Business Activity Monitoring and SOA Environments

A Case Study

Frank Greco
fgreco@javasig.com

Topics

- SOA Environments
- Business Activity Monitoring (BAM) and Operational Business Intelligence (BI)
- Case Study – Lehman Brothers Fixed Income
  - History and Environment (circa 2005)
  - Management Requirements and KPI
  - Architecture and Implementation
  - Tuple Spaces, Interfaces, Abstractions
  - Querying Business Activity in Real-time
- Extending over the Web
- Lessons Learned
Service-based Environments

- Build Applications as a set of Collaborating Components
- Applications live on Different Servers, Different Technologies, Different Platforms
- Improves Business Agility with Flexibility
- Distributed Nature is a Strength
- Distributed Nature makes it Harder to Manage
- Optimally, Needs to be Self-Managed
  - Maybe one day…
  - Until then, we rely on monitoring

BI and BAM

- BAM needs to be first-class citizen of the SOA, not an after-thought
- A Service equally important to other services
- Part of initial architecture
- KPI monitoring
  - Operational BI
  - SOA monitoring
  - Dynamic query of SOA infrastructure
Lehman Brothers – Case Study

- Global Fixed Income Division – circa 2005
  - US, UK, India, Japan
- Traditionally Service-oriented with Large Grids
- Mostly Java, some C++/C
- ESB based on Tibco RV/EMS and JMS
- Multiple Services, Large and Small
- Needed Central Messaging System Solution
- Senior Management wanted to monitor KPI
  - Multiple websites
  - Multiple subsystems
  - Multiple interfaces

Diagram:

- Java App
- .NET App
- C++ Mortgages
- Java Govt
- Java Credit
- Etc…
Lehman Brothers – Case Study

Java App

.NET App

XML protocol

C++
Mortgages

Java
Govt

Java
Credit

Etc…

LB FI - Key Performance Indicators

- Avg/Max/Min service response time
- Service availability (up/down)
- Status of service request (sent/received/pending-in-queue/calculating/et al)
- Average service-request queue length
- History of service requests
- SLA history (color-coded) per service
- Percentage of network/cpu/other-resources in service-request/response
LB FI - Key Performance Indicators

- Tibco RV messages per second
- Tibco JMS messages per second
- Market data availability from all sources
- Trading system cache status
- Which grid has highest utilization
- Which grid has lowest queue length
- SLA performance by customer
- Java Exceptions per Service
- Et al…

But each service had its own monitoring!

Solution? Evaluate Vendors First

- Current Vendors did not have Global solution
- UI’s were very primitive
- No Vendor APIs available
  - But JMX was available
- Several Monitoring Vendors
- No Viable Management Vendors

- If Can’t Buy, Build…
- If possible, hand off built system to vendor
Java Management eXtensions

- Standard Java Technology
- Provides a simple, standard way of managing resources such as applications, devices, and services

JMX Issues

- Need to query a JVM for resource status
- Need to aggregate information
- Integrating C++ applications
- Persisting monitoring data for historical analysis
- Geographically distributed monitoring
- Many users monitoring same systems potentially affecting performance
- Stand-alone UI, Web UI and API
- How to handle versioning of monitoring subsystem
- Etc…
Tuple spaces

- Why Tuple Spaces?
- Distributed Computing Models
  - Point to Point (rpc, sockets)
  - Client/Server (jdbc, app servers, et al)
  - Publish/Subscribe (jms, tibco rv)
  - Master/Worker (traditional grid)
  - Peer-to-Peer (agents)
  - Others...

Models of Distributed Computing

- Point to Point
- Client/Server
- Publish/Subscribe
Models of Distributed Computing

- Master/Worker
- Peer to Peer
  - agent-based

Issues with Distributed Computing

- Service Locator
- Partial Failure
- Network Latency
- Concurrency
- Transactional Integrity
- Security
- Heterogeneity
- Scalability (Growth/Complexity)
**Better Approach: “tuple spaces”**

- “JavaSpaces” – Java version
- Distributed, Transactional Shared Memory
  - Memory is Cheap, Networks are Very Fast++
- Works by Matching Objects, Pull-Model
- Natively Peer to Peer
- Can Implement other Distributed Models
- Derived from Linda Project at Yale University (1983)
  - David Gelernter
  - java.sun.com/developer/technicalArticles/Interviews/gelernter_qa.html

**JavaSpaces**

- Expressive with Small API Set
  - write()
  - read(), take()
  - notify()
- Asynchronous and Synchronous
- Associated, Class-oriented Lookup
- Handles Concurrency, “Multi-thread Hot”
- Transactional
- Objects are “leased”
- Inherently Loosely-coupled
Spaces - Services “Glue”

JavaSpaces Programming Model

- Service Providers register with the space by creating an object of an agreed-upon class and fills in fields with desired matches
- Service Requesters create an object of an agreed-upon class and send it to the Space
- If SP's has a (polymorphic) match, the SP can take the object (or just read it) from the space
Native Communication Mechanism

Cooperating Spaces for High Availability
Spaces – Highly Available

Cluster of Shared Memories

Replicated Clustered Name Spaces
Security per Space
Event-based

Compute Close to Data as Possible

Features of Spaces

- Simple
- Shared Data
- Handles Concurrency
- “Leased” Objects
- Exchange Executable Content
- Exchange Executable Content
- Associative, Class-oriented Lookup
- Transactional
- Expressive w/ Small Set of Primitives
- Inherently Loosely Coupled
Tuple spaces

- Now we need to select JavaSpaces vendor…
  - Sun Reggie
  - Blitz
  - GigaSpaces

- GigaSpaces was selected
  - Speed (tests over Infiniband/RDMA were impressive)
  - Plug-in architecture
  - Can handle Java, C/C++ and .NET objects

JMX Repository

- Match JMX Events with Spaces Reliable Delivery
- JMX Events saved via Spaces-based repository
  - Can be Distributed Geographically
  - Replicated for Robustness and Performance
  - JMX Containers with Spaces
- To Scale, Just Add More Space Servers for Increased Monitoring Loads
Geographic JMX Monitorability

Interfaces

- Swing UI
  - Users could monitor any ZEUS-enabled subsystem
  - Plug-in architecture
- Web UI
  - Static snapshots
  - Viewable from standalone browser or Swing-Browser (WebRenderer.com)
  - Zero-install situations
- API
  - Developers wanted access for their applications
  - Needed specific GUI
  - CLI-based monitoring
Capturing Real-time BI

- Javaspaces allowed real-time monitoring of Java objects (and C++)
- Persistence layer within GigaSpaces allowed historical analysis
- Log files were kept (Splunk was investigated)
- Partnered with Skytide (skytide.com) to generate BI databases dynamically
- Generated OLAP cubes dynamically that can be queried in real-time

Real-time Web UI

- Java Swing UI was very popular
- Management also wanted Web 2.0 interface
- Kaazing.com
  - Provides socket interface between browser and server
  - Fully compliant with WebSocket and Server-Send Event APIs from HTML 5.0 spec
  - No more page refreshes or timer refresh
  - More efficient than Comet-style long-polling
Kaazing

Full Socket Connectivity via Browser (http)

Lessons Learned

- FI SOA needed to include Monitoring from the beginning
- KPI’s need to be identified at the start
- Be Prepared for unexpected applications
  - Our transport layer was used outside of monitoring
  - Users asked for non-trivial CEP engine
- Handling versioning in a large SOA is difficult
  - Javaspaces made it easier though…
- Use of Choreography Description Language (CDL) for all services would guarantee SOA workflow
Current Monitoring Tools/Vendors

Oracle

EvidentSoftware.com
Current Monitoring Tools/Vendors

BAM and SOA

Questions?