Achieving Success in Open Software Ecosystems: The Role of Architectural Styles

Richard N. Taylor
University of California, Irvine
An architectural style is a named collection of architectural design decisions that (1) are applicable in a given development context, (2) constrain architectural design decisions that are specific to a particular system within that context, and (3) elicit beneficial qualities in each resulting system.
Unpacking the Title (2): Success? What’s That?

- Decreased time to market?
- Decreased production cost?
- Widespread use?
- Profit?
- Adaptability?
Unpacking the Title (3): Ecosystems

- **Product Line**: separate products that share significant *technical* commonality in components and structure
  - Examples: Philips TV sets; the iPhone family

- **Ecosystem**: a complex system composed of multiple organisms, interacting with it and with each other
  - Examples: Amazon, Photoshop, Apple’s iOS Apps
Success Factors for Product Lines

**Business**
- Business goals motivate
- Minimize costs: reuse assets when possible
- Maximize market: develop many related applications

**Domain**
- Constrains the problem space enabling focused development

**Technology**
- Technological solutions—tools, patterns, architectures & styles, legacy systems—provide a non-trivial, sustainable basis for success
Product Lines v. Ecosystems

- **Product Lines**
  - usually **single** agency (a.k.a. development organization)
  - success criteria: reduced dev costs; faster time to market; higher quality (esp. initial quality of each product)

- **Ecosystems:**
  - **multi**-agency
  - widely varying success criteria: profit, visibility, reach, “coolness,” mindshare, functionality
Ecosystems According to Jan Bosch

“A software ecosystem consists of a software platform, a set of internal and external developers, a community of domain experts and a community of users that compose relevant solution elements to satisfy their needs.
Platform-based Ecosystems

- 1 big vendor and lots of “hangers-on”
  - SAP
  - Facebook
  - SalesForce
  - eBay
  - Amazon

- Revenue model: biased towards the platform vendor
  - AutoCad/AutoDesk
  - Microsoft
  - Adobe Flash
  - Photoshop
The simplest “style”: APIs
- “value-adding” products call into the platform
- Note: The bigger the vendor the less elegant the APIs need to be; the less there needs to be any evidence of a clear, coherent style
The platform provides a language for value-adding products
- Richer, more coherent extension mechanism than APIs
- Flash ActionScript
- Visual Basic for Applications (VBA)
The Plug-In Architectural Style

Example: Eclipse
Example: Photoshop

Adobe Photoshop Lightroom 4 software includes an extensive and powerful array of tools for managing, editing, and showcasing your images. Even better, Lightroom is highly extensible. Its plug-in architecture allows third-party developers to create a huge variety of software plug-ins that let you add new features and capabilities to the already rich Lightroom toolset.
Example: Apple iOS Apps

“MVC is central to a good design for any iOS app or Mac app.”

The Most Important Design Pattern: Model–View–Controller

The Model–View–Controller design pattern (commonly known as MVC) assigns objects in an app one of three roles: model, view, or controller. The pattern defines not only the roles objects play in the app, it defines the way objects communicate with each other. Each of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries. The collection of objects of a certain MVC type in an app is sometimes referred to as a layer—for example, a model layer.

Available on the App Store
The Model-View-Controller Style

- **View** (Encapsulation of display choices)
- **Controller** (Encapsulation of interaction semantics)
- **Model** (Encapsulation of information)

Diagram:

- Graphical Display
- User-interface Events
- Model
- View
- Controller
iOS App Design & Dev

- Architectural Styles (aka Design Patterns)
  - MVC
  - Event Notification
- Frameworks
  - Cocoa and Quartz
  - Foundation, UIKit, Core Graphics
- Guidance and guidelines
  - “iOS Human Interface Guidelines”
- XCode SDK
Event-based Styles

- Key benefit: very strong decoupling of components
Event-based Ecosystems

- **TIBCO** and financial trading systems
  - (Also used as the backbone for FedEx tracking)
- The event system as “the platform”
  - Routing services
  - Event-definition language, standards, or framework
What is a Platform?

- Bosch: A vendor’s main product, holding state, providing key services, “the brand”, ...
- Is GPS a Platform?
- Aircraft transponder vectors?
- Financial information?
  - The Financial Information eXchange (FIX) Protocol is a messaging standard developed specifically for the real-time electronic exchange of securities transactions. FIX is a public-domain specification owned and maintained by FIX Protocol, Ltd.

A platform is a shared understanding. But just how much does one have to share in order to have an “understanding”?
Decentralized Ecosystems

- The essence of decentralization: multiple, independent spheres of authority
  - “Openness” is not necessary for decentralization

- Multiple domains
  - e-commerce, healthcare, defense, space systems, power grids, highways, ...

- Multiple objectives within a domain
  - Not all of which are shared; Not everyone is aligned
  - Not all of which are compatible
  - Not all of which are benign

- The presence of competition virtually guarantees non-alignment
Platforms for Decentralized Ecosystems

- Given independence, competing interests, and inherent risk, what suffices as a platform?

- Answer #1: Weak standards, like FIX or GPS

- Answer #2: Minimal protocols, like TCP

- But for substantive interplay?
Web Services, Take 1

- SOAP over HTTP plus WSDL and others
- In essence:
  - APIs
  - Simple transport protocol (using HTTP to do RPC)
- Not particularly “successful” — though widely used
- Later improved via Enterprise Service Buses
  - Providing a more event-based interaction platform
Web Services, Take 2: RESTful

- “The world of web services has been on fast track to supernova ever since the architect astronauts spotted another meme to rocket out of pragmatism and into the universe of enterprises. But, thankfully, all is not lost. A renaissance of HTTP appreciation is building and, under the banner of REST, shows a credible alternative to what the merchants of complexity are trying to ram down everyone’s throats; a simple set of principles that every day developers can use to connect applications in a style native to the Web.” -- David Heinemeier Hansson, Foreword to RESTful Web Services.
RESTful Design Principles

- Addressability of **information** (via URLs)
- Context-free interactions (application state on the client; resource state on the server)
- Links and connectedness (**HATEOAS**)
- Appropriate use of the uniform interface (i.e. GET, PUT, DELETE, HEAD, POST)

Example users: Amazon S3, IBM, Oracle
Many Extensions, Many Uses

- E.g. AJAX, Rails (Platform-building platform)
- Blending styles: language-based extension, mobile code, MVC, ...
- (Note: RPC-over-HTTP, or SOAP-over-HTTP is not REST)
Rails Ecosystem

- REST
- Language-extension
- MVC

Getting Started with Rails
This guide covers getting up and running with Ruby on Rails.

After reading this guide, you will know:
- How to install Rails, create a new Rails application, and connect your application to a database.
- The general layout of a Rails application.
- The basic principles of MVC (Model, View, Controller) and RESTful design.
- How to quickly generate the starting pieces of a Rails application.

Supporting 200,000+ websites
The Nasty Parts of Decentralized Systems

- Security
- Trust
- Adaptation: Innumerable requests for change and specialization

Thus, what style for open, decentralized, critical ecosystems, with such risks and demands?
COmputAtional State Transfer (COAST)

- The COAST style:
  - For decentralized applications (the context)
  - Based on mobile computations, communication constraints, Principle of Least Authority* (the constraints)
  - Yields dynamic adaptability, pervasive security, ... (some of the beneficial qualities as architectural consequences)

*All men are by nature fond of power, unwilling to part with the possession of it...[thus]...no man, or body of men, ought to be entrusted with the united powers of Government, or more command than is absolutely necessary to discharge the particular office committed to him" — Anonymous, 1776
The Key Style Insights

- Architecture can induce security
- Architecture can induce adaptivity
- Adaptivity and security need not be at odds
  - Architecture can embody capability-based security
Revenue Models in COAST: A “Level Playing Field” Ecosystem

- Resource-limiting, loggable CURLs
  - Per user
  - Per use
  - Resource capped
  - Time-outs

- Revocable

COAST Status

- Full infrastructure in place for evaluative applications
- Seeking application partners
- Working an electronic healthcare record scenario
Why Are Styles *So* Important?

- Styles are a key element in Product Line *and* Ecosystem success

Why? Styles carry experience, aid communication, provide vocabulary, speed design, yield predictable benefits
Take-Aways

- Take-away #1: A well-chosen, well-designed architectural style is key to a successful ecosystem
- Take-away #2: Multi-agency, decentralized applications offer special challenges and demand new approaches
- Take-away #3: COAST offers end-to-end security, client-initiated customization, and a flexible revenue model
Acknowledgments

✦ Eric M. Dashofy
  The Aerospace Corporation

✦ Michael M. Gorlick
  The Aerospace Corporation and UC Irvine