RESTful Approaches To Financial Systems Integration

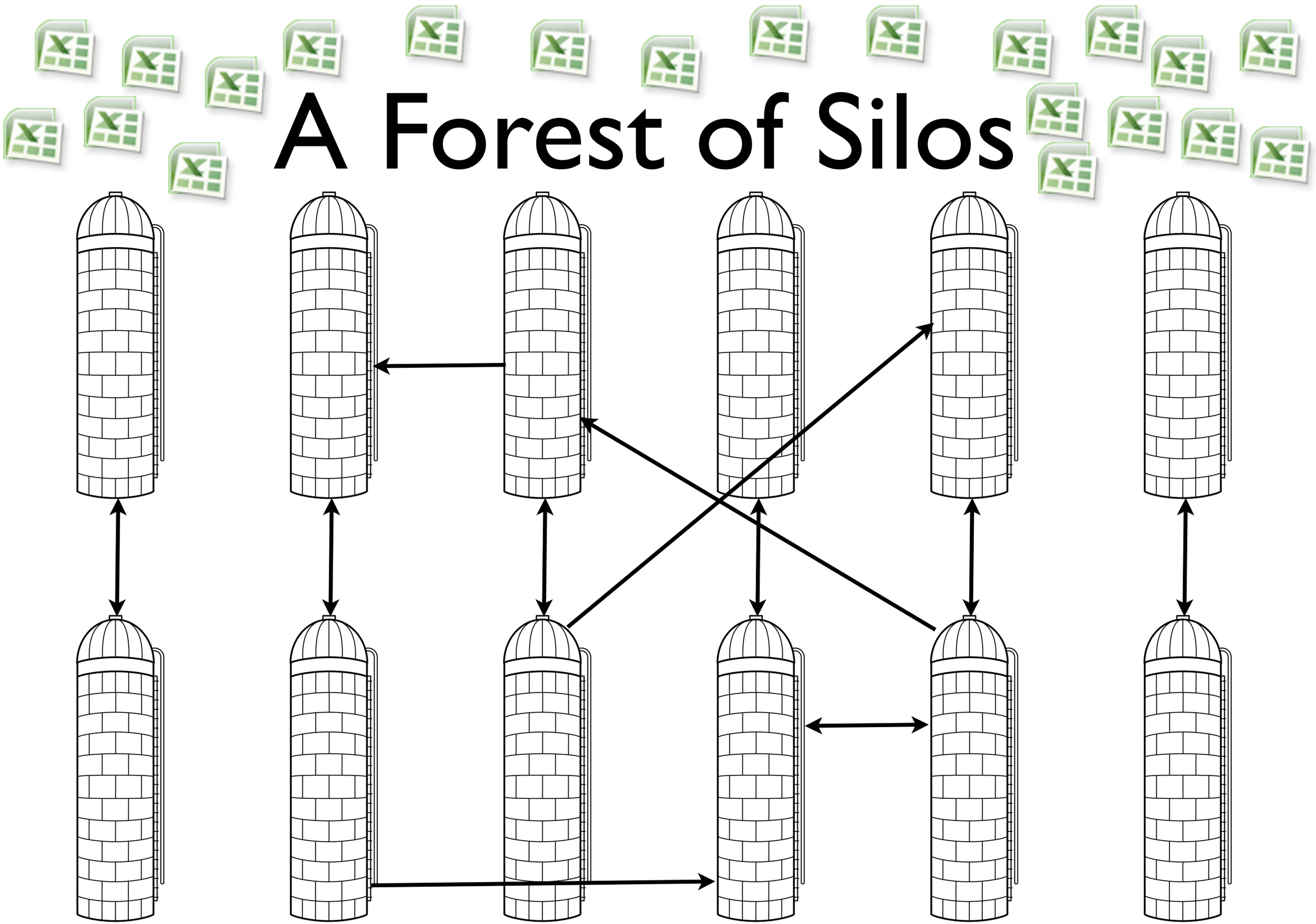
Kirk Wylie
qCon London 2009
kirk@kirkwylie.com, http://kirkwylie.blogspot.com/

About You

Introduction

- **Integration Problems in Financial Services**
- **REST to the Rescue**
- **Applying REST to Financial Services**
- **Questions**

A Forest of Silos



How Do We Get Into This Mess?

- **Every desk wants their own system**
  - Political and technical limitations to über-systems
  - Techies have to understand a particular business very well
- **Upgrading packaged software virtually impossible, so nobody does**
  - If one instance doesn't scale, add more! Even better if it's a newer version and the two don't have compatible data models
- **Different requirements for different levels - Front, Middle, Back Office**
  - Needs are completely different but they **must** communicate
- **Systems never die**

Integration Approaches

- **Flat Files & Email**
  - Fine for batch jobs, but what if you need to vary the frequency?
- **Database-based Approaches**
  - Select out of someone else's database, possibly using ETL technology. But what happens when you need to change the schema?
- **MOM**
  - Pump messages to queues and topic-based systems. But what happens when publishers and consumers can't agree on a rate?
- **SOA To The Rescue!**
  - Whose SOA? Which bus? What happens when each commercial vendor thinks their bus is the right one?

What's The Problem?

- **Systems upgrade on different schedules**
  - Moving one silo in lockstep hard enough. Convincing two teams to move together?
  - What about the system you're not allowed to touch anymore?
- **Every system a different set of tools**
  - No commonality of infrastructure or training for developers
- **How do you get data into Excel?**
  - Any system which doesn't consider the trader's pathological dependency on Excel is doomed to failure
- **Most approaches are "leaky"**
  - One side's choice of technology is forced on the other. Not so great when that side is horrible to work with!

REST to the Rescue!

REST to the Rescue!

XML/JSON!

REST to the Rescue!

XML/JSON!

Web Tech!

REST to the Rescue!

XML/JSON!

Web Tech!

Cool Kids!

Defining REST

- **Entities Have Uniform Names**
  - Every entity has its own name and uniform location
- **Use A Limited Set Of Verbs**
  - HTTP Put, Get, Delete, Post all you need for CRUD operations
- **Use Content Negotiation**
  - Client says what it can support, server gives it the best match
- **HATEOAS**
  - Hypertext As The Engine Of Application State

HATEOAS

- **Client applications navigate through links**
  - Clients never assume anything about the internal structure of the application beyond the defined content encoding
  - In particular, "deep-linking" should be avoided wherever possible!

- **Don't store client context or state on the server, keep it with the client**
  - Resource providers don't know anything about how a client is navigating through the application, so can scale better

- **Allows providers of resources to manage them**
  - Can change hosts, protocols, encodings based on client and configuration details without clients having to be updated

- **If your application has a single URL that defines entry to the system, yer doin' it right.**

Defining an Entity

- **An Entity is anything that can be individually named**
  - Most database tables are logically entities, but usually a RESTful entity includes much more data than just a single row
- **Entities get URLs**
  - Access the current state of that entity using that URL
  - Change the state of that entity using same URL
- **Entities have relations between them**
  - Most clearly represented as hypertext
  - Nothing stops you from delivering related entities with each other (for example, a company and its 15 top traded bonds)
  - Can deliver groups of entities at once, either as hypertext lists or as batches of actual content

XML/HTTP

- **XML excellent for RESTful integration**
  - Use of tooling or hand parsing using XPath or DOM walking
- **HTTP excellent protocol**
  - Client-initiation helps satisfy HATEOAS principles and avoid pumping data into the ether
  - Can use huge set of HTTP based assisting technologies
- **Everything speaks it**
  - Any language which can't process XML over HTTP will be extended or replaced by one which does
- **Solves the Excel problem**

Just XML/HTTP?

- **No, you can do RESTful services with a variety of encodings**
  - HTML, JSON, CSV, FIX, XLS are all good candidates in a financial services context.
- **No, you can do RESTful services over a variety of protocols**
  - HTTP is the most prominent, but FTP, SMTP, JMS, HTTP, Directory Scanning can all be used
- **I'm focusing primarily on XML/HTTP**
  - This solves the Excel problem particularly well
  - Financial Services firms have a lot of XML already flying around
- **Gopher probably the first RESTful service**

Actual Implementation: FOSSA

- **Standardized way to integrate applications at a medium-sized ($600MM/ year) derivatives trading group**
- **Used for Inter- and Intra-application integration**
- **5 trading systems (one in-house), 2 back-office systems, traders addicted to Excel**
- **No code sharing except for analytics library**

FOSSA Architecture

- **All entities exposed as XML over URLs**
  - Standardized URL naming structure, but still used gatekeeper URLs
- **Asynchronous updates provided as XML over JMS infrastructure**
  - Entities had meta links that indicated the precise subscription parameters necessary to receive updates
- **Cross-site support with intelligent proxies**
  - Read-through, asynchronous update listening, hot startup all supported for single applications spanning 4 sites in 3 continents
- **Heterogenous environment**
  - Producers/consumers in C#, Java, C++, Python, Tcl, Excel VBA
  - Linux, SPARC Solaris, Solaris x86, Windows

Handling Upgrades

- **Provider of data upgrades**
  - Check the Accept header for MIME types the consumer can support, and serve the best one. Transform on the fly if necessary.
  - Use your single input URL to change which deep URLs clients access
- **Consumer of data upgrades**
  - Provide multiple MIME types in the Accept header in order of preference. Make sure you still support everything that's in the wild!
  - Avoid deep linking!
- **Don't use brittle parsing!**
  - Postel's law reigns supreme
  - Most XSD-based tooling supports vast changes in XML content - with the right XSD.

Getting Data Into Excel

- **Existing options aren't pretty**
  - Database access requires views onto a database; users often put massive load on the database inadvertently.
- **VBA makes it super easy to populate a sheet from XML over HTTP**
  - http://msdn.microsoft.com/en-us/library/aa203724.aspx

```
Dim xmp as XmlMap
Dim xp as XPath

set xmp = Application.Workbooks(1).XmlMaps.Add(URL)

set xp = ActiveSheet.Range("B1").XPath
xp.SetValue xmp, "/Root/RepeatingElement/Element1", , True
set xp = ActiveSheet.Range("C1").XPath
xp.SetValue xmp, "/Root/RepeatingElement/Element2", , True
```

Configuration Changes

- **Leverage HATEOAS**
  - Clients have a single entry point that defines how it interacts with the rest of the system
  - Change that point and well-behaved clients will automatically follow the configuration change
  - Puts configuration changes in the hands of the data producers!
  - Can even selectively deliver navigation content based on client
- **Use Load Balancers to shield clients from nodes going up or down**
  - Particularly useful for "well-known" internal URLs
  - Leverage Internet-scale support for HTTP

Eliminate Unnecessary Polling

- **Do you really need to?**
  - If you have your caches set up properly, and are re-using keepalive HTTP connections, a single HEAD and GET are pretty fast.
- **Use the Message Oriented Middleware you already have**
  - When returning an entity, provide a reference to the middleware location that entity updates will be published on
  - Include the URL for the entity in the message headers for filters
- **Combine the two**
  - Have your edge caches listen to the asynchronous updates and invalidate the cache elements when new data is published

Handling Closed Systems

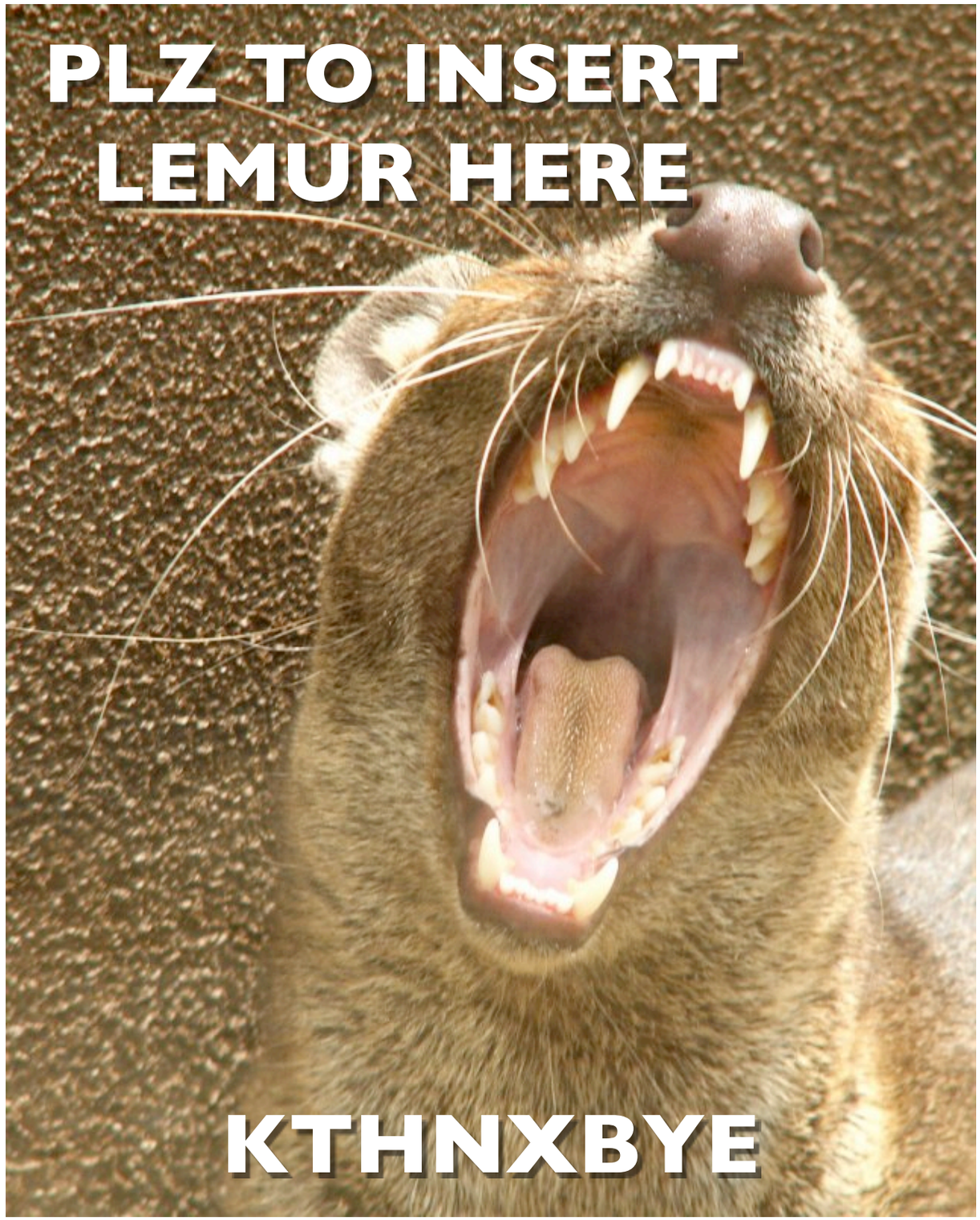
- **Some systems you can't change no matter what**
  - Legacy; packaged software; badly written; controlled by surly, angry people who don't read blogs or go to architecture conferences
  - 2-tier systems *everywhere* in Financial Services, particularly vendor-provided applications. How do you integrate with them?
- **Follow the SOA approach: Wrap it!**
  - Build edge gateways in the technology stack the closed system requires
  - Turns out you can reuse most of these, as closed systems have a few integration approaches
  - Where you can't reuse it, it's a system you **need** to defend yourself against!
- **URLs are under *your* control, not the wrappee's**

FOSSA Success Factors

- **Connecting all Front Office applications**
  - In-house developed Front Office and Back Office, and 3 different vendor-provided systems
- **Judged superior to existing approaches**
  - Load on databases and use of fiddly replication substantially reduced
  - Combination of tooling and hand editing made developers happy
  - Complicated data injection into Excel made traders happy
  - Caching improved access times even intra-system with little work
- **1 cross-system upgrades required**
  - Systems had to be upgraded (sometimes) to support FOSSA
  - Configuration changes and encoding tricks satisfied all point upgrades

Conclusion

- **Financial Services face different problems to other industries**
  - Integration latencies required, number of silos, amount of data to be integrated
  - We've got messaging (and how!)
- **Existing integration patterns don't work**
  - Too much labor, too link specific, too prone to failure on upgrades
- **RESTful integration FTW**
  - Constraints help silos work together
  - XML, HTTP, MOM all play nicely with Excel



Questions / Bonus LOLFossa

Note, DHH: NOT a LOLCat:
http://www.37signals.com/svn/posts/1614-no-more-lolcats-in-tech-presentation-plz