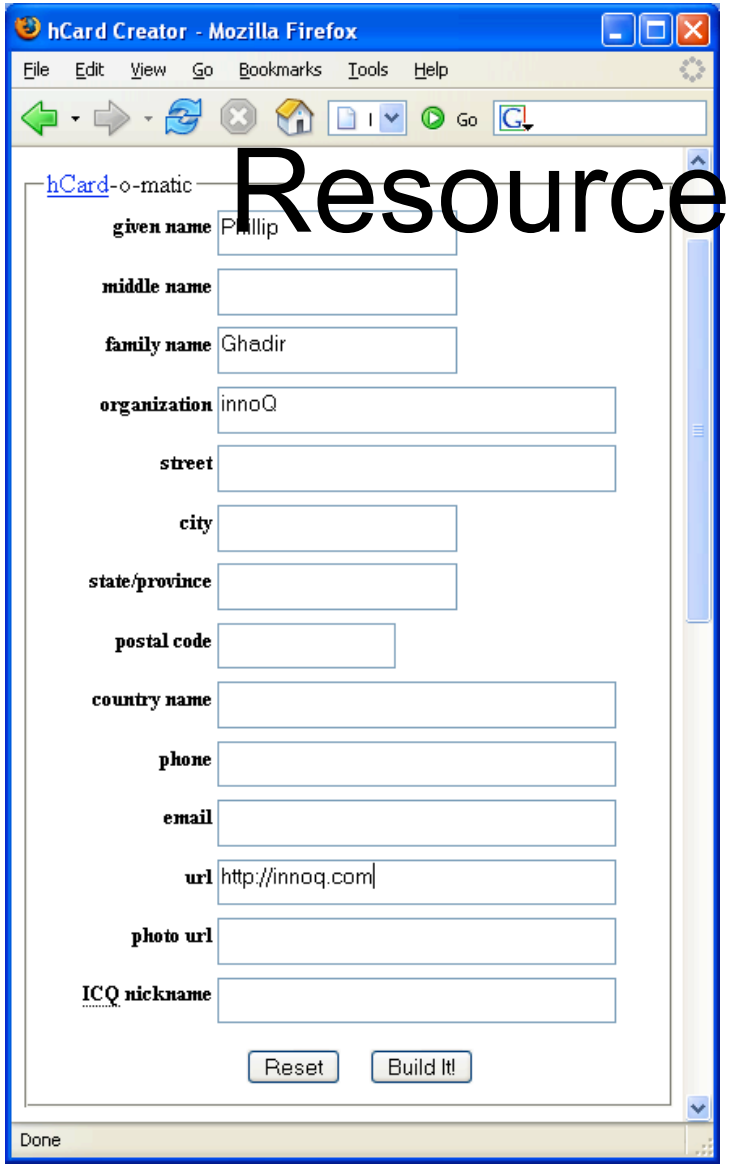
- DELETE

- GET 
- POST []

- PUT

Resource Representations

http://innoq.com/resources/employees/pg



As XHTML form

```
BEGIN:VCARD
VERSION:3.0
N:Ghadir; Phillip
FN:Phillip Ghadir
URL:http://innoq.com/
ORG:innoQ
END:VCARD
```

As vcard

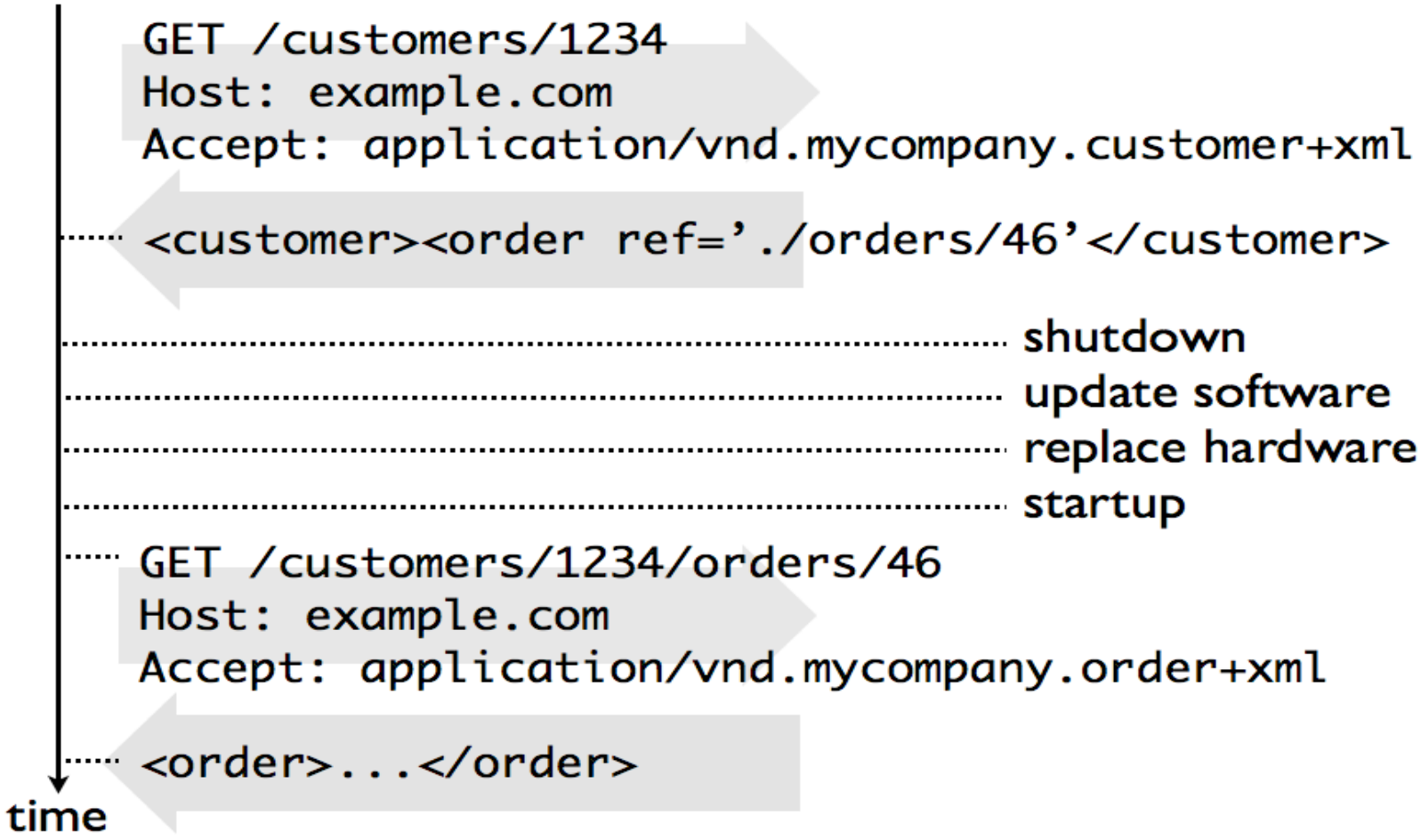
```
<div class="vcard">
   <a class="url fn"
      href="http://innoq.com/">Phillip Ghadir</a>
   <div class="org">innoQ</div>
</div>
```

As hcard

Most interesting representations

- Something we could easily process
  - Manually
  - Automatically
- Lists require other representations as entities
  - Forms (for queries)
  - XHTML (<ul> ...)
  - Atom
  - CSV
- Meta-Information provided via AtomPub

Stateless Communication



Design Guidelines

1. URIs contain no business data!
2. Exception tenant-ID (http://rating.example.com/01/ratings/2008)
3. Server builds URIs - not the client!
4. Collections are provided as Atom feeds
5. Discovery is done via AtomPub service documents
6. Links are selected by technical labels
7. Business data is content of an Atom feed entry
8. Metadata of the content is contained in the Atom entry
9. Requirements for http headers are forbidden

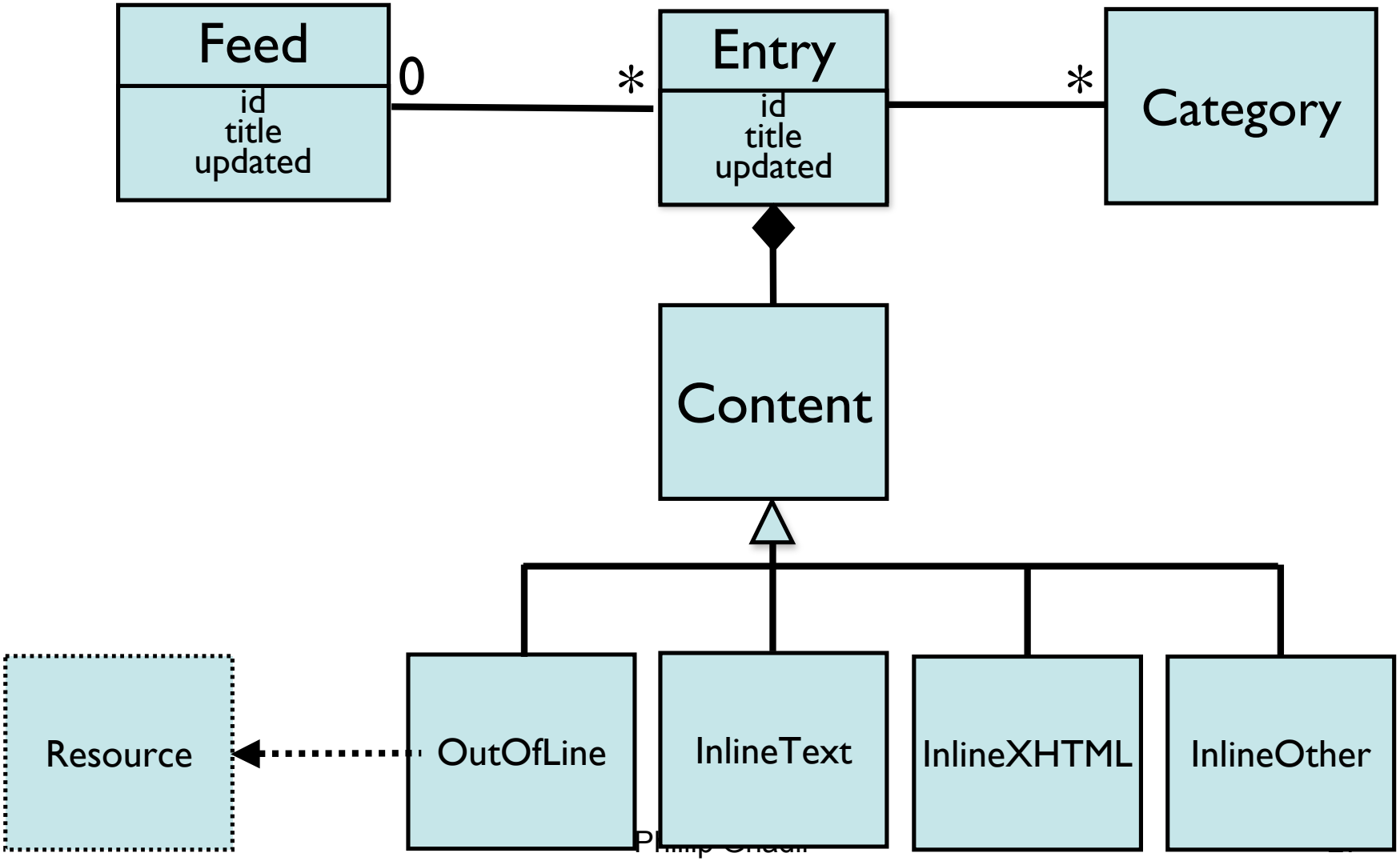
Short introduction
on
Atom & AtomPub

Atom Syndication Format

Standardized in: RFC 4287
MIME Type: application/atom+xml
Namespace: http://www.w3.org/2005/Atom

RSS Done Right

Atom Model


Feed
id
title
updated
0
*
Entry
id
title
updated
*
Category
Content
Resource
OutOfLine
InlineText
InlineXHTML
InlineOther

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Example Feed</title>
  <link rel="alternate" type="text/html" href="http://example.org/"/>
  <link rel="self" type="application/atom+xml" href="http://example.org/feeds/23.atom"/>
  <updated>2003-12-13T18:30:02Z</updated>
  <author><name>John Doe</name></author>
  <id>http://example.org/feeds/23</id>

  <entry>
    <title>Atom-Powered Robots Run Amok</title>
    <link href="http://example.org/2003/12/13/atom03"/>
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2003-12-13T18:30:02Z</updated>
    <summary>Some text.</summary>
  </entry>

  <entry>
    <title>A Second Contrived Example</title>
    <link href="http://example.org/2003/12/13/atom03"/>
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2003-12-13T18:30:02Z</updated>
    <summary>Some text.</summary>
    <content type="xhtml" xml:base="http://example.org/">
         <div xmlns="http://www.w3.org/1999/xhtml">
           <p><i>Very fine text.</i></p>
         </div>
    </content>
  </entry>
</feed>
```

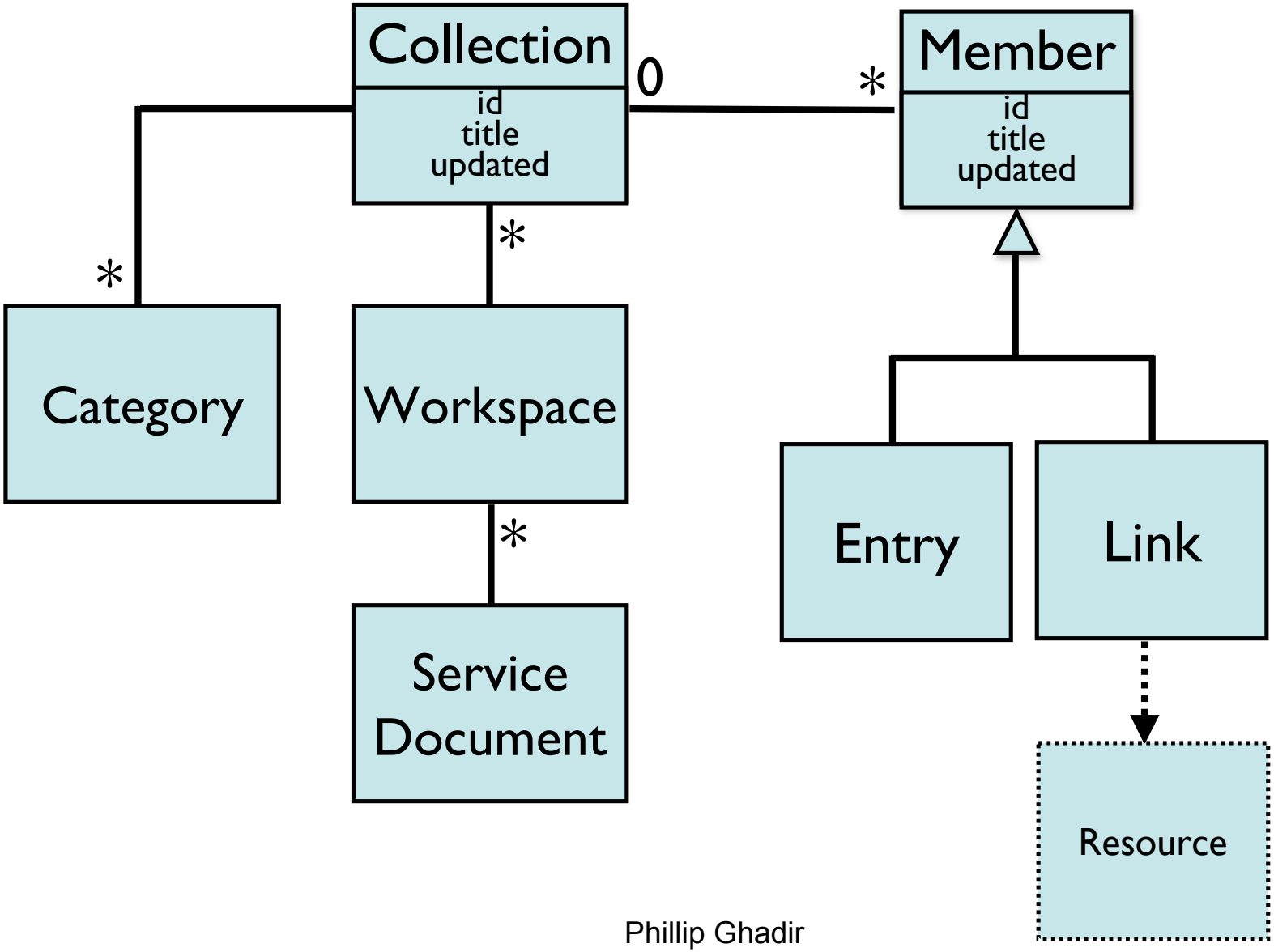
Atom Publishing Protocol

Standardized in: RFC 5023
Namespace: http://www.w3.org/2007/app

RESTful Collections Handling:

- Discovery, Description,
- Retrieval,
- Creation, Editing, Deletion

of Resources

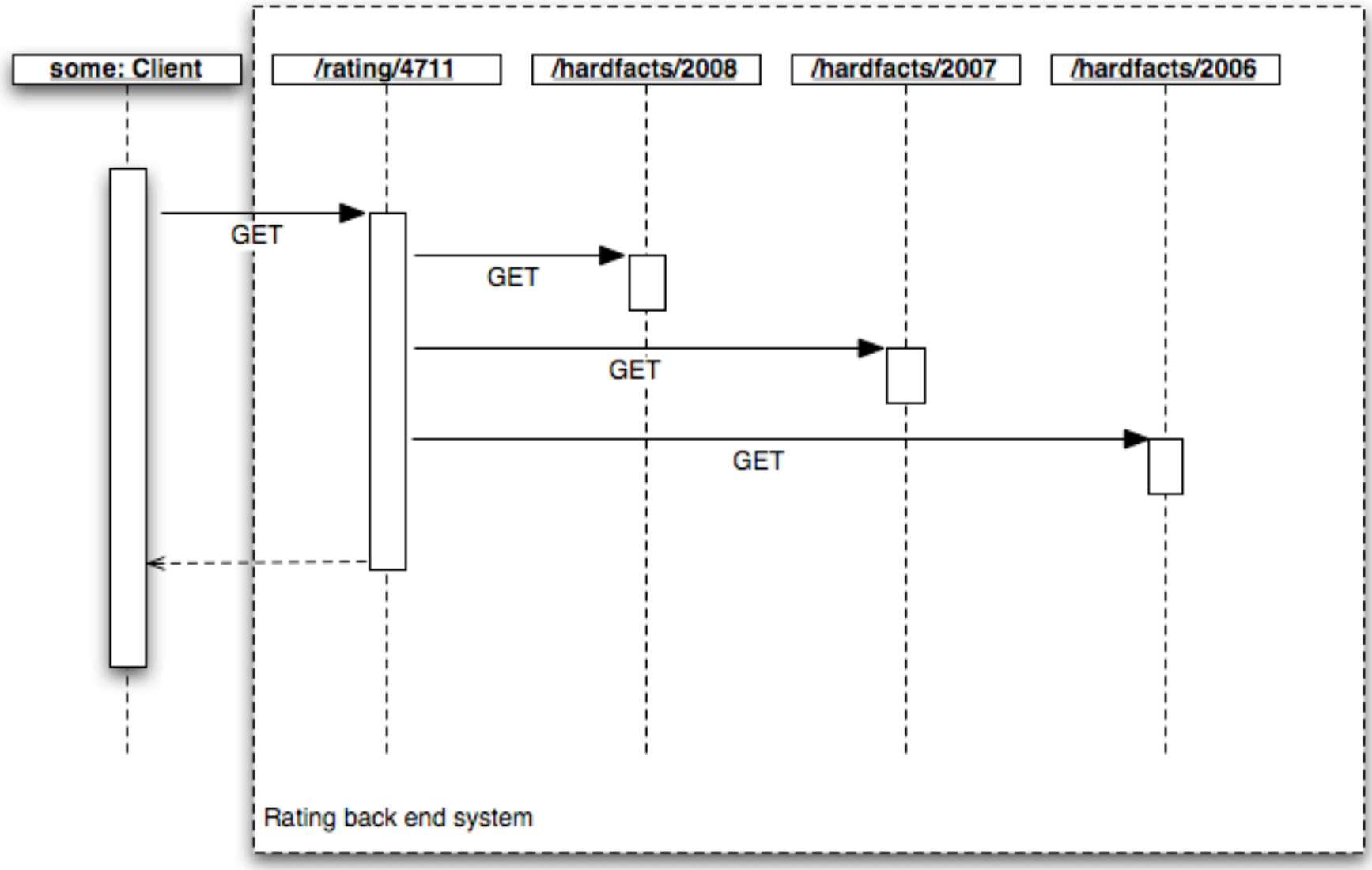

Collection
id
title
updated
0
*
Member
id
title
updated
*
*
Category
Workspace
*
Service
Document
Entry
Link
Resource

Issues with RESTful integration

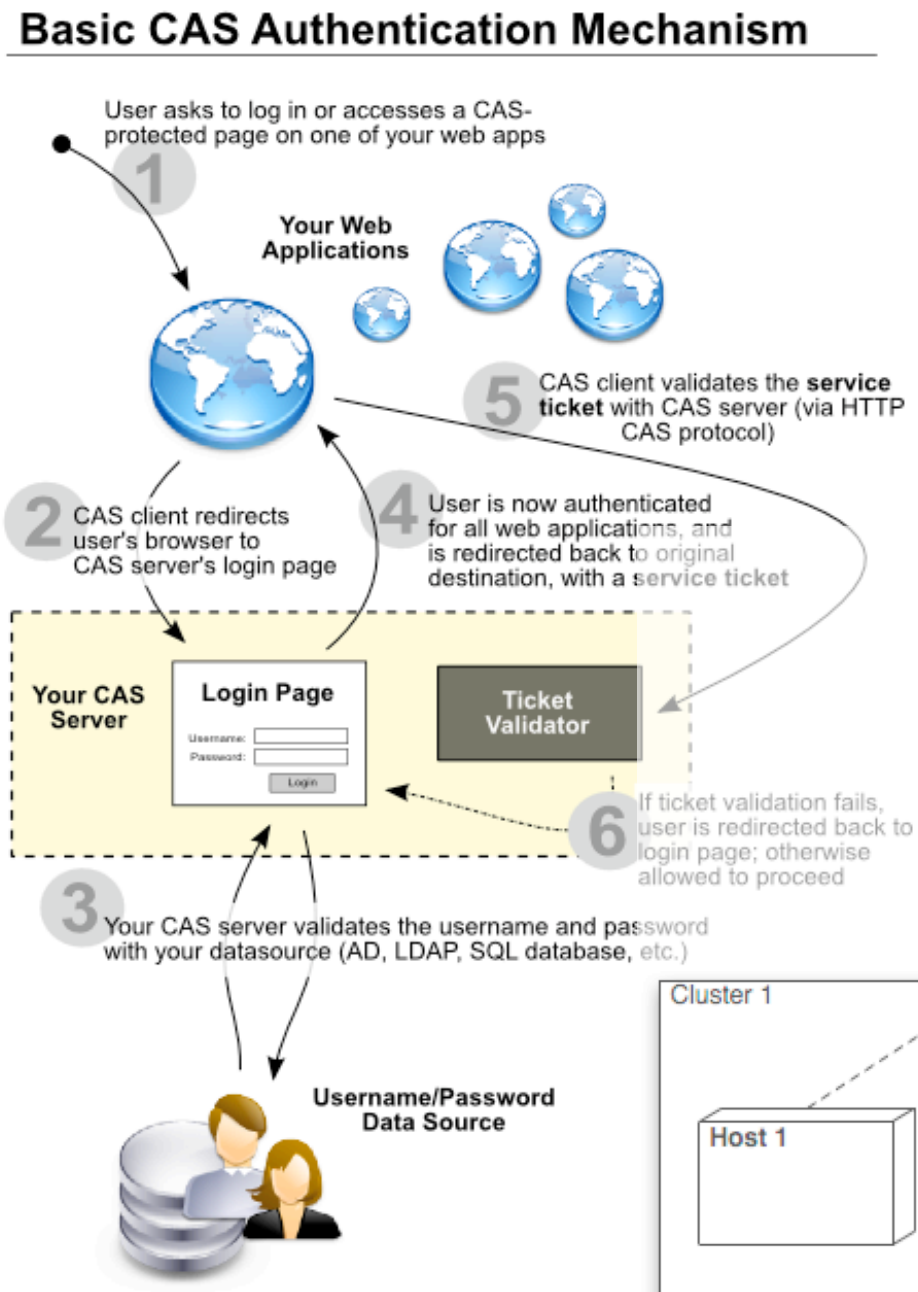
When to use which method?

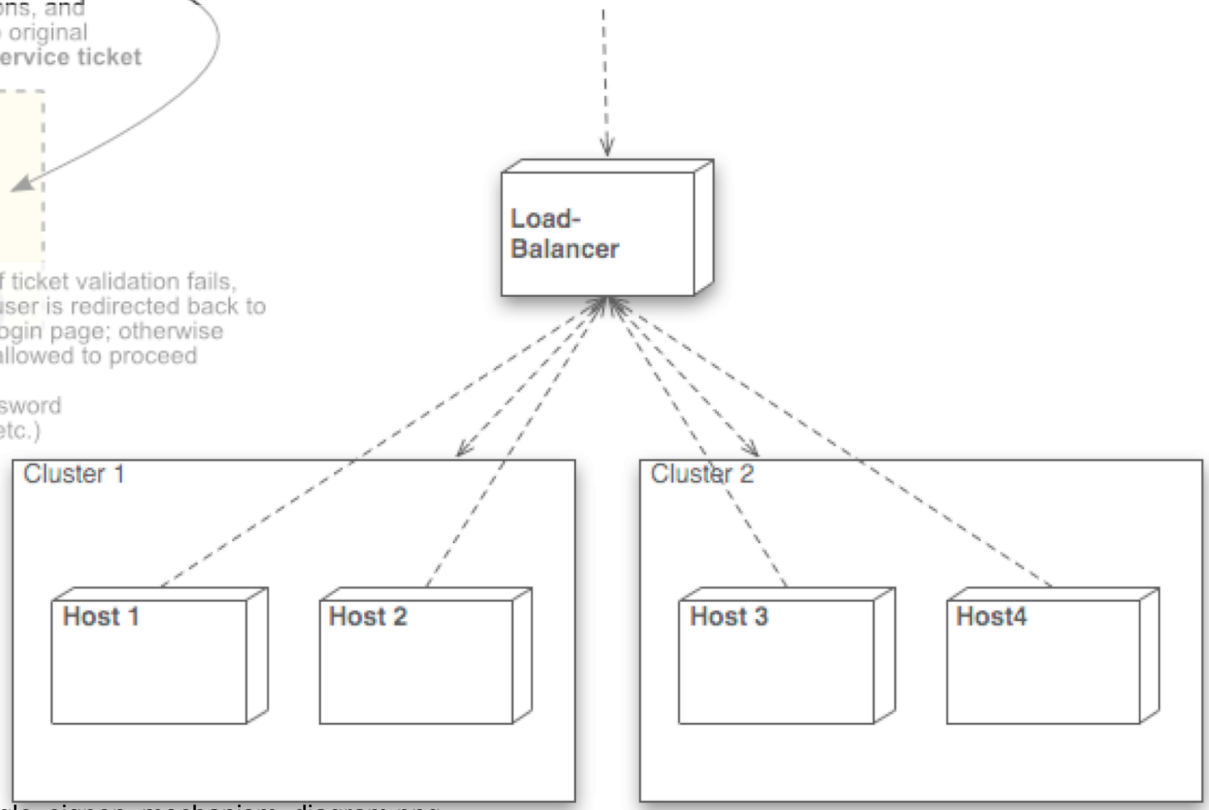
- During elaboration of the integration scenario
- Rule of thumb:
  - Always a GET first
- Exception from that:
  - Create Customer
  - Provide Report for data

Sequential (RESTful) Resource-Building



Infrastructure Dependencies





http://rubycas-server.googlecode.com/files/basic_cas_single_signon_mechanism_diagram.png

REST doesn‘t imply Loose Coupling

- Developers tend to use their beloved tools (here: JAX-B)
- The schemas used for bindings were inherited from the web services interface
- Independent development of client and server side required more effort than expected

Poor Atom & APP Libraries

- Back in 2006 there were:
  - ROME &
  - ROME Propono
- Both lacked support for XML data:
  - Either wrapped in !CDATA
  - Or XML tags escaped (< ... >)
- Building a custom library for AtomPub was a painless „no-brainer“

Missing feature in Atom

- Optimistic Locking requires an attribute like Version or Timestamp
- Atom/AtomPub rely on timestamp while the backend relied on version id

Lessons Learned

- Tight coupling can still be a problem
- Atom libraries had problems with XML-data
- RESTful integration is not a model for building a single application
- Single Sign On required an additional infrastructure component

Benefits

- Very low threshold to access the business data / logic

Many thanks

innoQ Deutschland GmbH
Halskestr. 17
40880 Ratingen
Germany
phillip.ghadir@innoq.com
http://innoq.com
http://ghadir.de/blog