

coding {the} architecture

An Architecture Case Study

Simon Brown
Sergio Anecchiarico

www.codingthearchitecture.com

1

Tonight's topic is an architecture case study, where we'll be looking at the architecture behind a small software system. We'll be covering the business problem and the possible solutions before moving on to talk about the option that we chose, both at the architecture and implementation levels.

For those new to the user group, the presentation will last 20-30 minutes, we'll have a 5 minute break, then come back together for discussion for the remainder of the session.

Following this, you're welcome to join us at The Crown (just around the corner) to continue the discussion.

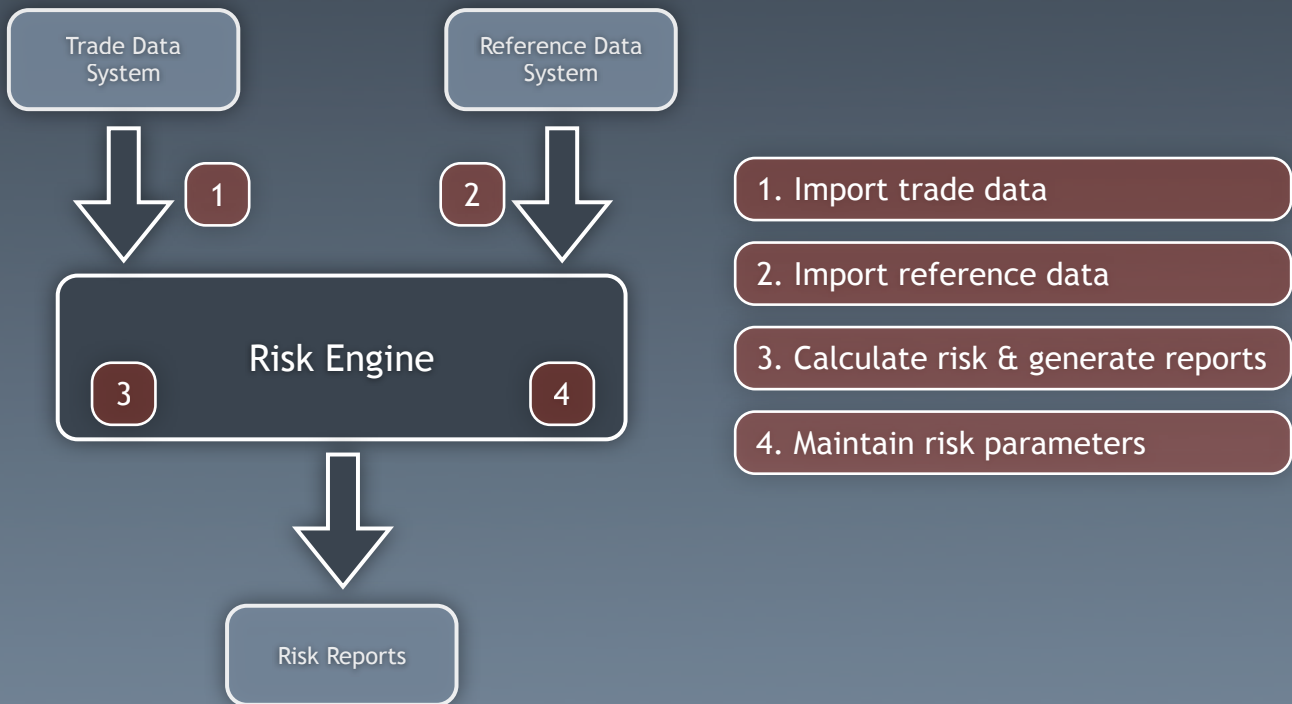
Business problem

www.codingthearchitecture.com

2

Let's start by looking at the business problem.

Business problem



www.codingthearchitecture.com

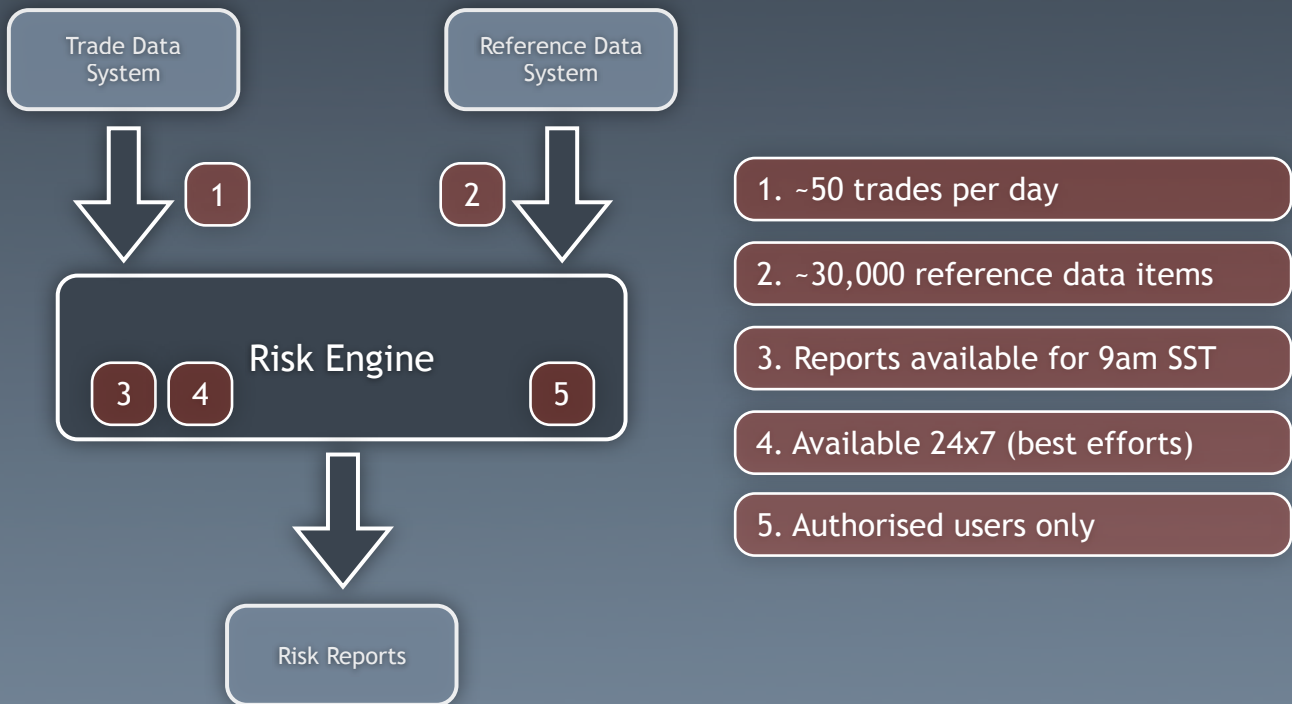
3

The business problem for this project was simple : the bank wanted to start trading a new type of product and needed a system that could tell them how much risk they were exposed to.

The bank had existing trade and reference data systems, and these were already configured to export data as tab/pipe delimited files. But they needed the risk system.

And they needed it fast; in just four weeks. Also, this would be a tactical/interim solution with a lifetime of 3-6 months.

Business problem



www.codingthearchitecture.com

4

Key non-functional requirements :

- Performance; reference data available close of business London, reports must be available for start of day in Singapore.
- Scalability; thousands of trades by 6 months.
- Availability; reports must be available 24x7 (on a best efforts basis).
- Security; only authorised users should be able to access the reports.
- Audit; it must be clear what data has been used in calculating the risk figures.

Options

www.codingthearchitecture.com

5

Given that this is IT, there are several ways of solving a particular problem.

Options

Option 1 : Microsoft Excel

www.codingthearchitecture.com

6

This was the most obvious solution, particularly given that some of the risk calculations had been documented and implemented in Excel. It would have been a few weeks work to complete the calculations for other trade types. One thing that the Excel prototype did prove was that the risk calculations were quite complex; including many arithmetic steps and reference data lookups.

But this option was (1) deemed not secure enough and (2) there was no control over the data.

Options

Option 2 : Desktop application

www.codingthearchitecture.com

7

A desktop application was the next option, providing a more structured way to access and manipulate the data. This wasn't necessarily a rich GUI application, more of a standalone process that just happens to be running on a desktop computer. This was appealing because it required no infrastructure setup, but would compromise security and availability. Also, given the complexity of the risk calculations, there were some doubts over whether the standard desktop build would be able to support the potential trade volumes. Typically, risk calculations are the sort of thing that are suitable for grid computing, so this led us onto our next option.

Options

Option 3 : Server-side Java application

www.codingthearchitecture.com

8

A full server-side application was the next logical option, with Java as the technology choice because this is the strategic choice of technology for the bank. Given that this is effectively a batch system, the application could have been anything from a full Java EE application server through to a standalone process. However, given the complexity of the risk calculations, experience suggested that we might need to scale out these individual application processes in order that we could guarantee the risk calculations would be complete in time. After all, any delay in the availability of the input data could impact the report generation process.

While this option made a lot of sense from an architectural perspective, there's no way that it could have been built in the available time. Four weeks to build-out a horizontally scalable server-side platform would be tough, to say the least!

Recommendation

Option 1 : Microsoft Excel

www.codingthearchitecture.com

9

Given the timescales, our recommendation was to go with Excel because the bank could use some of their own in-house resources to “complete” the proof of concept that we had been shown. However, the bank really liked the idea of a server-based application because it gave them a good feeling about aspects such as security, ability to audit the input data and that nobody would be able to take a copy of the application and start running their own version of it. The biggest problem was the timescales though; (1) the risk calculation logic was complex to build and (2) experience suggests that we would have needed a scale-out architecture to satisfy the performance/scalability NFRs.

Options

Option 4 : Server-side Java application, with reduced scope

www.codingthearchitecture.com

10

After some discussion, we came up with another option that kept the benefits of a server-side application but would be much, much quicker to build. The bank decided that since this was a tactical solution, they could live with less accurate numbers until the full solution was delivered and they scaled back the complexity of the risk calculations quite significantly. Probably <10% of how complex they were, yet the numbers were ~90% accurate.

Technology selection

www.codingthearchitecture.com

11

In terms of the architecture, this is effectively a batch system that reads in some files and generates some reports, so could have been implemented using a standalone Java process.

Technology selection

Apache Tomcat as the application container

www.codingthearchitecture.com

12

However, we decided to deploy it into Tomcat as a web application instead; (1) Tomcat is well-known in the bank and people understand how to support it, (2) the bank's Veritas Cluster Server (VCS) infrastructure is already configured to monitor Tomcat instances and fail them over to another node, (3) it made deployment really easy (single WAR file) and (4) a webapp provided a way to easily implement some of the requirements (distribute reports globally to authorised users and maintain risk parameters).

Technology selection

Sybase Adaptive Server Enterprise for persistence

www.codingthearchitecture.com

13

We needed somewhere to store some data (e.g. security credentials, risk parameters, etc) but the core batch functionality could have been satisfied without a database – we could have loaded everything into memory and generated the reports off the internal data structures. But we chose to use a database; (1) it was an easy pattern for those supporting the system to understand, (2) the reports became (more or less) SQL queries and (3) a tactical solution is never tactical and using a DB just left more future options open to us should the requirements change. Having seen the original set of requirements, we knew there were some interactive reporting features that would have been easier to implement on top of a database.

What we built, how we built it

www.codingthearchitecture.com

14

This application fits the architectural pattern (dislike that phrase) I've called "the munge-er". Stuff goes in – munge-ing happens – stuff comes out. Aka "the sausage machine".

More specifically, the application took in ->
trade data (amounts, prices, dates, rates, yields, etc),
counterparty data (customers),
counterparty hierarchy data (customers' relationships, rates
data (thrown away)

Munging -> trade level credit risk exposure calculations

Spat out -> risk reports delivered over the web

Classic old school file processing application

How we tackled the implementation

Key design decisions + why

Technologies chosen and rejected

Mistakes

Approach

- Take a day to do design up front
- Document it
- Split into tasks
- Estimate and plan
- Communicate amongst team
- Sit in the same room

www.codingthearchitecture.com

Producers

- Thread pool for timing + resilience
- Thread pool service
- Lightweight work queue
- All core Java

www.codingthearchitecture.com

16

Pattern is very over-used term.

Does add a common naming scheme to design

Producer / consumer is classic pattern from the concurrency chapters of patterns books

Add work items to a queue (enqueue) in response to events.

Remove work items from the queue (dequeue) and process.

Event driven = overstating it. Very few events. “Event triggered” more like it.

Producers = file system poller tasks.

Every 2 min, ask the file system for the file.

Run once, then thread returned to pool

File system error only kills one task – pool controls creation of replacement threads

Thread pool service provides lifecycle management for pool

Work Queue – one java object (BlockingQueue)

Consumers

- Concurrent programming is hard, so let's not bother
- Make this decision explicit in the implementation

www.codingthearchitecture.com

17

Key design decision to keep munge-ing single threaded

- No concurrent batch db updates
- No calculation complexity
- No concurrent file system reading / report writing / file archiving
- No need to speed it up
- Much simpler testing

Make this decision explicit in the implementation

Perverse decision to use a thread pool of fixed size = 1

Get all the benefits of the thread pool managing service built into java libraries, but guaranteed single threaded execution from that point on.

Rest of the Munge-er

- Typical OO app
- Service layer
- DAOs
- Components for archiving files, writing reports, parsing files, and so on.
- Documentation / code quality

www.codingthearchitecture.com

18

Not going into detail. Munge-er is your typical OO application
Various components (over blown term – often just an interface
and a class)

Each a couple of mandays of effort

Technology selection (again)

- Hudson
- Subversion
- Directed unit testing
- Spring
- Left out in the cold - Hibernate and Maven
- With best intentions - code reviews
- Some mistakes...

www.codingthearchitecture.com

19

Hudson was our friend

Trivial to set up (10 min – really)

No learning curve for those that hadn't used it

Pretty

Reliable, stable, juts runs the build – doesn't try to be the build

Second generation CI tool. Consumer appliance vs Cruise

Control's mecano set. Apple vs Linux.

SVN

Lucky escape from Clearcase

Run on a developer PC

Nightly back up

Hudson integration

Unit testing

Measure with Emma code coverage

Neat Eclipse integration, pretty graphs, ant integration

Make sure we test the stuff we care about – calculation engine

Summary

- Treat it as a one month iteration
- Design and plan up front as much as possible
- Use tools that at least one team member knows well
- Use the API/libraries
- Keep it simple so you have time to maintain quality

Break



www.codingthearchitecture.com

Discussion

What do you think of the decisions made?
Would you have done the same?
How would you have approached a
project like this if the time constraints
weren't imposed?



www.codingthearchitecture.com

coding {the} architecture

Website

<http://www.codingthearchitecture.com>

London User Group

Monthly mix of presentation and discussion

Google Group

<http://groups.google.com/codingthearchitecture>

www.codingthearchitecture.com

23

A site and community for aspiring and experienced software architects.

All of the contributors are practising software architects, who primarily work within the finance and capital markets industry within the City of London.