Pouring Data on Troubled Markets

Quantitative Portfolio Management Technology at BGI

Eoin Woods, Barclays Global Investors

www.barclaysglobal.com/careers
www.eoinwoods.info

V1.20090306

Introductions

- Software architect at BGI
  - lead software architect for the Apex portfolio management system
  - future state architecture responsibilities for Equities and Capital Markets
  - lead software architect for Equity Shared Services
- Software engineering for ~18 years
  - Systems & architecture focus for ~12 years
- Background includes system software products, consultancy and applications
  - Tuxedo, Sybase, InterTrust, bespoke capital markets work

Who are BGI?

- Barclays Global Investors
- Probably the largest fund manager you've never heard of
  - the asset manager in the Barclays group (alongside Barclays Capital and Barclays Wealth)
  - manages $1.5t* of client assets using scientific investment management techniques
  - formed by the 1996 merger of Wells-Fargo-Nikko and BZW Asset Management
  - headquartered in San Francisco
  - employs about 4000 people in San Francisco, London and Tokyo and Atlanta, Amsterdam, Chicago, Dubai, Hong Kong, Mexico City, Munich, New York, Paris, Singapore, Sao Paulo, Sydney.
  - ~1100 of the staff work in a Technology group

(*) as of 31st December 2008

Agenda

- Introducing Apex
- The Design of the Apex System
- Delving Deeper
- Lessons Learned
- Summary

The Apex Portfolio Management System

- This talk will concentrate on one of BGI’s many systems: *Apex*
- Apex is a new portfolio management system being created primarily for the Active Equity business within the firm
- A portfolio management system is a critical piece of the fund management process, automating and supporting fund rebalancing (what to buy and sell for each fund).
- The current state is three regional systems that have grown up over 5-10 years, leading to redundancy and inconsistency across regions
- The new system needs to be consistent globally and be easier/ quicker/cheaper to scale and change than the three existing systems

The Business Drivers

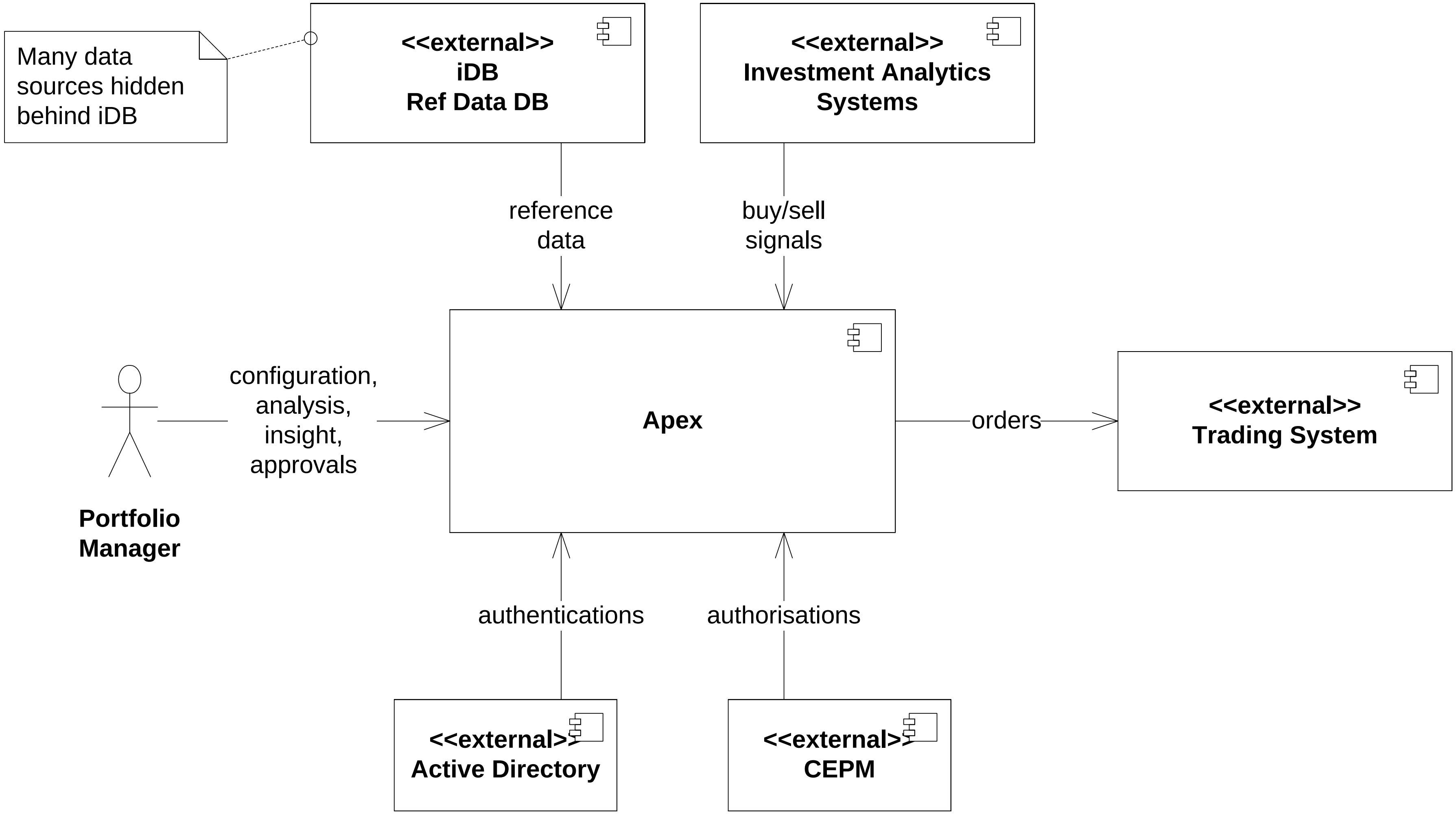
- Business process scalability (manage more money with less people)
- Sophistication of the user experience (don't get in the way)
- Geographical independence (run money anywhere from anywhere)
- Global standardisation/efficiency (do things one way, well)
- "Flexibility" (allow fund specific variation and changes to anything)
- Reliability (always on, mask infra failures, deal with business failures)
- Environment (interoperate flexibly)

And of course the implicit requirements of being infinitely fast, technically scalable, secure, and delivered in zero time!

Some of the Technical Challenges

- Sophistication of the required user experience
  - Cooper LLC were engaged to create a user interface design
  - the result is a powerful exception based interface that rarely blocks the user
    - implicit saving, asynchronous fetching, no (little) modality
  - many users come from the Unix shell and so are sophisticated users
- Long Running Processes
  - much of the business processing involves long running operations (minutes)
    - yet standard enterprise Java patterns tend to focus on transaction processing
- Lots of data from many sources
  - flat files, XML files, FTP sources, databases, messages, …
  - 180 tables between Apex and iDB
  - ~185k rows (40MB row data) typically output *per fund rebalance*

Runtime Context

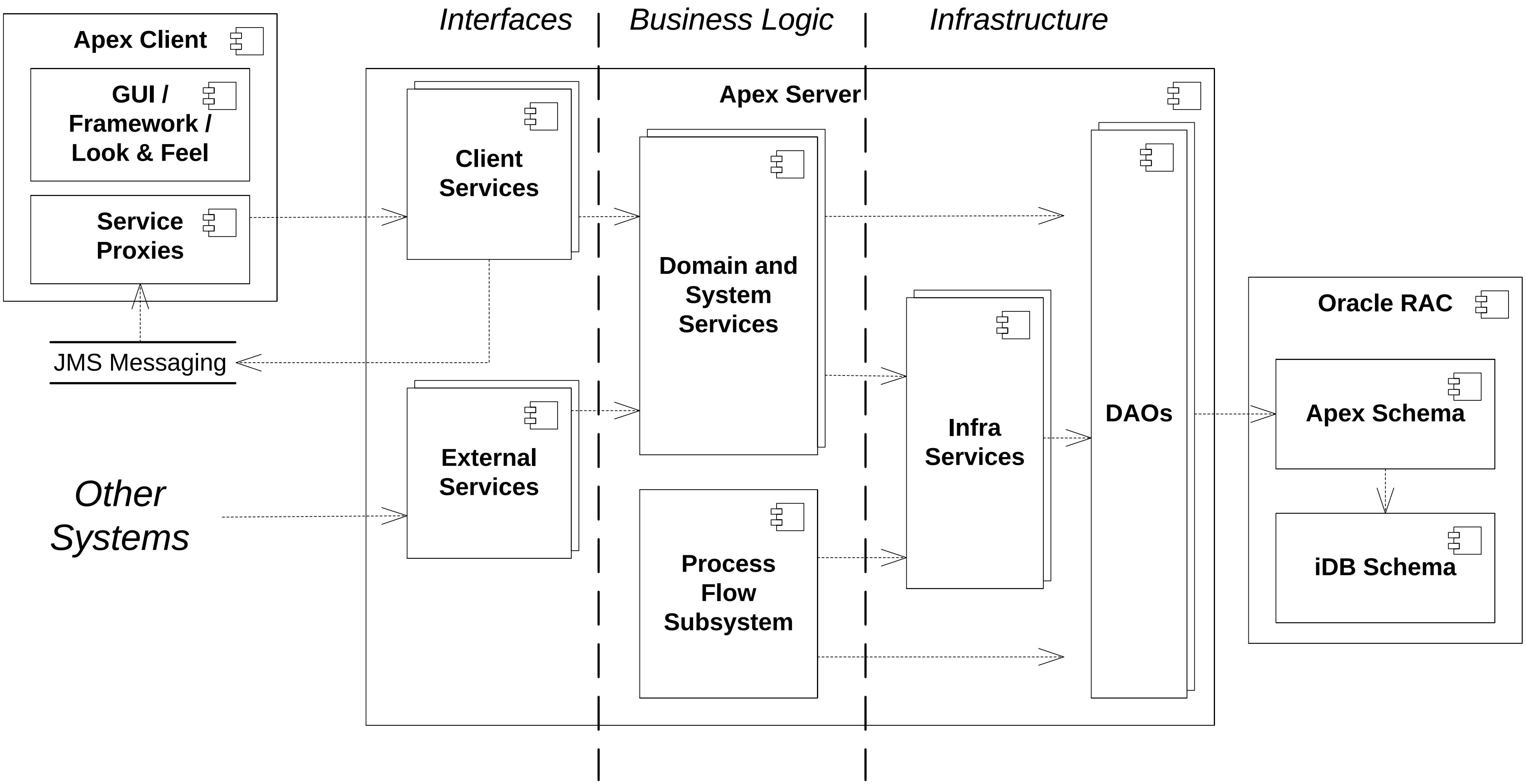


Many data sources hidden behind iDB
<<external>> iDB Ref Data DB
<<external>> Investment Analytics Systems
reference data
buy/sell signals
Portfolio Manager
configuration, analysis, insight, approvals
Apex
orders
<<external>> Trading System
authentications
authorisations
<<external>> Active Directory
<<external>> CEPM

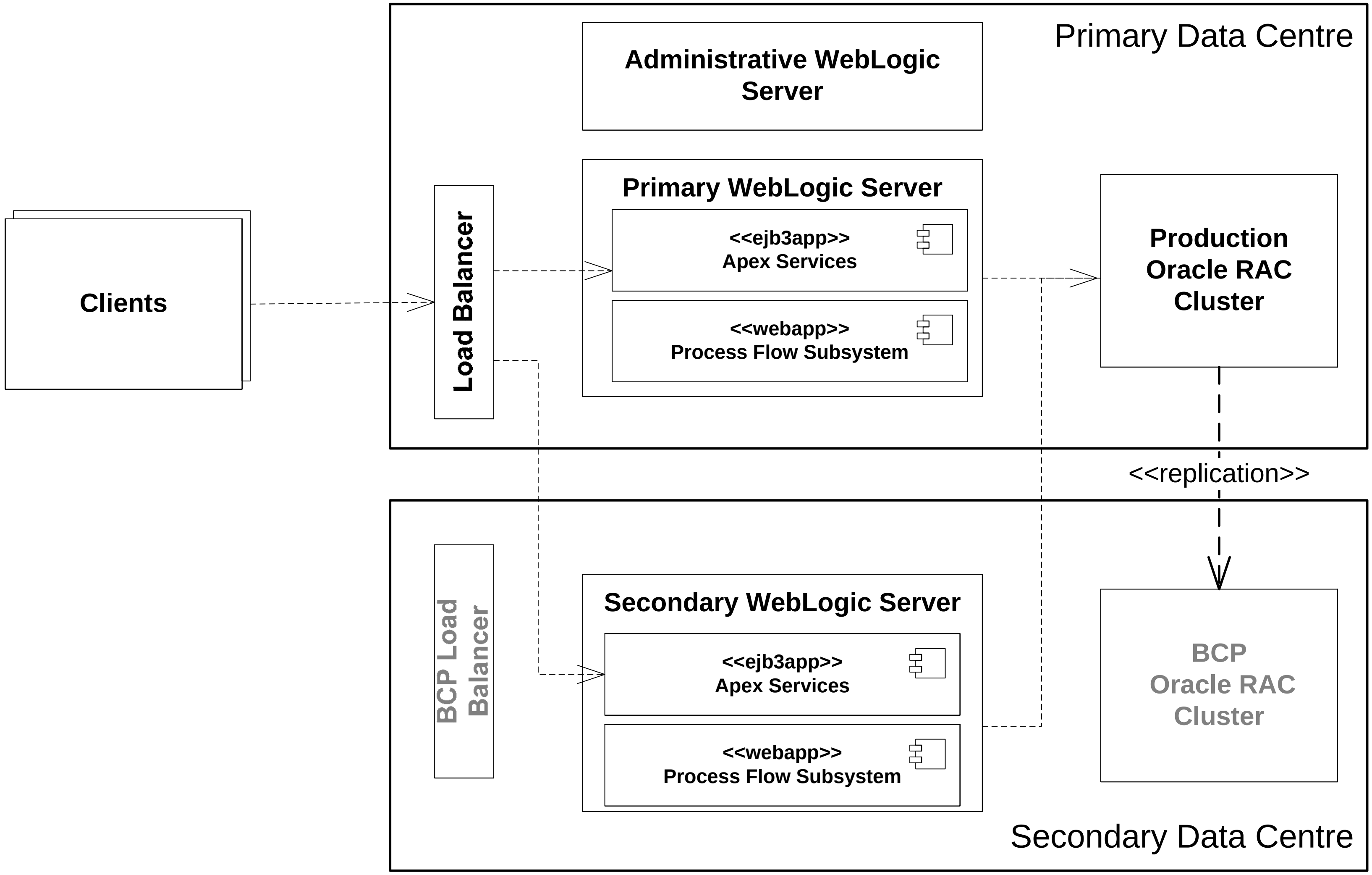
Agenda

- Introducing Apex
- The Design of the Apex System
- Delving Deeper
- Lessons Learned
- Summary

Apex's Functional Structure



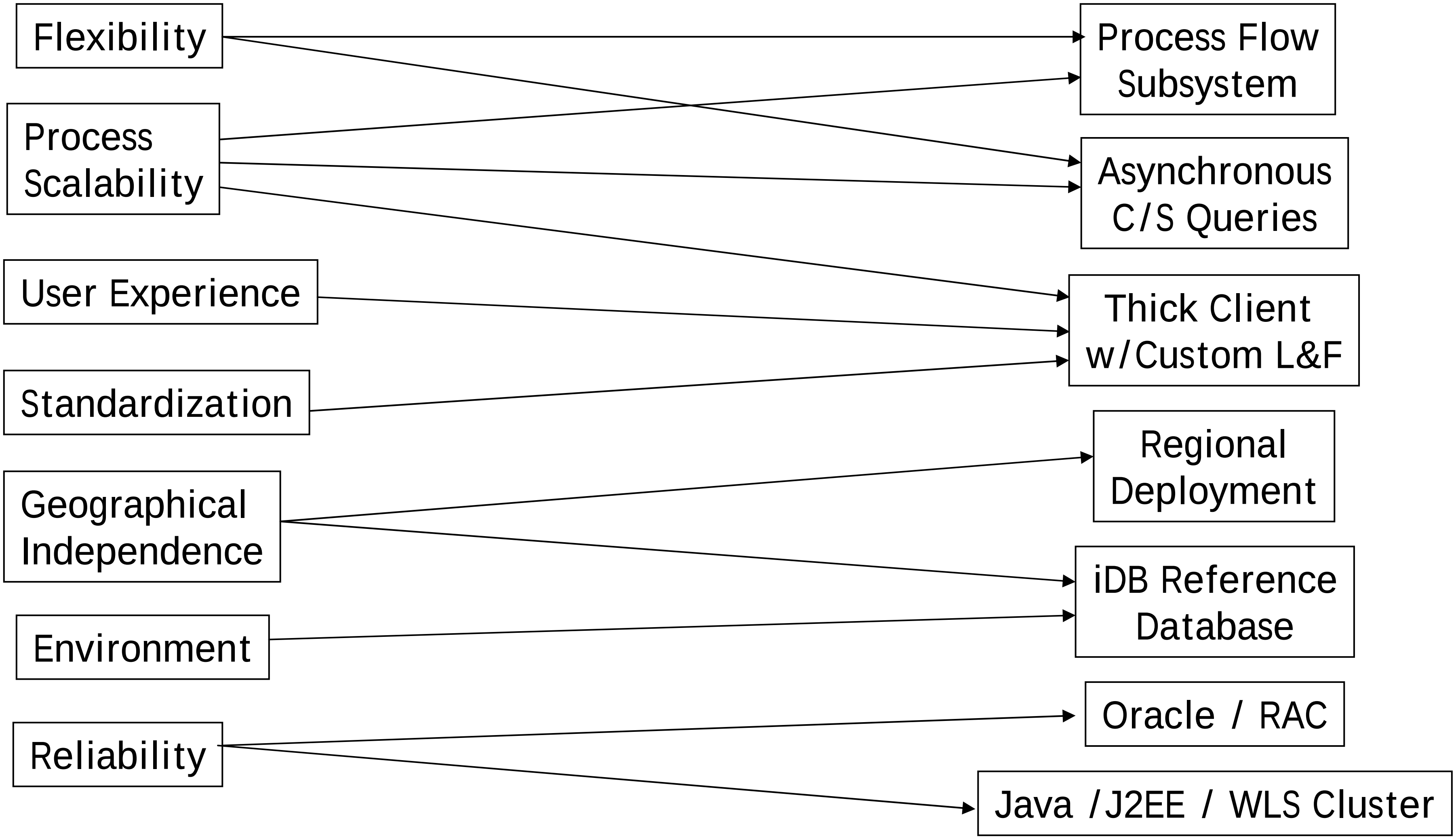
Apex's Deployment Structure



Some of the Big Decisions

- Java/J2EE in clustered WebLogic
- RDBMS store (Oracle RAC)
- Distinct “Process Flow Subsystem” (based on Flux batch engine)
- Thick client with custom look-and-feel (Swing / JIDE / BGI L&F)
  - look and feel is an implementation of the Cooper UI design
- Separate data supply (reference data) database (iDB)
  - hides the complexity of our sources from the core Apex system
- Asynchronous client/server queries (“streaming data”)
  - synchronous generic query request, asynchronous reply with meta-data
- Regional deployment

Influences for the Big Decisions



Flexibility

Process Scalability

User Experience

Standardization

Geographical Independence

Environment

Reliability

Process Flow Subsystem

Asynchronous C/S Queries

Thick Client w/Custom L&F

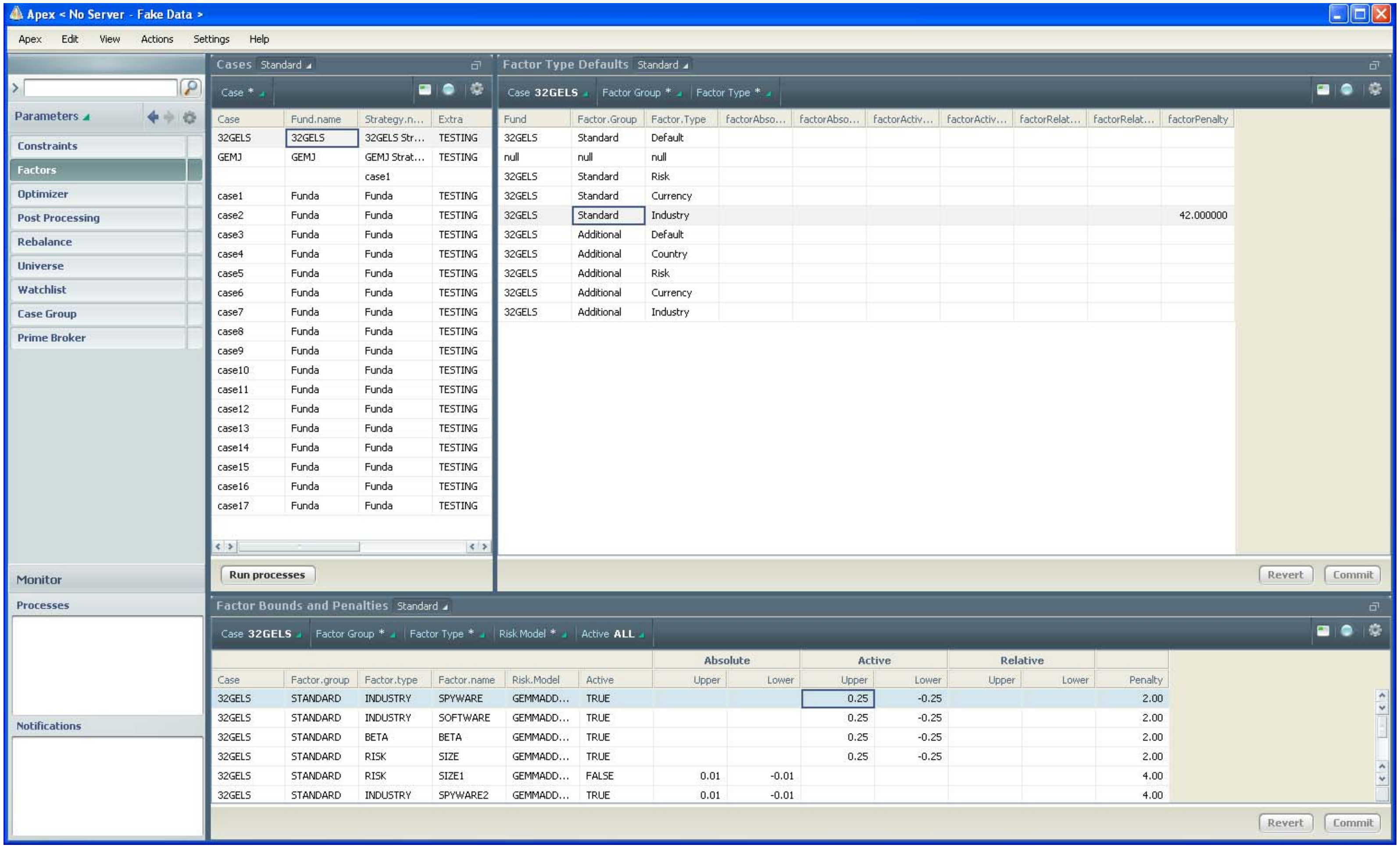
Regional Deployment

iDB Reference Database

Oracle / RAC

Java /J2EE / WLS Cluster

The Apex Client – Setting Parameters

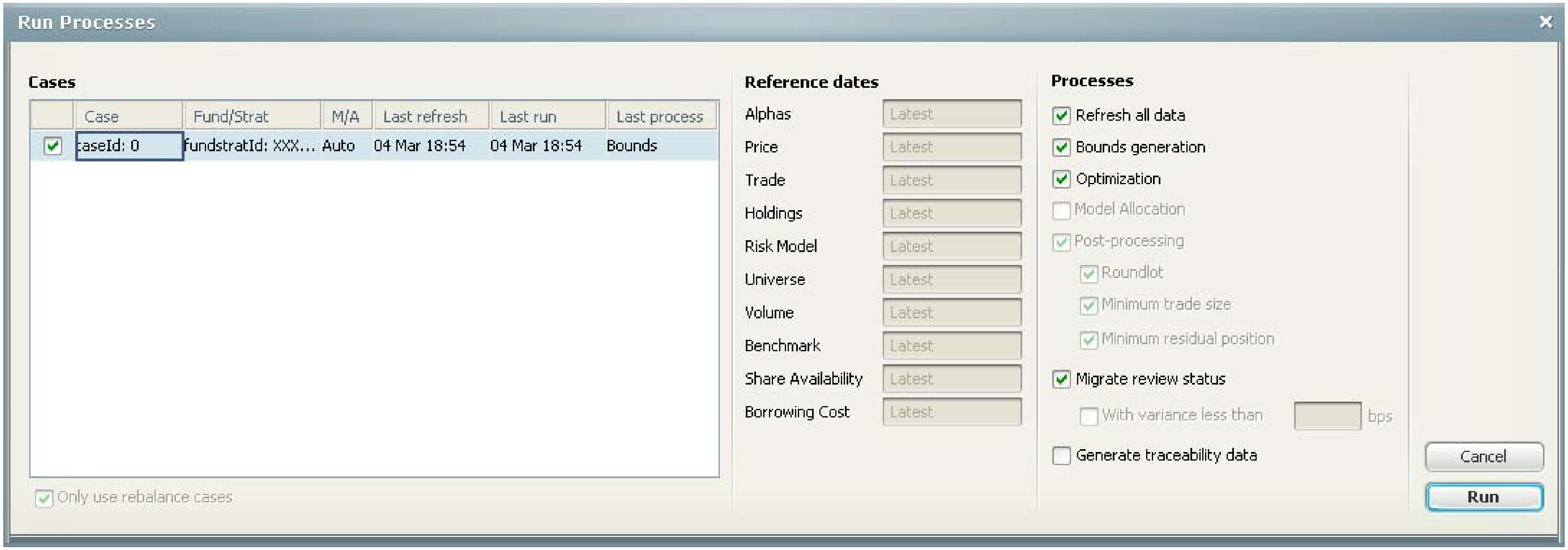


Case	Fund.name	Strategy.n...	Extra
32GELS	32GELS	32GELS Str...	TESTING
GEMJ	GEMJ	GEMJ Strat...	TESTING
		case1	
case1	Funda	Funda	TESTING
case2	Funda	Funda	TESTING
case3	Funda	Funda	TESTING
case4	Funda	Funda	TESTING
case5	Funda	Funda	TESTING
case6	Funda	Funda	TESTING
case7	Funda	Funda	TESTING
case8	Funda	Funda	TESTING
case9	Funda	Funda	TESTING
case10	Funda	Funda	TESTING
case11	Funda	Funda	TESTING
case12	Funda	Funda	TESTING
case13	Funda	Funda	TESTING
case14	Funda	Funda	TESTING
case15	Funda	Funda	TESTING
case16	Funda	Funda	TESTING
case17	Funda	Funda	TESTING

Fund	Factor.Group	Factor.Type	factorAbso...	factorAbso...	factorActiv...	factorActiv...	factorRelat...	factorRelat...	factorPenalty
32GELS	Standard	Default							
null	null	null							
32GELS	Standard	Risk							
32GELS	Standard	Currency							
32GELS	Standard	Industry							42.000000
32GELS	Additional	Default							
32GELS	Additional	Country							
32GELS	Additional	Risk							
32GELS	Additional	Currency							
32GELS	Additional	Industry							

						Absolute		Active		Relative		
Case	Factor.group	Factor.type	Factor.name	Risk.Model	Active	Upper	Lower	Upper	Lower	Upper	Lower	Penalty
32GELS	STANDARD	INDUSTRY	SPYWARE	GEMMADD...	TRUE			0.25	-0.25			2.00
32GELS	STANDARD	INDUSTRY	SOFTWARE	GEMMADD...	TRUE			0.25	-0.25			2.00
32GELS	STANDARD	BETA	BETA	GEMMADD...	TRUE			0.25	-0.25			2.00
32GELS	STANDARD	RISK	SIZE	GEMMADD...	TRUE			0.25	-0.25			2.00
32GELS	STANDARD	RISK	SIZE1	GEMMADD...	FALSE	0.01	-0.01					4.00
32GELS	STANDARD	INDUSTRY	SPYWARE2	GEMMADD...	TRUE	0.01	-0.01					4.00

The Apex Client – Running Process Flows



Run Processes

Cases

	Case	Fund/Strat	M/A	Last refresh	Last run	Last process
☑	caseId: 0	fundstratId: XXX...	Auto	04 Mar 18:54	04 Mar 18:54	Bounds

☑ Only use rebalance cases

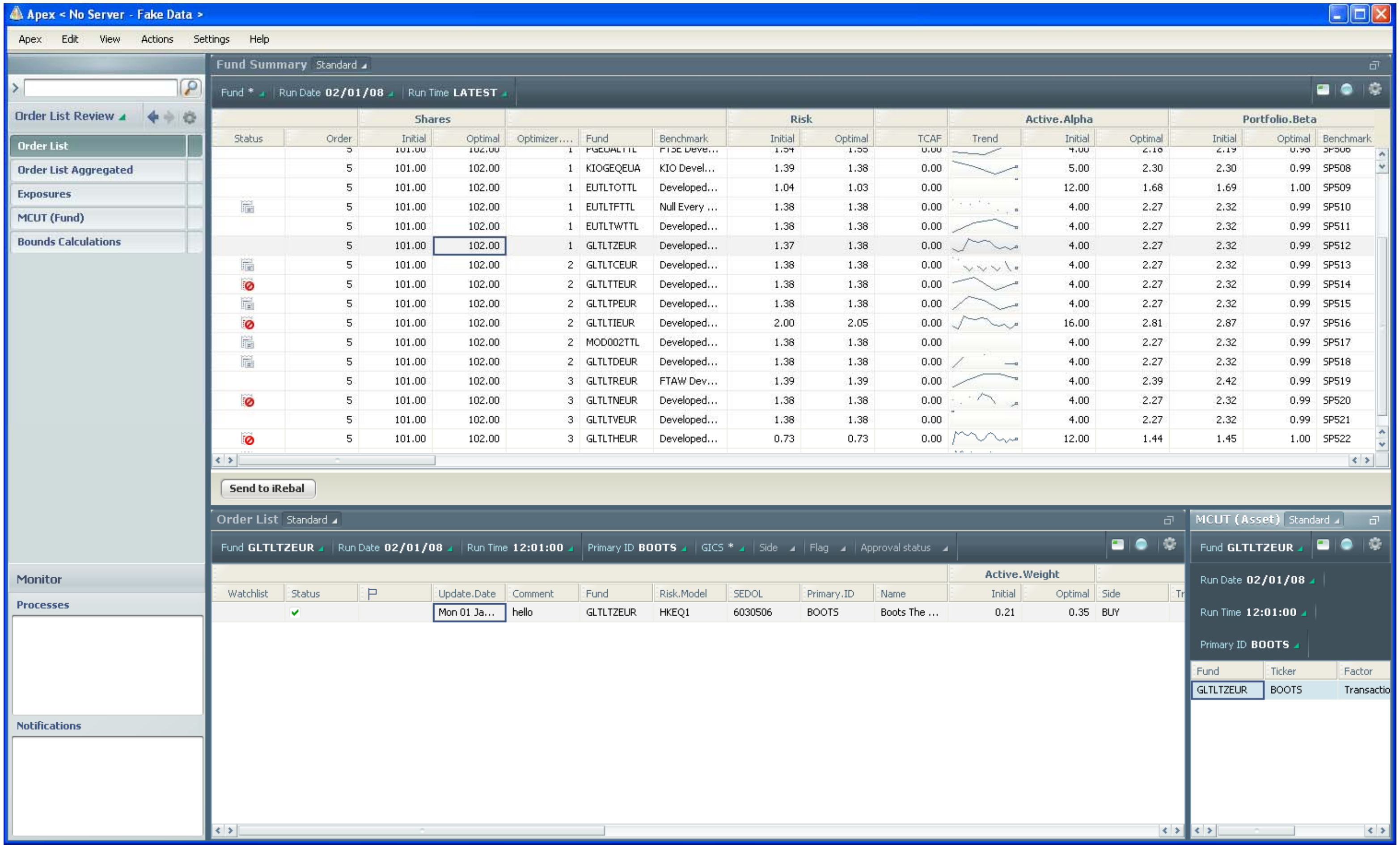
Reference dates

Alphas	Latest
Price	Latest
Trade	Latest
Holdings	Latest
Risk Model	Latest
Universe	Latest
Volume	Latest
Benchmark	Latest
Share Availability	Latest
Borrowing Cost	Latest

Processes

- ☑ Refresh all data
- ☑ Bounds generation
- ☑ Optimization
- □ Model Allocation
- ☑ Post-processing
  - ☑ Roundlot
  - ☑ Minimum trade size
  - ☑ Minimum residual position
- ☑ Migrate review status
  - □ With variance less than ____ bps
- □ Generate traceability data

Apex Client – Analysing Results



Fund * | Run Date 02/01/08 | Run Time LATEST

		Shares					Risk				Active.Alpha		Portfolio.Beta		
Status	Order	Initial	Optimal	Optimizer....	Fund	Benchmark	Initial	Optimal	TCAF	Trend	Initial	Optimal	Initial	Optimal	Benchmark
	5	101.00	102.00	1	PGEUALTTL	FTSE Deve...	1.54	1.55	0.00		4.00	2.18	2.19	0.98	SP506
	5	101.00	102.00	1	KIOGEQEUA	KIO Devel...	1.39	1.38	0.00		5.00	2.30	2.30	0.99	SP508
	5	101.00	102.00	1	EUTLTOTTL	Developed...	1.04	1.03	0.00		12.00	1.68	1.69	1.00	SP509
▤	5	101.00	102.00	1	EUTLTFTTL	Null Every ...	1.38	1.38	0.00		4.00	2.27	2.32	0.99	SP510
	5	101.00	102.00	1	EUTLTWTTL	Developed...	1.38	1.38	0.00		4.00	2.27	2.32	0.99	SP511
	5	101.00	102.00	1	GLTLTZEUR	Developed...	1.37	1.38	0.00		4.00	2.27	2.32	0.99	SP512
▤	5	101.00	102.00	2	GLTLTCEUR	Developed...	1.38	1.38	0.00		4.00	2.27	2.32	0.99	SP513
⊘	5	101.00	102.00	2	GLTLTTEUR	Developed...	1.38	1.38	0.00		4.00	2.27	2.32	0.99	SP514
▤	5	101.00	102.00	2	GLTLTPEUR	Developed...	1.38	1.38	0.00		4.00	2.27	2.32	0.99	SP515
⊘	5	101.00	102.00	2	GLTLTIEUR	Developed...	2.00	2.05	0.00		16.00	2.81	2.87	0.97	SP516
▤	5	101.00	102.00	2	MOD002TTL	Developed...	1.38	1.38	0.00		4.00	2.27	2.32	0.99	SP517
▤	5	101.00	102.00	2	GLTLTDEUR	Developed...	1.38	1.38	0.00		4.00	2.27	2.32	0.99	SP518
	5	101.00	102.00	3	GLTLTREUR	FTAW Dev...	1.39	1.39	0.00		4.00	2.39	2.42	0.99	SP519
⊘	5	101.00	102.00	3	GLTLTNEUR	Developed...	1.38	1.38	0.00		4.00	2.27	2.32	0.99	SP520
	5	101.00	102.00	3	GLTLTVEUR	Developed...	1.38	1.38	0.00		4.00	2.27	2.32	0.99	SP521
⊘	5	101.00	102.00	3	GLTLTHEUR	Developed...	0.73	0.73	0.00		12.00	1.44	1.45	1.00	SP522

Fund GLTLTZEUR | Run Date 02/01/08 | Run Time 12:01:00 | Primary ID BOOTS | GICS * | Side | Flag | Approval status

										Active.Weight			
Watchlist	Status	⚐	Update.Date	Comment	Fund	Risk.Model	SEDOL	Primary.ID	Name	Initial	Optimal	Side	Tr
	✓		Mon 01 Ja...	hello	GLTLTZEUR	HKEQ1	6030506	BOOTS	Boots The ...	0.21	0.35	BUY	

Fund GLTLTZEUR

Run Date 02/01/08

Run Time 12:01:00

Primary ID BOOTS

Fund	Ticker	Factor
GLTLTZEUR	BOOTS	Transactio

(May look a little constrained … standard specification is two 24” monitors)

Software Development

- A low ceremony version of RUP used to develop the system
  - inception, elaboration, construction, transition phases with lots of iterations
  - “viewpoints and perspectives” approach for architecture (unsurprisingly)
  - UML for architecture and (significant) design
  - continuous integration & automated testing
  - a fair number of tools (MagicDraw, Jtest, Structure101, U4J, …)
- Development team of 16 at peak, now 9 developers
  - plus tester, management and BAs
- Currently about 155 raw kloc; ~85kloc of executable code
  - 55kloc in the server, 76kloc in the client, 24kloc in shared module

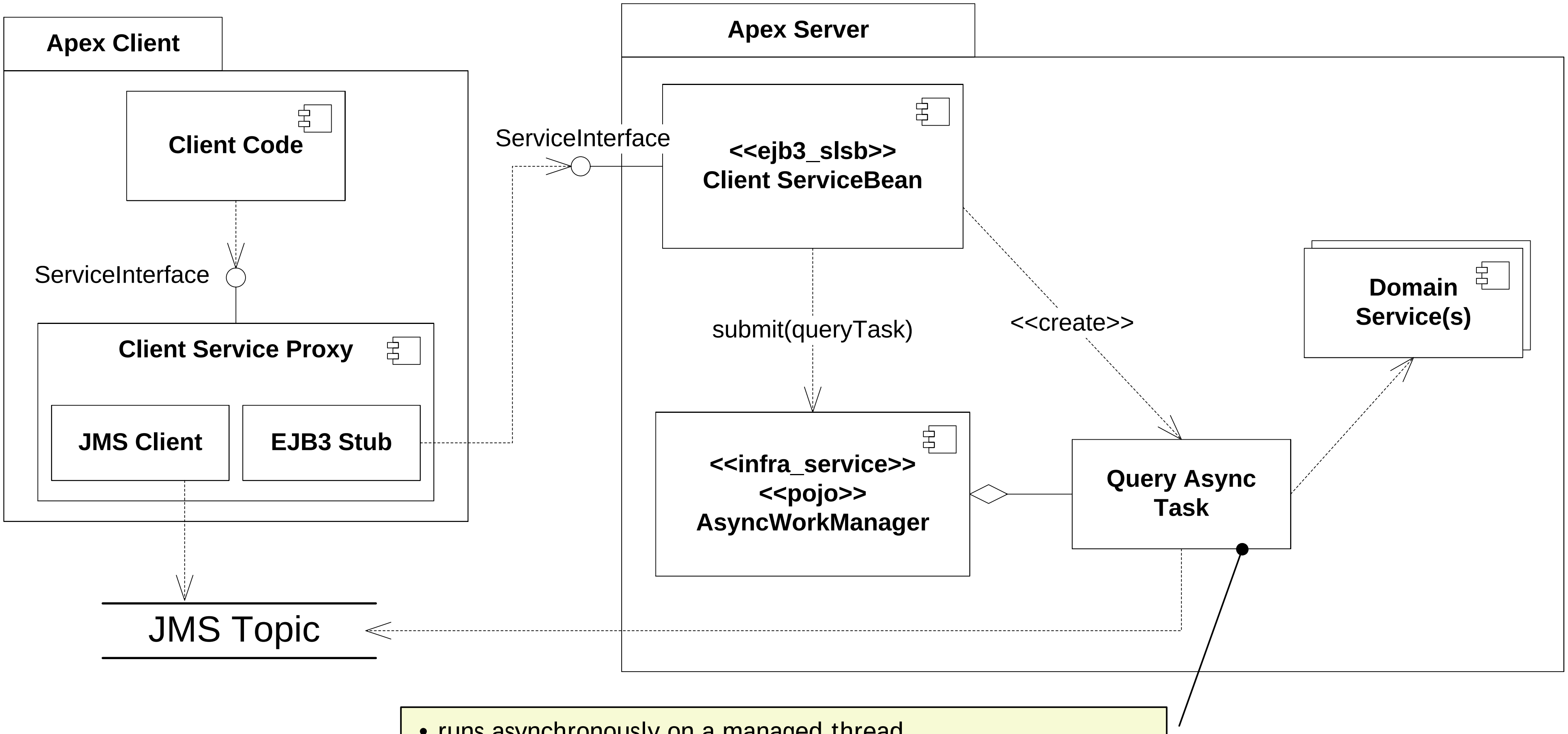
Agenda

- Introducing Apex
- The Design of the Apex System
- Delving Deeper
- Lessons Learned
- Summary

Delving Deeper

- Asynchronous Client Query Pattern
- Process Flow Subsystem
- Blending Different Types of Technology

Asynchronous Client Query Pattern



- runs asynchronously on a managed thread
- calls the domain service(s) needed to process the query
- transforms the domain objects into generic RowDTOs to return

(Note: this is pseudo UML!)

Asynchronous Client Query Pattern – Walkthrough (i)

- Client calls its service proxy, passing a callback to accept results
  - request contains a *subject* and a set of *filters*
  - Service proxy calls the server-side service via normal EJB3 invocation
- EJB service implementation checks its parameters and creates an asynchronous task object corresponding to the request type
  - the filters are passed to the task object for its use
- The new asynchronous task object is passed to the Asynchronous Work Manager for execution
- The AWM runs the task object on a WLS managed thread

Asynchronous Client Query Pattern – Walkthrough (ii)

- The task object calls the domain service(s) required
  - the filters are used to construct domain service parameters (e.g. limit > 10) or in some cases passed into the domain services to be used in HSQL
- The task object translates the domain objects returned into a generic result set for the client
  - results dispatched to the client via JMS messages
  - a set of meta-data headers are dispatched first to describe the result set
  - the data is sent as generic “RowDto” objects, which each contain one result row, with “Attribute” objects corresponding to the headers
  - the translation is done by a generic translator using OGNL
- The client service proxy receives the JMS message and calls the client callback to deliver each result row

Asynchronous Client Query Pattern - OGNL

- Generic translation from domain object to generic row/attribute form achieved via Object Graph Navigation Language

  `http://www.opensymphony.com/ognl/`

- OGNL interprets an expression in the context of Java Beans, allowing properties to be retrieved or set
  - e.g. "`fund.strategy.name`" interpreted at run time as if calling `fund.getStrategy().getName()` on the specified object

- Our Attribute objects include an OGNL expression to define how their value is derived from domain objects

- Many of the asynchronous query tasks use a standard OGNL based translator that uses the Attribute expressions and the OGNL library to translate domain objects into a row of Attribute values

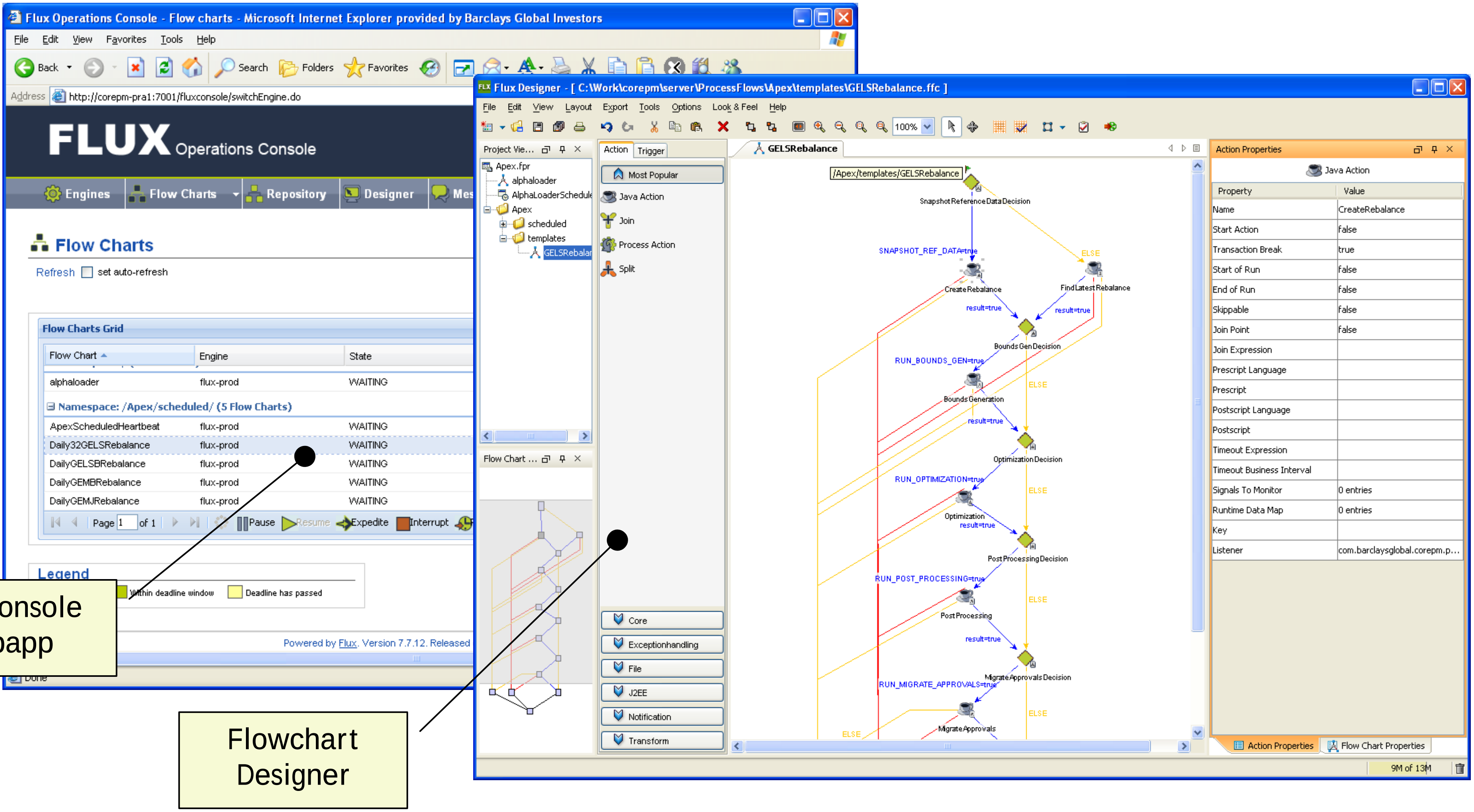
Process Flow Subsystem

- Apex’s batch subsystem (runs “process flows” containing “jobs”)
- Uses the Flux scheduler product as the core of the subsystem
  - provides the generic scheduling engine
  - includes an administration web interface and GUI tools for flowchart design
  - pure Java library (can be used as a standalone program or embedded)
  - hidden behind wrappers and abstractions but provides all of the generic scheduling functions
- Apex developers write jobs by extending (Apex) base classes that isolate our code from Flux and standardise its use
- We combine the jobs into flowcharts to orchestrate them into useful business processes that users can request or that run on schedules

Process Flow Subsystem - Flux

- Flux is a commercial Java-based scheduling package
  - not unlike an extended Quartz
  - product of the Flux Corporation (`www.fluxcorp.com`)
- Very flexible, extensible and embeddable
  - also quite complicated and needs to be used carefully
- The Flux model is one of “flowcharts”, “triggers” and “actions”
  - trigger – file arrival, time delay, cron-like schedule and custom triggers, …
  - action – run an executable, send a message, call Java, indicate an error, …
  - flowchart – a directed graph of triggers, actions and control structures
- Our use so far has been simple
  - manual and cron like schedule triggers, Java and error actions

Process Flow Subsystem – Flux Administrative Interfaces



Address http://corepm-pra1:7001/fluxconsole/switchEngine.do

Flow Charts

Refresh □ set auto-refresh

Flow Charts Grid

Flow Chart	Engine	State
alphaloader	flux-prod	WAITING
Namespace: /Apex/scheduled/ (5 Flow Charts)		
ApexScheduledHeartbeat	flux-prod	WAITING
Daily32GELSRebalance	flux-prod	WAITING
DailyGELSBRebalance	flux-prod	WAITING
DailyGEMBRebalance	flux-prod	WAITING
DailyGEMJRebalance	flux-prod	WAITING

Page 1 of 1 Pause Resume Expedite Interrupt

Legend

🟩 Within deadline window 🟨 Deadline has passed

Powered by Flux. Version 7.7.12. Released

Done

Apex.fpr
- alphaloader
- AlphaLoaderSchedule
- Apex
  - scheduled
  - templates
    - GELSRebalan

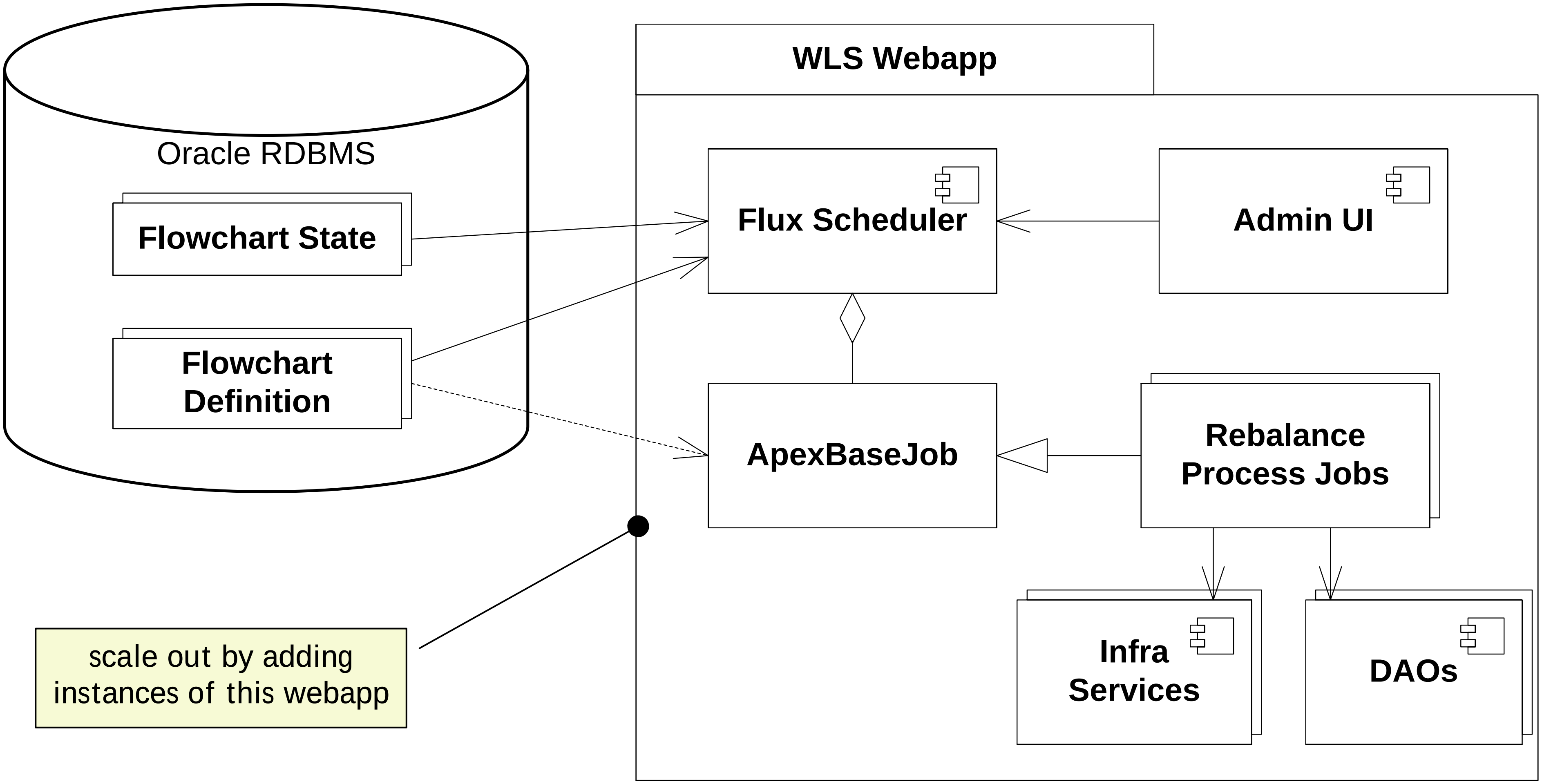
Java Action

Property	Value
Name	CreateRebalance
Start Action	false
Transaction Break	true
Start of Run	false
End of Run	false
Skippable	false
Join Point	false
Join Expression	
Prescript Language	
Prescript	
Postscript Language	
Postscript	
Timeout Expression	
Timeout Business Interval	
Signals To Monitor	0 entries
Runtime Data Map	0 entries
Key	
Listener	com.barclaysglobal.corepm.p...

Ops Console webapp

Flowchart Designer

Process Flow Subsystem - Design



scale out by adding instances of this webapp

(Note: again any similarity to UML here is illusionary!)

Blending Different Types of Technology (i)

- Blend of mainstream and niche, commercial and open source
- Mainstream commercial:
  - Java 1.5 and 1.6, EJB3, JPA, WebLogic Server, Oracle 10.x RAC, JIDE
- Niche commercial:
  - Flux scheduler, CPLEX, JMSL Numerical Library, Quadbase Libraries, JEP Parser
- Mainstream open source:
  - Spring, Hibernate, JavaHelp, Commons Lang/Logging/File/POI/…,
- Niche open source:
  - XStream, OGNL, Ostermiller Utilities, JDIC

Blending Different Types of Technology (ii)

- Mainstream Commercial

  + usually does what it says in the documentation, adequate information available

  + well known and understood, skills & experience readily available

  - vendor interaction is usually slow, product development relatively slow

  - new or obscure features can be hard to figure out

- Niche Commercial

  + highly responsive, motivated vendors

  + fast moving products with lots of frequent smaller releases

  - may have significantly less field testing (i.e. need to test yourself)

  - information and skills may be difficult to obtain

Blending Different Types of Technology (iii)

- Mainstream Open Source

  + generally very reliable, due to wide use

  + information and skills widely available

  + source code availability means you can do your own investigation

  +/- usage often assumed to follow a pattern, which you need to follow

  - integration with other products often needed and can be complicated

Niche Open Source

  + the functions are often fantastic and exactly what you need

  + often supported by a small enthusiastic group of committers

  + source code availability means a certain degree of self sufficiency

  - less widely used so less testing completed and less knowledge available

  - when you have a problem you may well be on your own

Agenda

- Introducing Apex
- The Design of the Apex System
- Delving Deeper
- Lessons Learned
- Summary

Lessons Learned

- Testing 3rd party components takes more time than you think
  - assuming certain behaviours or failure modes can cost a lot of time if wrong
- A separate read-only reference data database worked very well
  - separates concerns, team specialisation makes development more efficient
- Interactive work and bulk processing have very different profiles
  - e.g. latency to the database *really* matters for bulk operations
  - two data centres means modest latency from the secondary to the db
  - for *bulk* operations (e.g. large JPA flush) this causes significant slowdown
- Hibernate entity navigation needs to be done carefully (i.e. avoid N+1)
  - naive navigation of a persistent object model results in a lot of queries
  - may not notice for interactive processing; batch means 20,000+ sub-selects!

Lessons Learned (ii)

- Each type of software brings its own challenges and strengths
  - we've been pretty happy with the software we've chosen
  - had to learn to deal with the foibles of each type
- Investing in a domain model was time and money well spent
  - a lot of business knowledge in the domain model
  - well structured and normalised model means change is much easier
- Monitoring is more important (and harder) than you think
  - we had monitoring from day-1 but you always find you need more
- OGNL based transformers can be brittle
  - expressions embedded in the code can't be type checked
  - need strong unit tests or mistakes result in problems at runtime

Agenda

- Introducing Apex
- The Design of the Apex System
- Delving Deeper
- Lessons Learned
- Summary

Summary

- Apex is a new portfolio management system being built at BGI
- In many ways a conventional J2EE system, Apex faces some unusual challenges and meets these by using
  - a very sophisticated rich Swing client with a custom look & feel
  - batch processing via an embedded batch scheduler
  - a generic client query mechanism using asynchronous meta-data driven result sets
  - a diverse blend of mainstream and niche, commercial and open source technology
- We learned a number of useful lessons as a result of specific characteristics of Apex, but we think others will find them useful too

Acknowledgements

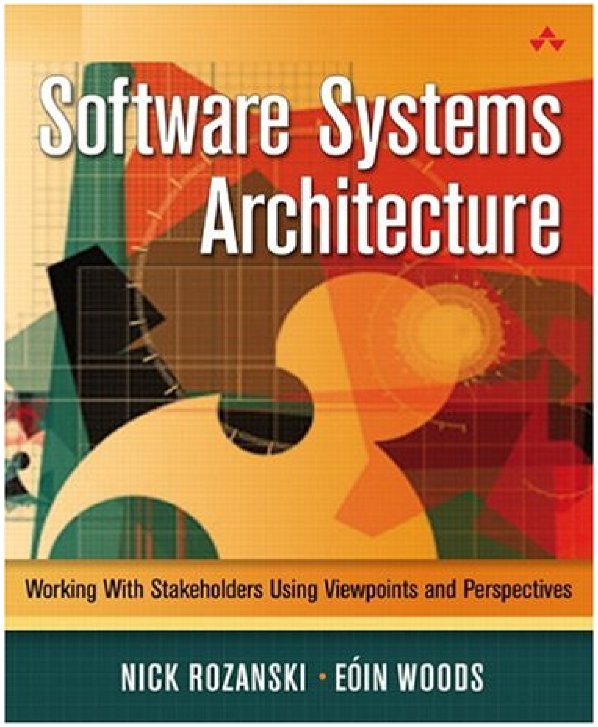
- The Apex Team*
  - Management: Dale Campbell, Phillip Sabbagh
  - Requirements & Test: Ed Hwang, Alex Rush, Nick Monge
  - Team Leaders: Brian Compton, Josh Outwater
  - Engineers: Richard Francis-Jones, Gerard Guillemette, Mark Kamiya, Wira Pradjinata, Roger Tanuatmadja, Rajat Tikoo
  - Database Admin: Sarah Brydon
- The iDB Team*
  - Russ Vernick, Raja Kurapati, Prashant Mehta, Alex Black
- The entire Active Equity Business who have funded and supported us

**As of March 2008 – many others have been involved over time and we gratefully acknowledge their efforts also*

More on the Architectural Approach

Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives

Nick Rozanski & Eoin Woods
Addison Wesley, 2005


Software Systems Architecture
Working With Stakeholders Using Viewpoints and Perspectives
NICK ROZANSKI · EÓIN WOODS

`http://www.viewpoints-and-perspectives.info`

Eoin Woods
Barclays Global Investors
eoin.woods@barclaysglobal.com
www.eoinwoods.info