The Investment Banking Technology Stack

By

John Davies

CTO, C24

Bienvenido a Barcelona!

- **Welcome to Barcelona**
  - *Benvenuto, Willkommen & Bienvenue*

- ***Agenda***
  - What do we do in Banks?
  - Technologies we use and how they fit
  - Technology for the future
    - tomorrow onwards…
  - *Warning – this is non-technical, full of buzzwords but thankfully sales-free (almost)*

Banks handle cheques - right?

- **Yes but they do a lot more…**
- **There are two main types of banks**
  - Retail and Wholesale
  - Retail handle your cheques
    - Internet banking, ATMs, High St. banks etc.
- **Wholesale or Investment Banking is where we deal with businesses rather than individuals, in general**
  - Goldman Sachs, Lehman Bros., Merrill Lynch, JP Morgan Chase, Citigroup, Bear Stearns, Deutsche Bank, Dresdner KW, BNP Paribas, UBS, Credit Swiss, West LB, RBS, RBC, ABN Ambro, Rabo Bank, SocGen, Credit Agricol, Daiwa, NAB
  - There are well over hundred of them

Banks can be split into 3 parts

- **Front Office**
  - This is where the dealing rooms are
  - Traders buy and sell currencies, metals, shares, bonds - basically anything
  - Typically high volume, low latency, medium value messages
- **Middle Office**
  - This is where the accounting is done
  - Trade matching, reconciliation, risk, P&L, limit keeping
  - Typically medium volume/latency, extreme calculations
- **Back office**
  - This is where the payments are settled (paid)
  - Typically low volume, very high value messages ($billions)

Front Office

- **Traders deal over the phone for the bank's clients (brokers, businesses, rich people)**
  - They want to know what they have to sell and what they can buy (sub second).
  - FIX is the most common messaging format here

- **Automated trading systems use rules to trade on the exchanges**
  - Volumes often exceed 10,000 messages a second
  - Reaction times are typically in the low milliseconds

- **There are several exchanges and many clients**
  - Each with different prices and risk profiles

Middle Office

- **Mostly accounting**
  - Not just adding up numbers but matching trades and reconciling balances
  - Maintaining positions in real-time, how much or each stock do we have today, how much will he have tomorrow
- **Monitoring and calculating risk in real-time**
  - Usually very complex calculation
    - Monte Carlo simulations, Value at Risk (VaR)
  - Fraud detection, Anti-Money Laundering (AML)
- **Trades can get very complex – FpML widely used here**
  - FpML is XML but very complex, it breaks most XML tools
  - FpML also has several dozen rules outside of the schema

Back Office

- **Typically settlement (payment)**
  - Very large sums of Money
  - Often a day's trading with one entity all rolled into one payment
- **Messaging is usually S.W.I.F.T.**
  - Society for Worldwide Interbank Financial Telecommunication
- **SWIFT messages are nasty**

```
{1:F01MIDLGB22AXXX0548034693}{2:I103BKTRUS33XBRDN3}{3:{108:MT103}}{4:
:20:8861198-0706
:23B:CRED
:32A:000612USD5443,9
```

- **Parsing is one thing but if the message is mal-formed the bank gets fined**
  - This is where C24 makes most of business
  - C24 maintains object models for SWIFT, FpML and other standards, it can produce Java objects (POJOs) from the models

Real-time

- **Real-time in banking == "faster than everyone else"**
- **If your competitor can trade in 5ms you** need **better than 5ms, any less and you lose the deal**
  - We build trading engines physically close to the exchange to reduce the speed of light time to the exchange
- **Before you can trade you need to know…**
  - How much you currently have
  - How much you want or don't want
  - How much you have to spend
  - What your price range is
  - What the risk is between now and the next 'n' years
- **Do all this in under 5ms at >10k/sec – it's fun!**

High Performance Computing

- **Given the speed we're working in we don't have time to use classic databases**
- **Information is held in memory**
  - This is usually segmented and distributed
- **Grid or HPC technologies like DataSynapse, Platform, GigaSpaces, Gemstone and Tangosol are used to provide distributed objects pools**
- **Most large banks have 2-5,000+ CPU grids**
  - Often a mixture of blades, Egenera, V40s etc.
  - recently Azul has appeared with interesting technology

Jini & JavaSpaces

- **For some reason West Coast Sun seem to hate anything in Sun that isn't JEE**
  - JavaSpaces was a better server-side container before J2EE - It is still better
  - Jini provides an excellent discovery mechanism – used with JMX

- **The Master-Worker pattern works perfectly in JavaSpaces**
  - They were made for each other

- **JavaSpaces are playing a big part in Financial HPC**

- **JavaSpaces is not a cache, Coherence is a cache**
  - I will be speaking about this in more detail on Friday

Hardware

- **We often benchmark machines, the CPUs and OSs**
  - Very few banks use MS based servers other than to service desktops
  - There is a preference for Linux but other UNIX-based OSs are popular, particularly Solaris
- **It's not just the speed but the heat and power**
  - Imagine power and cooling for 5 rows of 8 racks of 64 blades with 2 CPUs each
  - Internal charges for managing hardware can amount to $10-20k per box per year over a minimum of 3 years
- **With one OS across the board we can virtualise the others if we need them**
  - We use VM-Ware and Xen
  - Vendors who somehow slip through with MS based applications can be hosted in VM-Ware sessions

JVMs

- **A lot of development is now on JSE 5 (version 1.5)**
  - Some are still stuck with 1.3 because they use WebSphere
  - Many banks however still prefer J2SE 1.4 in production
  - JSE 6 is only used by the innovators – for now

- **JSE 5 gave a notable performance gain over 1.4**
  - The added JMX was excellent for monitoring and debugging
  - We justified the use of JSE 5 on performance alone

- **We restricted many of the new features in 1.5**
  - New features are great but when you have several 1000 programmers it's difficult to educate them all at once
  - We provide guidelines, what to use, what not to use
  - Remember it was the performance we liked not the features

Messaging

- **In the front office we find Tibco Rendez Vous (RV)**
  - This is a high speed Pub/Sub system
- **The rest of the bank tend to use IBM's MQ Series**
  - Rarely because they like it, mostly because it works on everything
- **Most Java applications abstract the messaging behind the JMS**
  - This isn't so easy with RV, it has more features and the majority of clients are MS based (desktops)
- **New innovations like AMQ will change the dominance of MQ and RV over the coming years**
  - More on AMQ in a few minutes

Application Servers

- **J2EE application Servers have been dying for some years now**
  - We still use Application Servers but you often need to justify their usage
  - A high proportion of App Servers are third-party application
- **We still like the Java stack, just not the restrictions of J2EE**

- **Servlet containers are very popular, Struts, Servlets, JSPs, Tiles etc.**
  - Tomcat is all over the place

- **JPMC created an internal stack**
  - Basically Maven, Tomcat, Hibernate, Spring etc.

Application Serves – Open Source

- **We need flexibility on containers**
- **Applications must scale up AND DOWN**
- **Spring has become a very popular framework to work with**
  - It has created plenty of its own problems but it’s the next generation of framework after J2EE
  - BEA did well to recognise this
- **Spring gives us flexibility beyond J2EE into the realms of grid – by wrapping JavaSpaces**

- **Banks are very open to Open Source**
  - In fact it is actively encouraged
  - Some examples of Open Source coming from banks to follow...

Open Source – Is Safe

- **If the supplier is**
  - Too small we worry about support and longevity
  - Too big we worry about innovation and unwillingness to adapt
  - "just right" we worry about them being bought
- **Software, domain knowledge and skills are what we look for in a supplier**
  - Price comes into it but it's less of an issue that you might think
- **Open source offers a nice way to protect our software purchases**
  - It avoids escrow agreements
- **Having open source protects against crap support, speed of innovation and someone buying the company**

Open Source & Standards

- **Just making something open source doesn't mean it's instantly attractive**

- **It should be part of a standard**
  - Remember, one implementation is proprietary, open or not
  - Other implementations can be commercial products or open

- **If the source code is available we can...**
  - Check the quality
  - Narrow down bugs and issues, sometimes fixing them ourselves
  - Improve our understanding of the product
  - Better optimise code/integration etc. for the product
  - Be sure there's nothing nasty in it

Open Source - Reality

- **We rarely actually use the source code**
  - The real advantage of open source is the protection it offers
  - From a sample of several dozen projects using Tomcat less than 20% had downloaded the source code
    - Non of those had compiled it
- **In reality very few people actually commit to open source projects**
  - The majority actually work for the open source company/project
  - Well over 90% of Spring is written by Interface 21, this is good
- **The open source community is therefore not the committers but the users**
  - Users provide support an documentation, via forums and wikis

Open Source - Politics

- **Open Source is full of politics – too much**
  - You're either JBoss or Apache
  - Axis or XFire
  - Something from James Strachan – or something finished ☺
  - Mule or Celtix
  - Tomcat, Jetty or Resin etc.
- **And then there's the licenses!**
  - Code is code whether is GPL, LGPL, BSD, MPL or MIT etc.
- **Many companies naively release code to open source as a token of good will without realising the politics of it all**

Open Source from the Banks

- **OpenAdaptor (http://www.openadaptor.org)**
  - Started by Dresdner Bank in late 90s, released in 2001
  - Provides adaptor framework for integration
  - Used by several banks, currently beta 3.0
- **Spring (http://www.springframework.org)**
  - Rod might disagree with the origin but he spends most of his life working in large financial institutions in London
- **AMQP (http://infoq.com/news/amq)**
  - Announced yesterday at a press conference at JPMorgan
  - A wire-level asynchronous protocol to provide interoperability at the wire level (accessed in Java through JMS)

AMQP

- **AMQP – Advanced Messaging Queue Protocol**
  - AMQP is a specification for queue-based messaging that is broadly applicable for enterprise use, is technology agnostic, totally open, and completely interoperable
  - AMQP defines a protocol and model for queue-based messaging
  - AMQP fills the "gap" in current web service and messaging specifications for asynchronous messaging
- **Why AMQ?**
  - There is a large gap in the networking protocol standards
  - No open messaging standard JMS offers a partial solution, but is limited by its reliance on Java and by not specifying any wire-level interoperability
  - AMQP can be used to provide interoperability to the JMS API Any solution has to be an open work with the ability to be used from any platform and any language

Who's behind AMQ?

- **JPMorgan Chase**
  - "Invented" by **John O'Hara,** VP, Distinguished Engineer and Senior Architect
- **Cisco**
- **Envoy Technologies Inc.**
- **iMatix Corporation**
  - Played a major role in the protocol specification and early implementations
- **IONA Technologies**
  - Proving products and support around AMQP
- **Red Hat, Inc.**
  - Will supply an AMQ implementation with their distro
- **TWIST**
  - Will be the transport for the Twist network
- **29West**
- **Several others – can't be named**

Databases

- **Legacy**
  - Commoditised
- **With today’s caching technologies e.g. Tangosol, Gemstone and GigaSpaces why have a database?**
  - We just need them for archival
- **We have enough memory to run most applications in distributed memory**
  - We only need the disk for long-term archival
- **These shouldn’t be called “caches” they are in-memory databases**

In-memory databases

- **Databases beyond 1 TByte (1000 Gig) can be held in highly available and resilient memory**
  - Data can be available locally – local queries
  - Return arrays and complex Java objects rather than ResultSets
  - They are much more Java friendly
- **We do this already in several banks**
  - Performance gains in 2+ orders of magnitude
  - Using C24 we can store complex trades in distributed memory
- **Just archive off to a “classic” database at the end of the day**
- **Use Google to search through it**
  - Already in use, provide amazing results

WebServices

- **In its classic form (XML/HTTP) WebServices have very little use in an Investment Bank**

- **The services pattern (SOA) and standards behind it are very useful however**
  - We can adopt WSDL and Schema but we don't have to implement them in pure XML

- **We can define SOA in WSDL and Schema etc. but implement it in Java rather than XML**

- **Java can also implement rules, constraints and security**
  - This is just not possible in pure XML
  - FpML is a good example, it includes several dozen rules

SOA

- **The "A" is for Architecture not an Application**
- **Think about Service enabling at the service**
  - Not connecting services together via some application – i.e. EAI
- **This can be done beautifully with pure Java, we only need to use XML on the outside of our borders**
- **Java is fantastically rich for defining models, constraints and rules**
- **IONA's Artix is distributed – A good SO-Architecture**
  - It can use WSDL & Schema and generate Java – very fast SOA

IDEs & Tools

- **As a rule we use Eclipse**
  - There however a good proportion using IntelliJ’s IDEA
    - Productivity with IDEA justifies the reasonable price
  - There is very little of anything else
    - Real programmers use Emacs though ☺

- **Eclipse has become a standard platform for developing traders tools**
  - Provides good integration with Windows tools

- **Atlassian’s Jira and Confluence have become almost de facto**
  - Banks are insisting that vendors use Jira to log issues/bugs
  - Jira is used internally in many banks
  - Confluence is used as an internal Wiki in many banks

I'd love to go on...

- **There's so much to cover**
  - Source control, C/C++/.NET integration, AOP, AJAX, SWT/Swing/Web development, portals etc. etc.
- **Hopefully you've got a good feel for just how much Java we use In the banking world**
  - And just how much support the banks give to the world of Java
- **For a more technical drill down please come to my Case Study**

Banking/Grid

Friday 12 noon

- **Thank you for your time and have a great Symposium**

Thank you, Gracias

Adiós!

John.Davies@C24.biz

http://www.C24.biz