Java Enterprise Application Standards and Why the Industry Moved to Lightweight Open Source

Rod Johnson

CEO, SpringSource

Topic

- *Not* a talk about particular technologies
- An attempt to explain how bad ideas can flourish (with examples), and how to avoid repeating the mistakes of history

A reference model

A Dangerous Myth

The long-term narrative

- Two major factors help us to understand the history of enterprise Java
  - Developer empowerment
  - Economic fluctuations

Java started out simple

- Java started off as a simple language
  - C++ --
- Even today, Java is not particularly complex

Where did it all go wrong?

- Excessive complexity was introduced along the way
  - "Enterprise" Java lost touch with its roots

The growth of the *Complexity Industry*

- The "E" word
  - "Enterprise"
- Ex CORBA architects who wanted to have any shot at some of their fetishes
- Embrace by huge companies (software vendors and SIs) who helped to define platform
- Resume padding among developers and architects
- "No pain no gain" fallacy
- Economic exuberance

A BRIEF HISTORY OF J(2)EE

The Three ages of enterprise Java

- Before J2EE
- The glory days of J2EE
- The post Java EE era

Before J2EE

- Mid 1990s
  - Java gradually moves to the server side
- Largely unregulated
- Many competing products in different areas
  - NetDynamics
  - TopLink
  - Silverstream
  - Persistence PowerTier
  - Apple WebObjects

Before J2EE…

- Good and Bad
  - Innovation and choice of approaches
  - Applications needed to use in-house frameworks, but many companies got good results based on Servlet API
  - Fragmented server-side market
  - Real danger of vendor lock-in
  - Many solutions very expensive
    - No impact from open source

The Glory Days of J2EE

- 1999-2003
  - The JCP becomes dominant in the space
- TopLink and other "non-standard" technologies cannot compete with J2EE standards
  - ORM versus EJB entity beans
  - Velocity vs JSP
  - WebObjects vs web tier

The Glory Days of J2EE

- Good and bad
  - A market is created
  - Vendor lock-in is reduced, but not eliminated
  - Increasing thought control strangles innovation
  - Flaws in the model take years to be resolved

The Glory Days of J2EE

- Enterprise Java gains a reputation for complexity
- Many projects fail due to flaws in the platform
- Greatest offender is the programming model, based on EJB

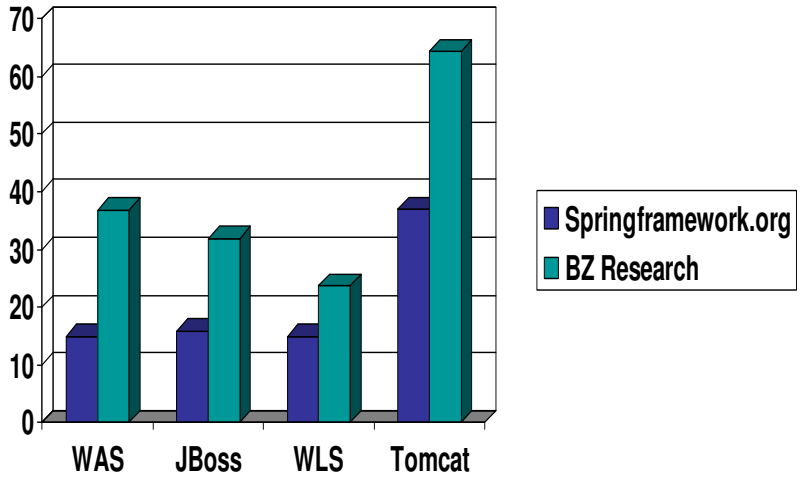
The Glory Days of J2EE/aka The Dark Ages

- Some *horrible* technologies and fundamental flaws in the platform
  - Idea that all business objects (EJBs) should be distributed
  - Naïve ORM solution (CMP entity beans)
  - Grossly excessive complexity across the platform

The decline of J2EE

- 2003-
- Move away from traditional application server towards lighter-weight solutions such as Tomcat
  - Tomcat now clear leader in enterprise Java deployments

- Proportion of enterprise Java users using Tomcat



The Post J2EE Era

- JCP specifications now just one input
- More grass roots innovation

Major driver: The rise of open source

- Fewer and fewer organizations develop enterprise Java applications without using open source
  - Those that do face increasing competitive disadvantage
- Numerous open source projects help to shape the future
  - Eclipse
  - Spring
  - AspectJ

SOME OF THE TRAPS THAT PRODUCED THE PROBLEMS OF "OLD J2EE"

Traps/ideas that created complexity

1. Design by committee is desirable
    1. Everything has to be standardized
2. Tools can make excessive complexity acceptable
3. Lack of openness to other platforms, sources of ideas
4. Developers are stupid and must be controlled
5. Complex solutions are better
6. “I need a solution that enables my enterprise SOA/XYZ strategy”

Myth: Design by committee can work

- You've heard of the *Cathedral* and *Bazaar* as sources of software

The Commissar

- Java has its own somewhat unique model
  - The Commissar
- In this model, the politburo knows what's best for the proletariat (you)

The Commissar Knows Best

- **Essentially, design by committee**
- JCP expert groups talk largely in private
- Typically composed of software vendors
- Relatively slow pace of change, like Soviet 5 year plans

Why standards are needed

- Standards can create markets
- Standards can provide a base on which competing open source and commercial alternatives can flourish
  - JTA
  - Servlet API
  - JMS
- Standards can protect customers from lock in to a proprietary technology
- To ensure interoperability
  - Web Services
  - IIOP

How much standardization is too much?

- In the Java world we have an unhealthy obsession with standards
- Desire to standardize *everything*
- Failure to critically evaluate standard technologies

Where Standards Don't Work

- Jim Waldo (Sun Distinguished Engineer)
  - *Kowtowing to the god of standards is, I believe, doing great damage to our industry, or craft, and our science. It turns technical discussions into political debates. It misunderstands the role that standards have played in the past. Worst of all, it is leading us down absurd technological paths in the quest to follow standards which have never been implemented and aren't the right thing for the problems at hand.*

Where standards don't work

- CORBA history (1990s)
  - Death by committee
  - Attempts to innovate by committee (distributed persistent objects)
- When they're too slow
- When they're divorced from reality
  - Ivory castle
- When they are about politics, not technology

Case study in failure

- CMP entity beans
- Grew out of CORBA
- No implementations when specification was released
- Designed by smart people with little domain knowledge
- Ignored successful products like TopLink
- Inefficient, barely useable specifications
  - Problems were available to anyone who wrote Hello World

The standards check list

1. Will the pace of change and innovation required by met in a standards process cycle
2. Do we benefit from competing implementations?
3. Does this affect wire protocols (in which standards are probably outside Java)
4. Is there an entrenched open source solution, in which case competition may not occur?
5. Is the field mature and well understood
    1. Has the proposal been tested in the market?
    2. Do *not* want design by committee?
6. Is the standards committee representative of the users of the technology?

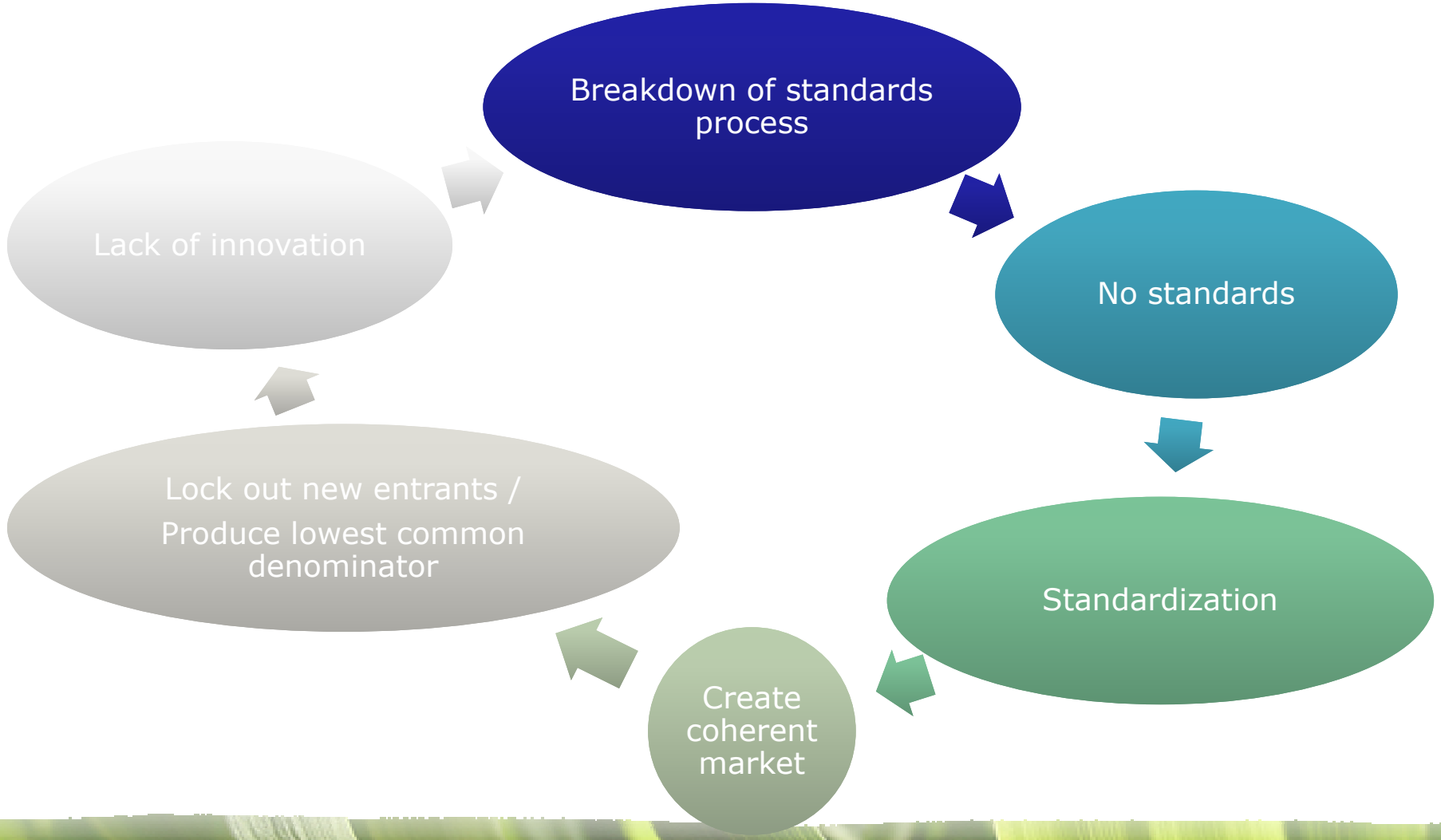
Does the standard exist to lock out newcomers?

- When standards become too complex, like J2EE, they effectively lock out new entrants and benefit existing franchisees, not consumers

Things need to become faster

- Competition and experimentation needs to occur rapidly
- Technology change and the increasing pace of business leaves 2-3 years committee-driven cycles looking less and less relevant

The standardization cycle



The appropriate role of the JCP

- The JCP is unlikely to produce innovation but should focus on what it can succeed at
  - Creating a market where innovators can compete above fundamental stanards
- Innovation by committee is a bad idea, and has traditionally produced poor results

Myth: Tools can conceal excessive complexity

- Belief inherent in early Java EE specifications that complexity didn't matter as it would be hidden by tools
- CMP entity beans
  - Mammoth deployment descriptors
- JSF

The Microsoft delusion

- This was partly a response to Microsoft's success with Visual Studio
- Failed because
  - MSFT understands tools in a far deeper way than anyone in the Java community until recently
  - Fragmentation of IDE market meant that until the rise and rise of Eclipse there wasn't enough critical mass to create a Visual Studio challenger

The appropriate role of tools

- I am not someone who works in emacs or vi (any more)
- I’m not suggesting that tools are bad
  - …just that we shouldn’t use them to sweep complexity under the carpet, as it will still cost us in the end

Lack of openness to other ideas

- MSFT influence on tooling was an anomaly
- Java specifications have too often ignored prior art
  - TopLink and real ORM (CMP)
  - java.util.logging
  - EJB 3.0
  - Ongoing reinvention of AOP without any real domain experts involved
- May be improving, but has long been a challenge

Failed reinvention in the JCP

JCP technology	Ignored existing technology	Negative consequences
Entity beans	TopLink and all other ORM solutions	•Two complete failures (EJB 1.x and 2.x) •ORM in Java loses at least 6 years •Billions of dollars of wasted development effort from customers
Commons Logging	Log4J	Added complexity of pointless abstraction layers such as Commons Logging
EJB (DI)	Spring, PicoContainer, Hivemind	Limited DI functionality in EJB 3 specification misses opportunity to match best practice
EJB3 (interception)	Spring, AOP Alliance, AspectJ, AspectWerkz	Lack of knowledge of AOP in the expert group produces fragile, clunky API missing central AOP concepts
JSR 277 (modularization)	OSGi	•Ignoring input and experience from OSGi •May split JCP as many organizations are deeply committed to OSGi

Trap: The Myth of the Code Monkey

- Belief that developers are dumb
- Primary goal is to prevent them making decisions
- Sadly, far from unique to Java

Problems due to the Myth of the Code Monkey

- Belief that persists in the EJB specification that developers are incapable of using language-level concurrency features
- Aim of *many* old in-house frameworks to provide a straightjacket for developers

Primate Programming Inc.

- *Humans and higher primates share approximately 97% of their DNA in common. Recent research in primate programming suggests computing is a task that most higher primates can easily perform. Visual Basic 6.0™ was the preferred IDE for the majority of experiment primate subjects."*

Primate Programming Inc.

- *Great apes (hominids) do not have tails, while monkeys do. Research indicates that great apes are very productive in the areas of software maintenance and report writing, while most monkeys will struggle. Monkeys however are great at software testing. So the rule of thumb is, if you don't have a tail, you can probably program.*

Myth: Complexity is good for you

- Natural tendency to believe in the old adage, *no pain no gain*
- Leads to an Emperor's New Clothes syndrome
  - EJB most obvious offender
- Causes people to *accept* the complexity resulting from the other factors

Myth: This needs to fit into our over-arching SOA/XYZ strategy

- A variant of the love of complexity
- Means that common sense is thrown out the window
- SOA is probably the most obvious case
  - Complex term for what people already do
  - Enables selling products
  - Karl Marx, *alienation*

HOW DID THINGS BEGIN TO CHANGE?

It’s the economy, stupid

Money to spend gets spent

- Economic exuberance
- Who cares about efficiency?

Economic downturns tend to reduce complexity

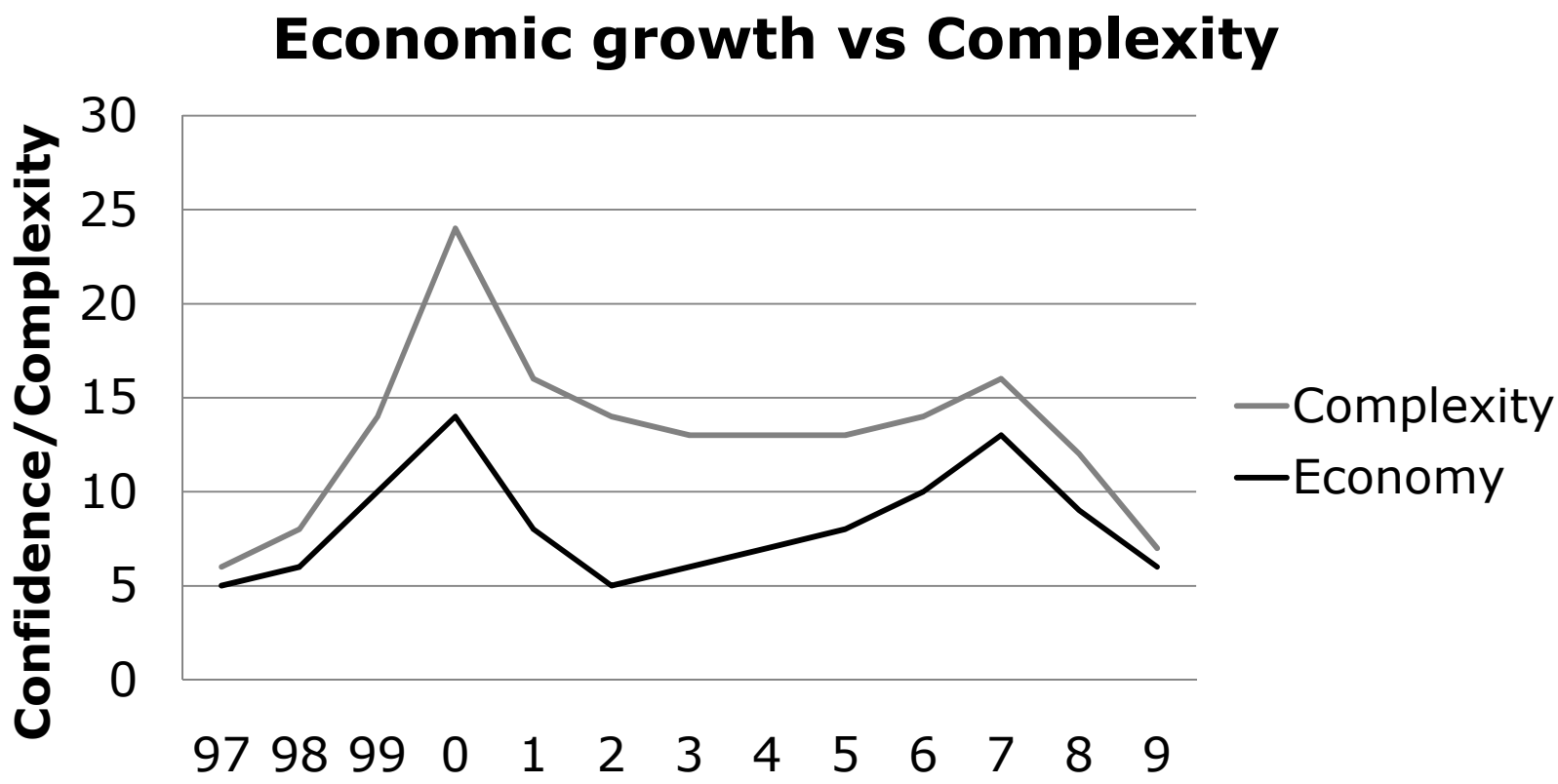
- Complexity is an expensive luxury

- License cost of complex proprietary products is just one factor
- Ongoing complexity throughout the software lifecycle is even more problematic

Hemline Theory

- American economist George Taylor (1926)
- "Hemline Index"
- Theory that lower hemlines mean a falling market
- Market confidence, availability of money changes peoples' behaviour

A short history of enterprise Java complexity



Developer empowerment: A key part of the solution

- We've seen more and more development empowerment
- Why?
  - Because the previous failure was so severe
    - People got promoted for adopting open source
  - Because things change quicker than in the past, and management need to rely on developers more

Developers have enormous power today

- We live in great times for developers
- Ability to make a difference in a deep way
  - Much choice developers can make
    - Compare with the mainframe days
  - Open source allows participation

Innovation from the community has made decisive change

- Spring
- Hibernate
- Ruby on Rails
- Django
- Grails

Rise of agile ways of working

- Another developer-led initiative
- Helped to expose the flaws of the original J2EE programming model
  - Untestable code
  - Slow test cycles
  - Lack of immediate feedback

WHAT HAPPENS FROM HERE?

What happens from here?

- Unlikely we will go backwards
- Developer empowerment is not going to change
- Java productivity needs to improve further due to external challengers
- Complex portfolio solutions out of favor
  - Push toward effective point solutions
- Likely to see the traditional application server fade away

Cloud: The coming tyrannny of the developer

- Long-term trend is going to further strengthen power of developers at the expense of operations

Escaping the traps?

Yesterday's Traps

- Design by committee is a good idea
  - Everything has to be standardized
- Tools can make excessive complexity acceptable
- Lack of openness to other platforms, sources of ideas
- Developers are stupid and must be controlled
- Complex solutions are better
- "I need a solution that enables my enterprise SOA/XYZ strategy"

Today

- Innovation that gets adopted comes largely from open source
- Solutions are simpler, even as tools are better
- Responsiveness to other platforms (Grails etc.)
- Successful frameworks treat developers with respect (Spring)
- Shift away from WebSphere & co to simpler solutions
- SOA buzzword is less hyped

Tomcat adoption shows a profound shift in the market

- By far the most popular application server today, in development and production
- Used by around 70% of organizations developing Java web applications
- Represents a developer-driven switch away from complexity

Summary

- Java is in a far healthier state than for much of its history
- Developer empowerment is the dominant trend of the last few years
- Present economic troubles have at least some positive results in technology

Call to action

- The key person who can make a difference is *you*
- Developers have brains and should think for themselves
- Developers are highly capable of seeing through flawed solutions
  - Should not accept the Emperor's New Clothes again

Q&A