



# Developing in a Polyglot World

**Craig R. McClanahan**  
Senior Staff Engineer  
Sun Microsystems, Inc.



# Agenda

- Hello
- Remember When
- A Polyglot World
- Next Generation Web Drivers
- What Should We Do?
- Summary

**Hello**

# Hello

- Hi, I'm Craig
- Around the Java web tier for many years:
  - > Apache Tomcat (*Catalina* servlet container)
  - > JavaServer Faces 1.0 (JSR-127)
  - > Sun Java Studio Creator (now part of NetBeans)
- Also know in open source circles
  - > Original developer, Apache Struts
- Currently working on a large scale *next generation web* application and architecture

# Remember When

# Remember When ...

- I could learn *one* technology stack
  - > Platform, OS, Language ...
- And *one* application architecture model
  - > Batch, Local PC, Client-Server, Rich Client
- And I could cruise through my career
  - > An expert in my narrowly chosen focus area
  - > Without having to sweat learning anything new

# No, I Can't Either

- Best shot was in the early days of computing
  - > “All the world is a batch queue”
  - > “COBOL is the answer, now what is the question?”
    - Although, for me, it was actually RPG II :-)
- Applications tended to be introverted
  - > Basically self contained within the data center
  - > Relatively little import/export of data
- But even 25 years ago, signs of an emerging *polyglot world* were emerging

# A Polyglot World

# Polyglot?

# We Will Look At Three Definitions

- Composed of several languages
- Program valid in multiple languages
- Person fluent in multiple languages

# Composed Of Several Languages

- From the beginnings of the Internet:
  - > People started sharing programs as well as data
  - > Started caring about multiple platforms
- Web applications today certainly qualify:
  - > HTML (and Dynamic HTML)
  - > CSS
  - > JavaScript (and Ajax Callbacks)
  - > Server side web frameworks in some other language

# Program Valid In Multiple Languages

- Most important trend in next generation web:
  - > It's the *data*, not the *content*, that is important
- A flashy browser UI will attract human users:
  - > Who may already prefer other sites for interactions
  - > Even if they like the information that your site offers
- An elegant API will attract developers:
  - > Who will create flashy browser UIs for you
- An elegant API will also attract mash-up users:
  - > Who will recognize you as market leader for this content
- What does this have to do with us, though?

# Program Valid In Multiple Languages

- Consider XML ... as a programming language
  - > Explicitly or implicitly specify operations
  - > Provide data (or reference to data) to operate on
- Is a SOAP message a *program*?
- How about an Ant build.xml file?
- Focus on providing XML (or JSON) content:
  - > Free of assumptions about underlying implementations
  - > Clients and servers can be implemented independently
- Pick best implementation technologies at each end

# Person Fluent In Multiple Languages

- It is straightforward to become an expert at a particular narrow branch of technology
  - > If that technology is popular, you have a career path
  - > At least for a while
- It is more difficult to master multiple technologies
  - > Ongoing commitment to learning
  - > Must be able to synthesize to gain maximum benefits
  - > Can lead to advanced positions, higher pay
- Example: The project I'm working on needs:
  - > Ruby on Rails + DHTML + Ajax + CSS + Database + REST + Apache Configuration + ...

# Next Generation Web Drivers

# Next Generation Web Drivers

## Four Key Drivers

- Enhanced user interfaces
- Lightweight programming models
- Web as a platform
- The read/write web

# Next Generation Web Drivers

## Enhanced User Interfaces

- Traditional HTML user interaction model:
  - > Fill out a form and submit it
  - > Complete page gets redisplayed
- Limited usability compared to locally installed apps
- Moving towards a solution:
  - > Ajax for background client-server interaction
  - > Dynamic HTML for partial page updates

# Next Generation Web Drivers

## Lightweight Programming Models

- Emotional backlash against complexity:
  - > Complexity should not be *necessary*
  - > Complexity reduces *agility*
- Emotional backlash against configuration files:
  - > XML configuration files – *poster child* “problem”
  - > Trend towards *convention over configuration*
  - > But what about toolability?

# Next Generation Web Drivers

## Lightweight Programming Models

- Ruby On Rails community view of the WS-\* stack:



# Next Generation Web Drivers

## Lightweight Programming Models

- Increasing preference for *iterative development*:
  - > Marketplace demands quick turnaround
  - > *Eternal Beta* versus *Big Bang* release cycle:
    - Every 3-14 days versus 6-18 months
    - New features and/or bugfixes in each release
  - > *Many* implications for current development processes

# Next Generation Web Drivers

## Web As A Platform – Technology Basis

- HTTP is becoming ubiquitous
- Internet connectivity (at least part time) can be assumed
- Building *Internet scale* applications:
  - > Is still *hard work*
  - > But is no longer *rocket science*

# Next Generation Web Drivers

## Web As A Platform – Social Basis

- The *network effect*:
  - > The value of a service increases exponentially as the number of participants increases
- Users want “more and better”:
  - > *Identity* – Remember who I am and what I want
  - > *Flexibility* – Interacting when and where I want
  - > *Ubiquity* – No matter where the content is found

# Next Generation Web Drivers

## Web As A Platform – Software As A Service

- Eliminate the need to package, download, install
  - > And no need to install updates later, either
- Eliminate the need to live within the limits of the client's hardware capabilities
- Reduce *local islands* of important data
  - > “80% of the world's business logic and data is embedded in Microsoft Excel spreadsheets”

# Next Generation Web Drivers

## The Read/Write Web – The Original Business Model

- *The web as a one way channel*
- Similar in concept to other media
  - > Newspapers, radio, television
- Value of the channel is *content management*
  - > Syndication
  - > Filtering
  - > Reviewing
  - > Sponsorships

# Next Generation Web Drivers

## The Read/Write Web – The Obvious Next Step

- *The web as a bulletin board*
- Companies can publish advertising brochures
  - > At relatively low cost compared to other media
- Tech savvy individual can create sites as well
  - > Development of user friendly tools (blogs, wikis, etc.) expands this capability much more broadly
- Hyperlinks make *search* a viable business model
  - > And further blur perceived boundaries between apps

# Next Generation Web Drivers

## The Read/Write Web – Nothing Stays The Same

- *The web as a collaborative experience*
- Users can *provide* content as well as consume it
- Users can *comment on* and *rate* content
- Value of an application results from synergy
  - > Content provided by the application itself
  - > Content provided directly by users
  - > Content provided as commentary by users
  - > Content aggregated from external applications
  - > Coupled with functionality that mixes and presents everything in interesting ways

# What Should We Do?

# What Should We Do?

## *A Ten Best List Set Of Suggestions*

- *If we want to build a next generation web application of our own ...*
- And deploy it to potentially *millions* of users around the world ...
- Here is what we should do

# What Should We Do?

## (10) Expose Data and Logic as Services

- The *content* presented by your application is more important than how it is *presented*
- What your application can *do* is more important than the *user interface*
- Your content and functionality can become even more valuable when other applications use it
- So, make your content and business logic available:
  - > Use REST based services when you **can**
  - > Use SOAP based services when you **must**

# What Should We Do?

## (9) Incorporate External Content

- Essentially all useful content is available *somewhere* on the Internet
- No single application or database can ever contain everything
- Compelling value can be provided by *combining* available content
  - > From multiple sources
  - > In innovative ways

# What Should We Do?

## (8) Seek Quality Of Service Deals From Sources

- Applications that incorporate external content become dependent on the quality of service of your content providers
- Negotiate contracts with key providers:
  - > Availability guarantees
  - > Performance provisions
  - > API and data format stability

# What Should We Do?

## (7) Give Quality Of Service Deals To Users

- Downstream applications will also become dependent upon your content
  - > Their developers are listening to this advice also
- External consumers will ask you for *quality of service* guarantees
  - > **This is a good thing – they will pay for it!**
- Be prepared to commit to quality
  - > And to live up to your commitments

# What Should We Do?

## (6) Adopt Agile Development Processes

- Traditional application development:
  - > Decades-old *waterfall* model
  - > *Big bang* release every 12-18 months
- *Next generation web* development:
  - > Continuous *iterative improvements* development model
  - > *Incremental* releases every 7-14 days
- Guess which model our users prefer?

# What Should We Do?

## (5) Embrace Test Driven Development

- Biggest objection to agile development model often comes from *Quality Assurance* group:
  - > How can we maintain quality ...
  - > Across a complex matrix of technologies ...
  - > When the testing itself can take weeks?
- *Reduce* release testing workload by incorporating aggressive unit and functional testing into our development processes
  - > Release testing can focus on integration and usability testing

# What Should We Do?

## (4) Architect Your Application For Scalability

- **Simple** web applications:
  - > Intermix presentation and business logic ...
  - > In a single monolithic architecture ...
  - > And are very difficult to scale
- That is fine for 10 user departmental applications
- **Scalable** web applications:
  - > Separate persistence, model, logic, and view ...
  - > Into layers that can be *independently* scaled ...
  - > So we can add resources as needed for bottlenecks

# What Should We Do?

## (3) Embrace Heterogeneous Technologies

- Large applications **require** a variety of technologies to meet requirements:
  - > Firewalls, load balancers, computing power, storage
  - > Authentication, authorization, personalization
  - > Caching, transactions, failover
- Agile applications apply *appropriate* technologies (such as implementation language) to each need
- Beware of *one size fits all* technology stacks
- Leverage services to insulate layers
  - > Both internally and to support mashups

# What Should We Do?

## (2) Reach Out To Mobile Clients

- The install base of potential client devices will be:
  - > Browsers – millions
  - > Mobile devices – ten of millions
- If you have been following these recommendations
  - > It will be easy to add another view layer
  - > And *share* your existing services
- Watch for upcoming innovations around:
  - > Intermittently connected clients
  - > Offline local storage

# What Should We Do?

## (1) Enable User Provided Content

- The big *next generation web* winners will:
  - > Enable users to *provide* your content
  - > Enable users to *comment on* your content
- Participating in mashups counts!
  - > Inbound – Create innovative combinations
  - > Outbound – Become domain market leader
- Users like to *participate*, not just *view*

# Summary

# Summary

## Where Have We Come From?

- The days are gone when one technology stack could satisfy all requirements:
  - > Even Java
  - > Even Solaris

# Summary

## Where Are We Today?

- Today we are developing in a polyglot world
- Must be able to utilize multiple technologies
  - > Sometimes interchangeable
  - > And therefore must be selected wisely
- Developers who exhibit polyglot skills are more economically valuable than those who do not

# Summary

## What Should We Be Doing?

- Design applications around:
  - > *Scalability*
  - > *Sharing content*
- Refine development processes to enable turnaround within *weeks* instead of *years*
- Expand vision of *application* beyond **our** content delivered to **browsers**
- (0) Be passionate and have fun



# Developing In A Polyglot World

**Craig R. McClanahan**  
Craig.McClanahan@sun.com