micro-services

adaptive architectures and organisations

ThoughtWorks

james lewis - @boicy - jalewis@thoughtworks.com
One of the goals of SOA has been to decouple services by exchanging human-readable business documents instead of programming parameters. However, in implementing SOA, many businesses have simply used web services to expose the underlying programming models of back-end systems. Procedure oriented integration is nothing more than remote procedure calls implemented via a different protocol. The consequences of this are additional layers of complexity with no improvement in business flexibility. To avoid this, implementers of SOA should first understand the business meaning of their services, then implement human-readable contracts that are independent of legacy system implementation.

All too often caching is an afterthought used to address performance problems with a blanket approach and common cache lifetime. This leads to issues and workarounds. The “time value” of information is inherently linked to the business purpose and hence needs to be captured at the same time as other requirements. We believe thoughtful caching should be addressed early in the project and not just treated as a last minute performance fix.

Starting performance tests late in a project is risky and costly. Very simple performance tests that exercise key parts of the system, run on a regular basis, are good enough to track trends, so we can react if we see a change in performance. Run these tests with your build or as an overnight job and graph the results to create simple performance trending. Complex performance tests in a truly representative environment are still useful, but don’t wait for them to start understanding how the performance of your code is changing.

RESTful APIs have become standard in our industry. A good REST API provides a simple, lightweight means of building customizations and integrations. However, a lot of the quick, high value integrations we’d like to build require knowing when something happened. Consider building an event API, which, when used in conjunction with a REST API, facilitates simple workflow, notification, and synchronization integrations. These integrations often require no more than 20 or 30 lines of code. Often event APIs take the form of a “web hook” or callback mechanism, but don’t be afraid of using a poll-based Asynchronous style either. An Atom event API scales cheaply and gives the client the power to guarantee delivery.
what is this talk about?
what is this talk about?

Evolutionary architecture
what is this talk about?

Evolutionary architecture

Emergent Design
what is this talk about?

Evolutionary architecture

Emergent Design

People and teams
Oh, and...
Oh, and...

Giants, Monoliths, The Tardis, A Volcano, Mushrooms, Turtles and a Cave
are we building systems that are too big?
90% of the TCO of an application is incurred post launch
NHS told to abandon delayed IT project
£12.7bn computer scheme to create patient record system is to be scrapped after years of delays

Denis Campbell, health correspondent
The Guardian, Thursday 22 September 2011
Jump to comments (44)

The NHS has spent billions of pounds on a computerised patient record and booking system, which has never worked properly. Photograph: Martin Godwin
we have to rewrite entire systems every few years
we have to rewrite entire systems every few years

does this make you happy?
scary story
Retail Site

Departure Control
Retail Site

48 Cores
256 GB RAM (NUMA)

~ $1 \times 10^6 \text{ per machine}

Departure Control
You may as well be doing this right?
Airline

Tightly coupled

"Golden Hammer Syndrome"

Single point of failure

Expensive to scale

High operational cost

High cost of failure
hexagonal business capabilities
owning their own data
Each capability decomposed into smaller applications based on your functional and cross-functional needs
Sharing a uniform interface
But what?
But what?
scary story
The stovepipe enterprise

Stovepipes are “systems procured and developed to solve a specific problem, characterized by a limited focus and functionality, and containing data that cannot be easily shared with other systems.” (DOE 1999)

Logic scattered all over the place
Data scattered all over the place

Difficult to predict the effect of changes
Which system is right?
BI / MI almost impossible to get at
Justice Will Take Us Millions Of Intricate Moves

A talk by Leonard Richardson
http://www.crummy.com/

Credit: Flickr user Sakurako Kitsa

November 20, 2008, at QCon.
Leonard Richardson's maturity heuristic

- Level 0: The Swamp of POX
- Level 1: Resources
- Level 2: HTTP Verbs
- Level 3: Hypermedia Controls

Glory of REST
Leonard Richardson's maturity heuristic

Level 0: The Swamp of POX
Level 1: Resources
Level 2: HTTP Verbs
Level 3: Hypermedia Controls

Glory of REST

http://martinfowler.com/articles/richardsonMaturityModel.html
HATE OAS!
behold the twin fists of hypermedia as the engine of application state and tremble
<html>
<body>
 <div>
  <span id="firstName">James</span>
  <span id="lastName">Lewis</span>
 </div>
 <div id="links">
  <a rel="self" href="/users/ae3fc" type="application/vnd.foobar.user+xml">Register User</a>
  <form id="update" method="POST" name="updateForm" action="/users/ae3fc">
   <input id="updatedFirstName" type="text" name="firstName" value="James"/>
   <input type="text" name="lastName" value="lewis"/>
   <input type="submit" name="updateButton" value="update"/>
  </form>
  <a rel="userIndex" href="/users/index">Search Users</a>
 </div>
</body>
</html>
small with a single responsibility
“objects should be no bigger than my head”

my conjecture
and while I have a giant head, its not full of much stuff so thats ok...
as we chunk up domains of abstraction, each domain should be small enough to fit in my head
in this case, it meant 100’s of lines of code per application
a word about reuse...
a word about reuse...
don’t
a word about reuse...

don’t

well not domain code anyway
Use standard application protocols to bridge the semantic gap

TIME FLIES LIKE AN ARROW

TIME LORDS LIKE A TARDIS
The **semantic gap** characterises the difference between two descriptions of an object by different linguistic representations, for instance languages or symbols. In computer science, the concept is relevant whenever ordinary human activities, observations, and tasks are transferred into a computational representation.

The **semantic gap** characterises the difference between two descriptions of an object by different linguistic representations, for instance languages or symbols. In computer science, the concept is relevant whenever ordinary human activities, observations, and tasks are transferred into a computational representation.

The **semantic gap** characterises the difference between two descriptions of an object by different linguistic representations, for instance languages or symbols. In computer science, the concept is relevant whenever ordinary human activities, observations, and tasks are transferred into a computational representation.

RFC 5023 is an example
Atom with AtomPub gives us:
Atom with AtomPub gives us:

Workspaces

<metadata>
<collection>
<collection>
<collection>
Atom with AtomPub gives us:

Workspaces

```
<metadata>
<collection>
<collection>
<collection>
```
Atom with AtomPub gives us:

Workspaces

<metadata>
<collection>
<collection>
<collection>

Collections

<metadata>
<entry>
<entry>
<entry>

composed of
Atom with AtomPub gives us:

Workspaces

<metadata>
<collection>
<collection>
<collection>

Collections

<metadata>
<entry>
<entry>
<entry>
Atom with AtomPub gives us:

Workspaces
- `<metadata>`
- `<collection>`
- `<collection>`
- `<collection>`

Collections
- `<metadata>`
- `<entry>`
- `<entry>`
- `<entry>`

Entries
- `<metadata>`
- `<content>`
- `<state>`
Atom with AtomPub gives us:

- **Workspaces** composed of
  - `<metadata>`
  - `<collection>`
  - `<collection>`
  - `<collection>`

- **Collections** composed of
  - `<metadata>`
  - `<entry>`
  - `<entry>`
  - `<entry>`

  linking to

- **Entries** composed of
  - `<metadata>`
  - `<content>`
  - `<state>`
Atom with AtomPub gives us:

- **Workspaces** composed of:
  - `<metadata>`
  - `<collection>`
  - `<collection>`
  - `<collection>`

- **Collections** composed of:
  - `<metadata>`
  - `<entry>`
  - `<entry>`
  - `<entry>`

- **Entries** composed of:
  - `<metadata>`
  - `<content>`
  - `<state>`

- **user type** linking to content
POST /users HTTP/1.1
Host: example.org
User-Agent: Thingio/1.0
Authorization: Basic ZGFmZnk6c2VjZXJldA==
Content-Type: application/atom+xml
Content-Length: nnn
Slug: First Post

<?xml version="1.0" ?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>James Lewis</title>
  <id>urn:uuid:1225c695-cfb8-4eb9-aaaa-80da344efa6a</id>
  <updated>2003-12-13T18:30:02Z</updated>
  <author><name>James Lewis</name></author>
  <content>stuff</content>
</entry>
independently deployable and scalable
scary story
use the right tool for the job
Design for Replaceability

Cut here
Product Teams organised around Product Lines
Conway’s Law

“…organizations which design systems ... are constrained to produce designs which are copies of the communication structure of those organizations”

Melvin Conway, 1968

“If you ask seven people to write a compiler, you get a seven pass compiler”

Dan North
adaptive architectures
adaptive organisations
The Unix Philosophy

Lions commentary on Unix 2nd edition
but "invented a slightly better one. That finally got changed once more to what we have today. He put pipes into Unix." Thompson also had to change most of the programs, because up until that time, they couldn't take standard input. There wasn't really a need; they all had file arguments. "GREP had a file argument, CAT had a file argument."

The next morning, "we had this orgy of 'one liners.' Everybody had a one liner. Look at this, look at that. ...Everybody started putting forth the UNIX philosophy. Write programs that do one thing and do it well. Write programs to work together. Write programs that handle text streams, because that is a universal interface." Those ideas which add up to the tool approach, were there in some unformed way before pipes, but they really came together afterwards. Pipes became the catalyst for this UNIX philosophy. "The tool thing has turned out to be actually successful. With pipes, many programs could work together, and they could work together at a distance."

The Unix Philosophy

Lions commentary on Unix 2nd edition
Lions commentary on Unix 2nd edition

The Unix Philosophy

The next morning, "we had this orgy of 'one liners.' Everybody had a one liner. Look at this, look at that. ...Everybody started putting forth the UNIX philosophy. Write programs that do one thing and do it well. Write programs to work together. Write programs that handle text streams, because that is a universal interface." Those ideas which add up to the tool approach, were there in some unformed way before pipes, but they really came together afterwards. Pipes became the catalyst for this UNIX philosophy. "The tool thing has turned out to be actually successful. With pipes, many programs could work together, and they could work together at a distance."

The Unix Philosophy :s/pipes/http/
consistent and reinforcing practices
consistent and reinforcing practices

Hexagonal Business capabilities composed of:
consistent and reinforcing practices

Hexagonal Business capabilities composed of:
consistent and reinforcing practices

Hexagonal Business capabilities composed of:

Micro Services that you can
consistent and reinforcing practices

Hexagonal Business capabilities composed of:

Micro Services that you can
consistent and reinforcing practices

Hexagonal Business capabilities composed of:

Micro Services that you can

Rewrite rather than maintain and which form
consistent and reinforcing practices

Hexagonal Business capabilities composed of:

Micro Services that you can

Rewrite rather than maintain and which form
consistent and reinforcing practices

Hexagonal Business capabilities composed of:

Micro Services that you can

Rewrite rather than maintain and which form

A Distributed Bounded Context.
consistent and reinforcing practices

Hexagonal Business capabilities composed of:

Micro Services that you can

Rewrite rather than maintain and which form

A Distributed Bounded Context.
consistent and reinforcing practices

Hexagonal Business capabilities composed of:

Micro Services that you can

Rewrite rather than maintain and which form

A Distributed Bounded Context.

Deployed as containerless OS services
consistent and reinforcing practices

Hexagonal Business capabilities composed of:

Micro Services that you can

Rewrite rather than maintain and which form

A Distributed Bounded Context.

Deployed as containerless OS services
consistent and reinforcing practices

Hexagonal Business capabilities composed of:

Micro Services that you can
Rewrite rather than maintain and which form
A Distributed Bounded Context.
Deployed as containerless OS services
With standardised application protocols and message semantics
consistent and reinforcing practices

Hexagonal Business capabilities composed of:

Micro Services that you can

Rewrite rather than maintain and which form

A Distributed Bounded Context.

Deployed as containerless OS services

With standardised application protocols and message semantics
consistent and reinforcing practices

Hexagonal Business capabilities composed of:

Micro Services that you can

Rewrite rather than maintain and which form

A Distributed Bounded Context.

Deployed as containerless OS services

With standardised application protocols and message semantics

Which are auto-scaling and designed for failure
Architecture is all about trade-offs
Architecture is all about trade-offs

maintainability vs time-to-market
Architecture is all about trade-offs

maintainability vs time-to-market
Architecture is all about trade-offs

maintainability vs time-to-market

throughput vs cost
Architecture is all about trade-offs

maintainability vs time-to-market

throughput vs cost
Architecture is all about trade-offs

maintainability vs time-to-market

throughput vs cost

portability vs deployability
Architecture is all about trade-offs

maintainability vs time-to-market

throughput vs cost

portability vs deployability
Architecture is all about trade-offs

maintainability vs time-to-market

throughput vs cost

portability vs deployability

replacability vs the cost of adding new features
Architecture is all about trade-offs

maintainability vs time-to-market

throughput vs cost

portability vs deployability

replacability vs the cost of adding new features

Evolutionary architecture and emergent design are approaches that maximise your options.
thanks

james lewis - @boicy - jalewis@thoughtworks.com
ThoughtWorks®

is hiring

james lewis - @boicy - jalewis@thoughtworks.com