Platform-Oriented Architecture (POA)

Ted Neward

Neward & Associates

ted@tedneward.com http://blogs.tedneward.com

@tedneward

WTF Are We Doing?

Haven't we been here before?

WTF Are We Doing?

Our industry has had its fair share of "moments"

- Some are excusable
- Others.... Not so much

WTF Are We Doing?

Heard these before?

- "Why isn't software more like hardware? Why must every new development start from scratch? There should be catalogs of software modules, as there are catalogs of VLSI devices: When we build a new system, we should be ordering components from these catalogs and combining them, rather that reinventing the wheel every time. We would write less software, and perhaps do a better job at that which we do develop. Then wouldn't the problems everyone laments - the high costs, the overruns, the lack of reliability - just go away? Why isn't it so?"

WTF Are We Doing?

Heard these before?

- "The fast-moving and demanding world of e-commerce and information technology has put a new kind of pressure on application developers. Enterprise applications have to be designed, built, and produced for less money, faster, and with fewer resources than ever before."

WTF Are We Doing?

Heard these before?

- "To reduce costs and fast-track enterprise application design and development, {thing} provides a component-based approach to the design, development, assembly, and deployment of enterprise applications. {Thing} gives you a multitiered distributed application model, the ability to reuse components, a unified security model, and flexible transaction control. Not only can you deliver innovative customer solutions to market faster than ever, but your platform-independent {thing} component-based solutions are not tied to the products and APIs of any one vendor."

WTF Are We Doing?

Heard these before?

"The three standard and most common business drivers for any {thing} architecture are:

- The ability to respond to the business needs in a timely fashion so that the business can grow
- Improve the customer experience so that customer churn is reduced
- Reduce the cost to add more products, customers or business solutions"

WTF Are We Doing?

Truth is, that's been our goal since 1950

- computers
- functions
- transactional processing monitors
- objects (then, later, distributed objects)
- dynamic languages
- functional languages
- services
- ...

Service-Oriented Architecture (SOA)

Four Tenets of SOA

Service-Oriented Architecture

Service-Oriented Architecture

- a style of architecture focused principally on "services"
  - "SOA is an architectural approach to creating systems built from autonomous services. With SOA, integration becomes forethought rather than afterthought - the end solution is likely to be composed of services developed in different programming languages, hosted on disparate platforms with a variety of security models and business processes."
- popular from 2006 - 2010
- emphasis on encapsulation

Service-Oriented Architecture

The Four Tenets

- boundaries are explicit
- services are autonomous
- services share schema and contract, not class
- service compatibility is determined by policy

Service-Oriented Architecture

SOA Implementations

- implementations typically made heavy use of xml across the wire
  - XML Infoset, XML Schema Description (XSD), SOAP, WSDL
- developers typically steered clear of the XML directly
  - Microsoft WCF, WSE
  - Java: JAX-WS, JAX-M, JAX-B, ...

Service-Oriented Architecture

Results/outcome

- It didn't save the world...
  - ... but it did get us thinking about how the world interops
- Had the right ideas...
  - ... but the implementations went a little crazy
- Standards are good...
  - ... but not 40 of them for a basic implementation!

WTF Are We Doing?

And we keep making the same mistakes over and over again

- technological
- procedural
- people-ological

In each case, we keep

- ... expecting too much (Silver Bullet Syndrome)
- ... making the same mistakes (Santanyana Syndrome)

Fallacies

What are these again?

Fallacies

Fallacies are...

- widely-believed falsehoods
- incorrect assumptions
- lovingly-endorsed "alternative facts"
- mistakes that are all too easy to repeat
- "anti-patterns"

Fallacies

To put it in Deutsch's own words...

"Essentially everyone, ... makes the following assumptions.

"All turn out to be false in the long run and all cause big trouble and painful learning experiences."

Fallacies

In theory...

- ... once you know about them, you can avoid them

In practice...

- ... "Everyone makes these assumptions"
  - largely because they're easy to make
  - and they help us avoid hard truths
  - and painful realizations/pain points

The Fallacies of Distributed Computing

Mistakes are all too easy to repeat

The Fallacies of Distributed Computing

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

The Fallacies of Enterprise Computing

Mistakes are all too easy to repeat

The Fallacies of Enterprise Computing

1. New technology is always better than old technology
2. Enterprise systems are not “distributed systems”
3. Business logic can and should be centralized
4. Data, object or any other kind of model can be centralized
5. The system is monolithic
6. The system is ever finished
7. Vendors can make problems go away
8. Enterprise architecture is the same everywhere
9. Developers need only worry about development problems

What is the answer?

What next?

What is the answer?

More importantly, how do we not throw away what is good?

- functions: structured entry/exit points; data structures
- objects: union of state + behavior
- messaging: decoupled interaction; message/destination/runtime
- REST: limited verb vocabulary; resource-oriented
- SOA: the Four Tenets

The Four Tenets of SOA

a.k.a. SOA in a Nutshell

The Four Tenets of SOA

Four guiding principles around SOA

- Boundaries are explicit
- Services are autonomous
- Services share schema & contract, not class
- Service compatibility is based on policy

The Four Tenets of SOA

Boundaries are explicit

- Know your boundaries
- Provide few well-defined public access paths to your service
- Implementation details should never leak
- Recognize the cost of crossing service boundaries

The Four Tenets of SOA

Services are autonomous

- Deploy and version services independently from the clients
- Assume that once published, interfaces can’t be modified easily
- Adopt a pessimistic outlook.
  - Expect misuse of the service
  - Plan for unreliable levels of service availability and performance

The Four Tenets of SOA

Services share schema & contract, not class

- A service's contract should remain stable
- Contracts should be as explicit as possible to minimize misinterpretation
- Avoid blurring the line between public and private data representations
- A service must be able to convert its native data types to/from some technology-neutral representation
- Revise (version) services when changes to the service's contract are unavoidable

The Four Tenets of SOA

Service compatibility is based on policy

- Behavior should be governed by policy, not code
- Policies should be explicit
- Policies should be discoverable and agreeable/negotiable
- Policies can be amendable w/o recompilation

What is the answer?

What are we really trying to accomplish here?

- code that is...
  - usable
  - extensible
  - discoverable
  - reliable
  - maintainable
  - scalable
  - (... and more)
- ... valuable

What is the answer?

What are we really trying to accomplish here?

We're trying to build platforms

Platform-Oriented Architecture

The foundation for productive development

Platform-Oriented Architecture (POA)

What is a platform?

- "intermediaries that connect two or more distinct groups of users and enable their direct interaction"
  - Harvard Business Review (April 2016)
- "a platform is something that lifts you up and on which others can stand. ... By building a digital platform, other businesses can easily connect their business with yours, build products and services on top of it, and co-create value."
  - Harvard Business Review (Jan 31 2013)

Platform-Oriented Architecture (POA)

POA is...

- ... the logical successor to SOA/REST
- ... a core component of any OS
- ... not tied to any particular OS, language or database
- ... typically distributed (but not always)

Platform-Oriented Architecture (POA)

POA is...

- ... a developer-focused approach
- ... tied to a particular domain
- ... that shares an execution context
  - this notion of shared context is critical

Platform-Oriented Architecture (POA)

In technical terms, POA is...

- ... a communication backplane
  - this doesn't have to be remote
- ... an entity set/definition
  - this is where the domain will have the biggest impact
- ... an agent model (users, actors, systems, ...)
  - auth/auth; who can do what, who can't, etc
  - workflow (if any)
- ... a set of expectations aroundexecution topics
  - sync vs async
  - responsiveness
  - failover/reliability
  - error reporting/handling
  - QoS gurantees

Context

The all-encompassing 'missing link'

Context

"Context"

"the circumstances that form the setting for an event, statement, or idea, and in terms of which it can be fully understood" -- Oxford English Dictionary

Context

"Context"

"(L., *contexere*, "to weave together", from con, "with", and texere, "to weave"). The sum total of meanings (associations, ideas, assumptions, preconceptions, etc.) that (a) are intimately related to a thing, (b) provide the origins for and (c) influence our attitudes, perspectives, judgments, and knowlege of that thing." -- Dictionary of Philosophy

Context

"Context"

"an agent's understanding of the relationships between the elements of the agent's environment." -- *Understanding Context* (OReilly and Associates)

Context

Major components to a Context

- Agents
- Execution semantic agreement
- Relationship between the elements
  - how these elements relate to one another
- Environment
  - the space between and surrounding the elements

POA: Technical Details

How do I build one?

POA: Technical

Building a platform is actually not all that hard

- Define/SWAG out platform requirements
- Choose an implementation set
- Decide on the communications backplane
- Define the entity models
- Establish how actors are authenticated/authorized
- Build out the DevOps pipeline

POA: Technical

Define/SWAG out platform requirements

- What is the domain model? How "far" will it reach?
- How fast? (Performance)
- How much? (Scalability)
- How safe? (Security)
- How broad? (Interoperability)
- How safe? (Reliability)
- How autonomous? (Recoverability)
- ...

POA: Technical

Choose an implementation set

- OS/Language platform
  - Windows, macOS, Linux, whatever
  - C#/.NET, Java/JVM, PHP, whatever
  - Containers help make the OS irrelevant
  - Some languares run on VMs that make the OS (mostly) irrelevant
  - Some languages stretch across platforms/VMs
- What does your team know?
- Which best meet the team's coding style?

POA: Technical

Decide on the communications backplane

- The hot favorite at the moment is "REST"
  - which is kinda ironic, since most aren't doing HATEOAS
- The greater the need for "reach", the more generic the comm backplane
  - this is why HTTP is such a favorite
- Don't neglect other comm backplanes
  - Foreign Function Interfaces (Return of the C-die!)
  - Messaging channels (files, shared database, MOM systems, etc)
  - Wire protocols (BEEP, SOAP, Protobufs, etc)
  - RPC toolkits (CORBA, Apache Thrift, etc)
- [illegible]

POA: Technical

Define the entity models

- What are we exchanging?
  - don't define these as "objects"; that was SOAP's mistake
- How is this documented/discovered?
  - keep in mind the next question will be "can I code-gen this?"
- How will we extend/modify these?
  - versioning will be a key aspect to this
- How will clients produce/consume instances of these?
  - keep in mind the next question will be "can I code-gen this?"

POA: Technical

Establish how actors establish themselves

- How does an actor identify itself? (Authentication)
- How do we determine an actor's abilities (Authorization)
  - note that these two don't have to be the same technology decision
- How will this change over time?

POA: Technical

Build out the DevOps pipeline

- Testing
  - both unit-tests and "black-box" comm-originating tests
- Deploying
  - containers
- Versioning/modifying
  - including the comm backplane interface

POA: Technical

Implementation details

- Persistence
  - RDBMS, DocDBMS, GraphDBMS, KVDBMS, ...
- Language/language-type
  - OO, functional, logical, ...
- Cloud
  - Location transparency
- Deployment granularity
  - mapping of comm interface to deployment unit is opaque

Summary

Wrapping it all up

Summary

POA is our past, and our future

- Before, platforms were defined implicitly
  - I'm saying we need to define them explicitly now
- Before, platforms were accidental
  - I'm saying we need to define them deliberately

Credentials

Who is this guy?

- Director, Technology Culture -- Rocket Mortgage
- Principal -- Neward & Associates
- Author
  - *Professional F# 2.0* (w/Erickson, et al; Wrox, 2010)
  - *Effective Enterprise Java* (Addison-Wesley, 2004)
  - *SSCLI Essentials* (w/Stutz, et al; OReilly, 2003)
  - *Server-Based Java Programming* (Manning, 2000)
- See http://www.newardassociates.com