Java is a trademark of Sun Microsystems, Inc.

JavaOne℠

Rethinking the ESB: lessons learned challenging the limitations and pitfalls

Andreas Egloff
Frank Kieviet
Sun Microsystems

OpenESB
The *Open* Enterprise Service Bus

Agenda

- Background: from what did we learn?
- Three lessons we learned:
  - How to be lighter weight
  - The importance of architecture
  - How to avoid vendor lock-in
- Demo
- Conclusion

From what did we learn?

- SeeBeyond: integration middleware vendor
  - 1992: DataGate
  - 1999: eGate
  - 2003: ICAN / Java CAPS
  - 2008: OpenESB / GlassFish™ ESB v2
  - 2009: OpenESB v3
- 1994: Forte
- Now: Sun Microsystems

Feedback is shaping every new release

- Changing environments and requirements
- Measurements, research
- Positive feedback from customers, analysts
- Negative feedback from “thought leaders”, e.g.
  - “Integration middleware is...
    - big, heavy, expensive,
    - difficult to install,
    - difficult to set up,
    - complicated to learn and use,
    - a haven for lock-in”

ESBs pros and cons

- ESBs *as a product category* indeed come with risks, e.g.
  - Architecture concerns, good or bad for SOA?
  - More middleware
  - Concern of lock-in, over-reliance on one provider
- Properties of individual ESBs or course vary
- ESBs do serve a purpose:
  - Productivity increase
  - Message based integration style, decoupling
  - Service intermediary
  - Not everything is a web service

Java is a trademark of Sun Microsystems, Inc.

JavaOne℠

Lessons learned

OpenESB
The *Open* Enterprise Service Bus

What do *we* call light weight?

- Light weight framework
  - Quick to learn, not complex, good productivity
  - Modular, embeddable
- Light weight solutions
  - Easy to evolve
  - Easy to manage
- Very little to do with installation size!
- … but what are the *requirements*?

Lightweight: non functional requirements

- Enterprise scale integration
  - “Throwing bodies at the problem”
  - Scalable design time, runtime
- Change happens!
  - Decouple configuration, deployables from design time
  - Faster change loop
  - Patch management
- Long term commitments
  - Today's hot stuff is tomorrow's legacy

Accommodate large teams

Shorten the learning curve

- Observation: Enterprises like to scale by adding people
  - Teams have mixed skills
  - People flow in and out of teams
  - Lesson: easy-to-learn is critical
    - Provide tooling
    - Keep things simple!

Easy to learn...

… but don't let tooling get in the way

- Tooling helps with “hiding” complexities
  - Observation: total reliance on tooling is problematic
  - Lesson: also provide file access

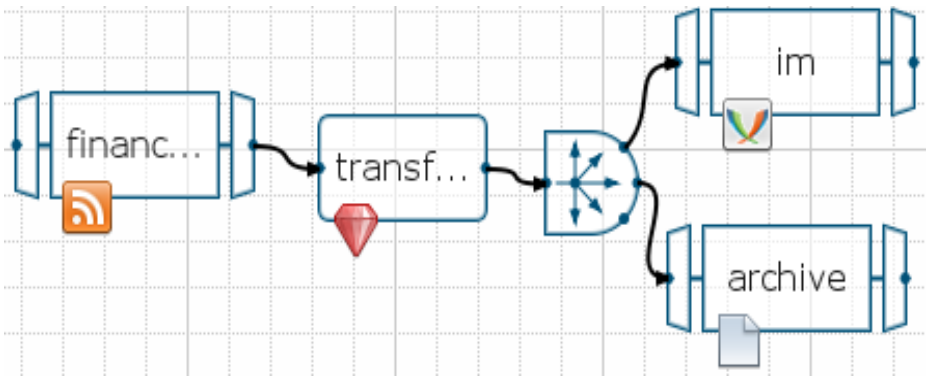
- Graphical tools are perceived as user-friendly
  - Observation: graphical tools difficult to scale
  - Lesson: also provide textual access
  - Lesson: Provide options for different roles, preferences

Increase developer productivity

- Download, installation, configuration
  - Observation: An expense that is paid multiple times
  - Lesson: reduce download to up 'n running time
    - Simpler installer with less interaction required
    - Add-ons can be installed via Java Web Start
    - Command line install option
    - Extreme: unzip and start

Increase developer productivity

- Observation: tendency to duplicate code
- Lesson: reduce coding
  - Prefer convention and configuration over coding
    - In that order!
  - Support patterns (EIP) as first class citizens




```
rss "finance-feed"
jruby "tranform"
file "archive"

route do
    from "finance-feed"
    to "tranform"
    broadcast do
        route to "archive"
        route to "notify"
    end
end
```

Don't let frameworks constrain

- Observation: frameworks are powerful to reduce code duplication

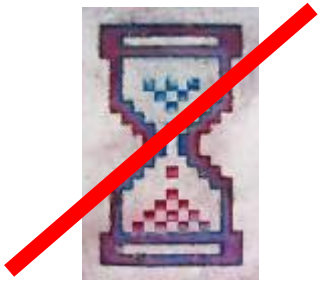
  - But sometimes constrain flexibility
  - Product is always used differently than expected
  - Users want flexibility in changing behaviors
- Lessons:
  - Interceptors a powerful way to add flexibility
  - Work with YOUR technologies
    - Choice of frameworks, languages, tooling

Increase developer productivity

- Observation: developers spend a lot of time *waiting*
  - Lesson: shorten the change-build-deploy-test cycle
    - Reduce build time, deploy time: smaller deployables
    - Extreme: in-place editing
  - Lesson: increase testability, debuggability
    - Tests built in
  - Lesson: support continuous integration

Modularity

- Observation:
  - Modularity improves startup time, performance by only loading what's necessary
  - Modularity leads to faster product development
  - Being embeddable opens more options
  - Modularity can help control complexity
- Lesson:
  - JBI
  - OSGi
  - Extensible language, UI



Scale the design time

- Observation: explosion of project artifacts

  - Lesson: Reduce artifacts per solution
  - Lesson: or make managing large numbers of artifacts possible
    - e.g. hierarchical ordering of artifacts

Reduce deployment size

- Observation: gigantic EAR files overwhelm runtime
  - Lesson: reduce the size of deployables
    - Shared components
    - Shared libraries
    - JBI engines / components

Connectivity configuration

- Deployables are used in different "environments"
  - Development, testing, staging, production
- Different roles involved
  - Developers, testers, administrators
  - Observation: people in different roles use different tool sets
- Lesson: decouple configuration from design time and tie to runtime
  - Tools support the workflow

Quicken business logic changes

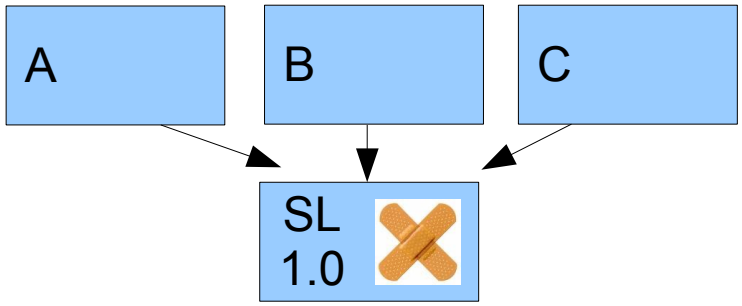
- Observation: message based services more robust in the face of change
- Observation: sometimes tiny changes in business logic are required in production systems
  - Would have to rebuild the application + release cycle
- Lesson: avoid rebuilding “the world” for small business logic changes
  - Use scripting languages
  - Externalize business logic (e.g. CBR)
  - Consider evolution, versions in handling messages
  - Externalize cross-cutting concerns (e.g. Aspects)

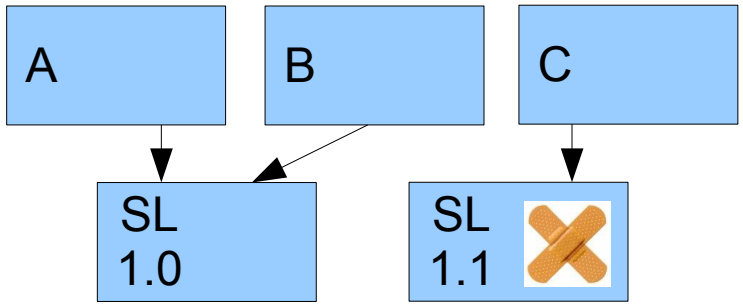
Patch management

- Patch management is important
  - Keeping track of what patches are applied
  - Patch distribution to reach all affected systems
- Observation: complex patch management interferes with minimal downtime
- Lesson: keep patching simple

The shared library problem

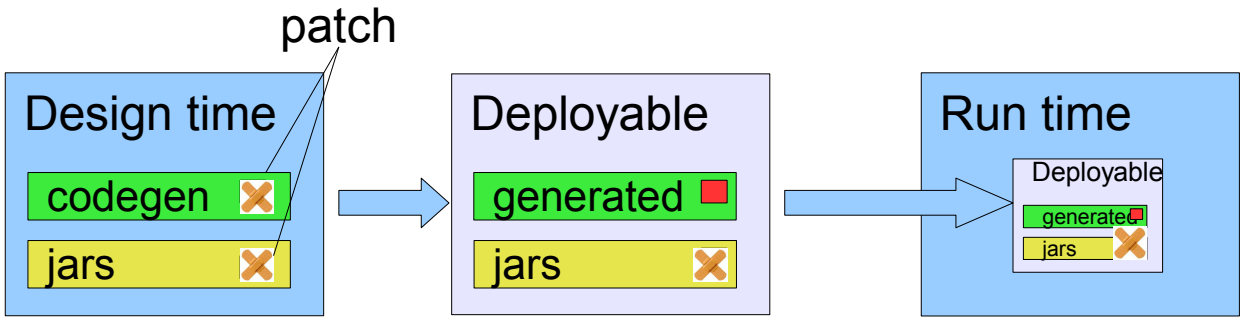
- Shared libraries keep size in check, allows simple patch-all
- Observation: patch-all too limiting
- Lesson: OSGi provides solution





Reproducible builds

- "The build" needs to be controlled, reproducible
- Observation: if design time generates code, or adds jars to deployable, *whole design time* needs to be in VCS
- Lesson: avoid code generation, move jars to runtime
- Lesson: small command line build environment



Middleware: long term commitment

- Observation: systems stay in production for many years.
- Observation: languages, frameworks, etc hot today will be like disco 10 years from now.
- Lesson:
  - Reduce reliance on IDE
  - Use mainstream languages, frameworks, etc.
  - Look for “staying power”

Architecture

- Two more architectural lessons-learned:
  - Component communication
  - Machine-machine communication

Efficient component communication

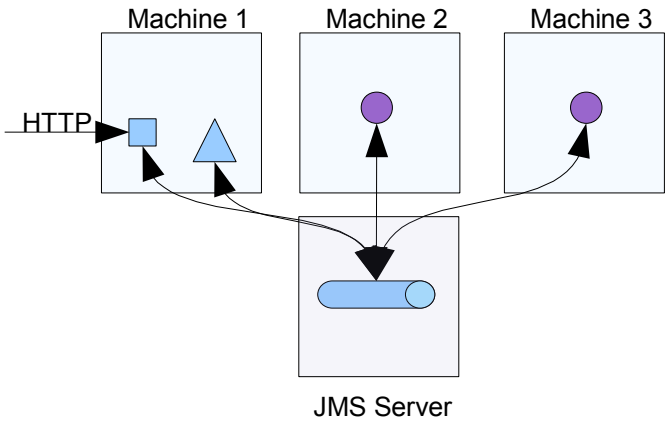
- Observation: JMS is often a bottleneck
- Lesson: component-component message-passing can also be done without JMS
- In-memory message exchange allows
  - Performance improvement
  - Message streaming, pass by reference
  - Propagation of security & transaction context
  - Easy throttling

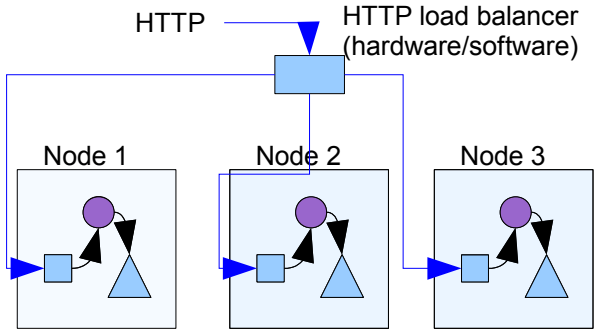
Machine-machine communication

- Observation: JMS has a proprietary wire protocol, requires yet more infrastructure
- Observation: HTTP infrastructure provides proven scalability and interoperability
- Lesson: JMS is just another transport option
- Service platform with ESB features vs. MOM-focused
  - Better, simpler scalability
  - No impedance mismatch to SOA
  - Choice of protocols scales outside the enterprise

Choose your topology

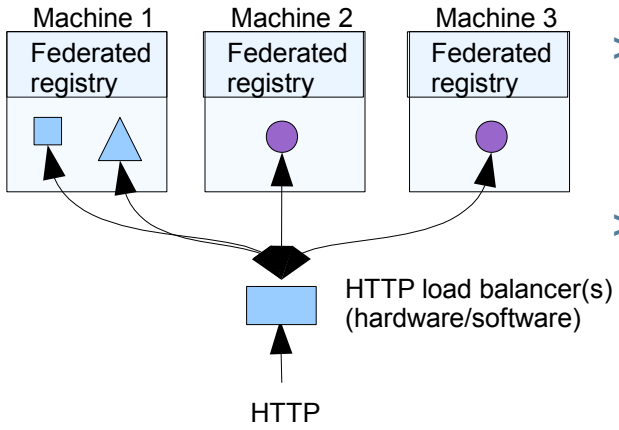
Scale easily with homogenous clustering

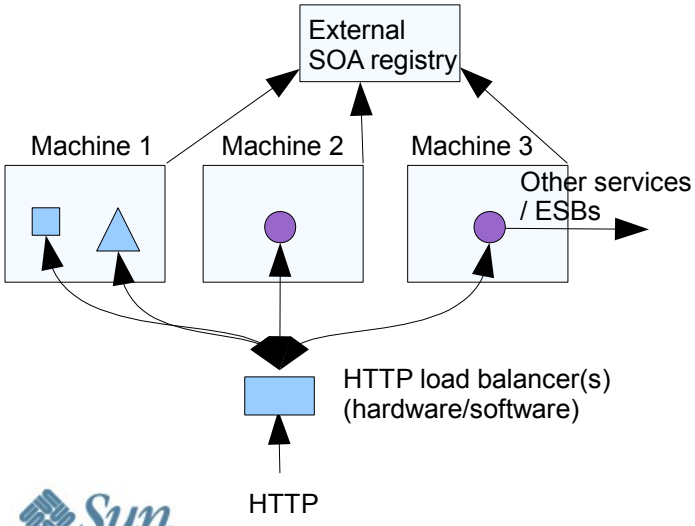




- Observation: heterogeneous clustering based on JMS is the tradition
- Lesson: homogeneous clustering also works and is easier to manage
  - Push load balancing out
  - Protocols:
    - JMS is an obvious choice, but:
    - HTTP / web services make for better interoperability, efficiency

Choose your topology
Simpler heterogeneous clustering



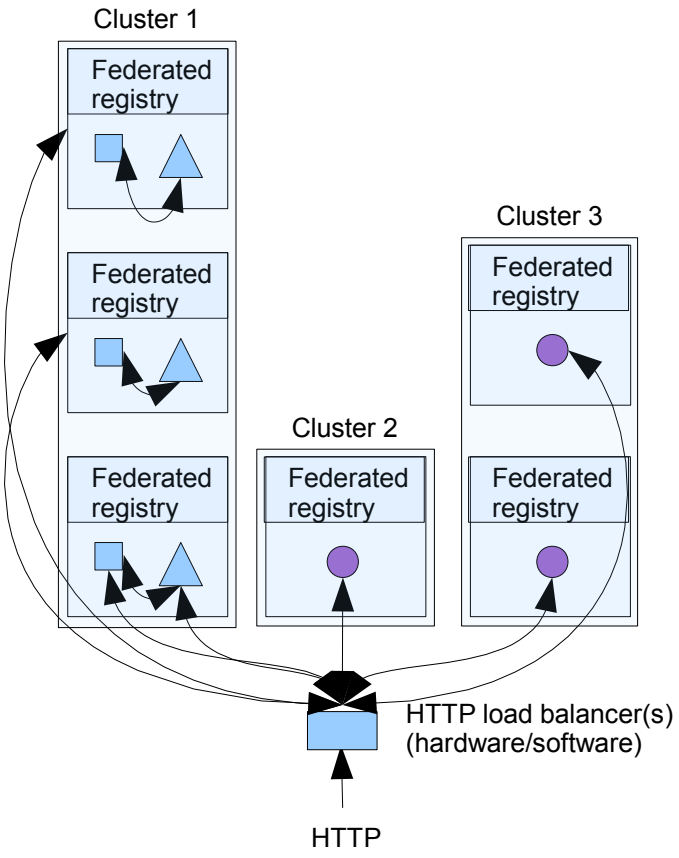


- Observation: Not everything will be homogenous
- Lesson: leverage SOA principles and standard protocols for heterogeneous topologies too
  - ESB can provide
    - location transparency for services
    - explicit calls and use of external (SOA) registries
  - Flexibility of protocol
    - JXTA, HTTP, JMS...

Choose your topology

Combined topologies

> Lesson: homogeneous and heterogeneous clustering complement each other



- Scaling, high availability
- Flexibility for varying requirements
  - Data or system locality
  - Differing scaling, HA requirements
- ESB features with choice of topology, transport
- In-memory when appropriate

Avoid vendor lock-in

It's good business!

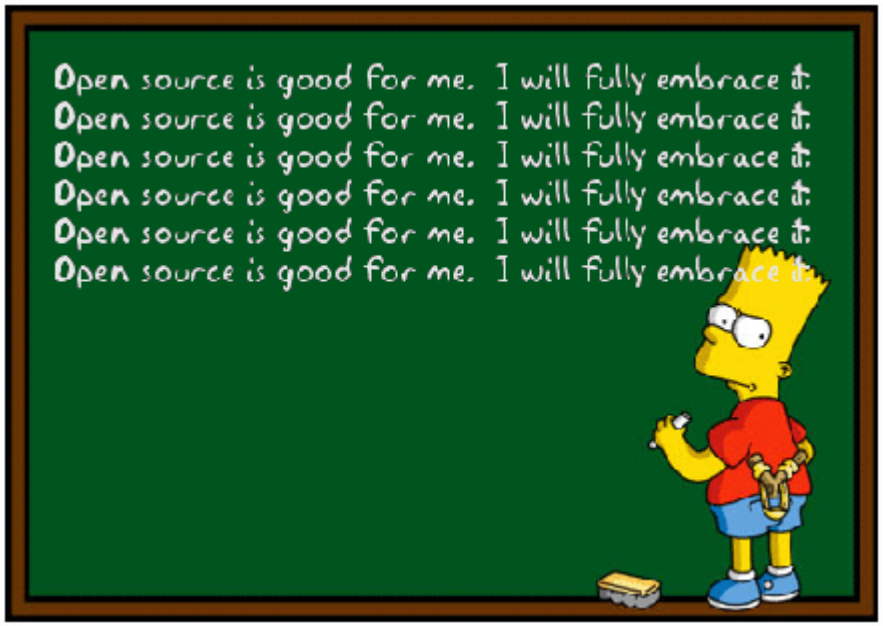
- Observation: Customers demand it
- Lessons: Change mentality... “come for our lack of vendor lock-in, stay for the product qualities”
  - Stick with open standards
  - Open source
  - Encourage best-of-breed
  - Provide an option to exit

Open standards

- Main concerns:
  - Interoperability
    - JBI
    - Standard Protocols, web services: communicate with other products
  - Portability of skills
    - e.g. Java, EE, Spring, BPEL

Open source

- Observations:
  - Customers increasingly demand it
  - Makes vendor lock in more difficult
- Lessons learned:
  - Allows better interaction with customers, faster turn-around
  - Leads to better products



Be open to “best of breed”

- Observation: the everything-from-one-vendor approach is very limiting
- Lessons: Allow customers to use other parts from other vendors: mix and match best of breed
- Standards and pluggability make this easier

Provide an option to exit

- Observation: Customers increasingly demand it
- Lesson: Enable users to move project artifacts to another product
- Lesson: Interoperate easily with other frameworks
  - Service based will continue to work as part of common SOA approaches
  - Open Standards ensures the existing system can work with new systems
  - Open Source
    - Ensures continued ability to fix a legacy system

Java is a trademark of Sun Microsystems, Inc.

JavaOne℠

What does it look like?
(Demo)

OpenESB
The *Open* Enterprise Service Bus

Conclusion

- Lessons learned:
  - Be light weight
  - Better architecture: better product
    - Not just rebranded MOM
  - Avoid vendor lock in
- Lessons learned are *requirements*
- Look beyond feature lists to systemic qualities
  - How will it fare over time in the real world
  - Choose the right topology and distribution model

JavaOne℠

Thank You

Andreas.Egloff@sun.com
Frank.Kieviet@sun.com
http://open-esb.org

Sun microsystems