

Investment Banking Technology

Case Study: Reference Data Consolidation

Managing Interactions Across the Enterprise

Executive Director



J.P.Morgan

Introduction

- Introductions
- When Mergers strike!
- *Interaction* versus *Integration*
- Terms of Art - A Ubiquitous Language
- Critical Success Factors
 - Attention, Need, *Urgency*
- Regional Data Services™ (RDS)
- Governance/Roles
 - The one-stop data shop

Simple Interactions

“Everything should be made as simple as possible, but not simpler.” - Albert Einstein.

“The future state securities view should be driven front to back by a **single access point** for data.”

OSGi Alliance says:

- “Software complexity is increasing at an alarming rate.”
- “Today, software development largely consists of adapting existing functionality to perform in a new environment...”
- “A key issue is that **today's software environments focus on writing new software, instead of integrating existing software** into new systems. In reality, integrating existing code has become a large part of the work of software developers.”

Business Drivers

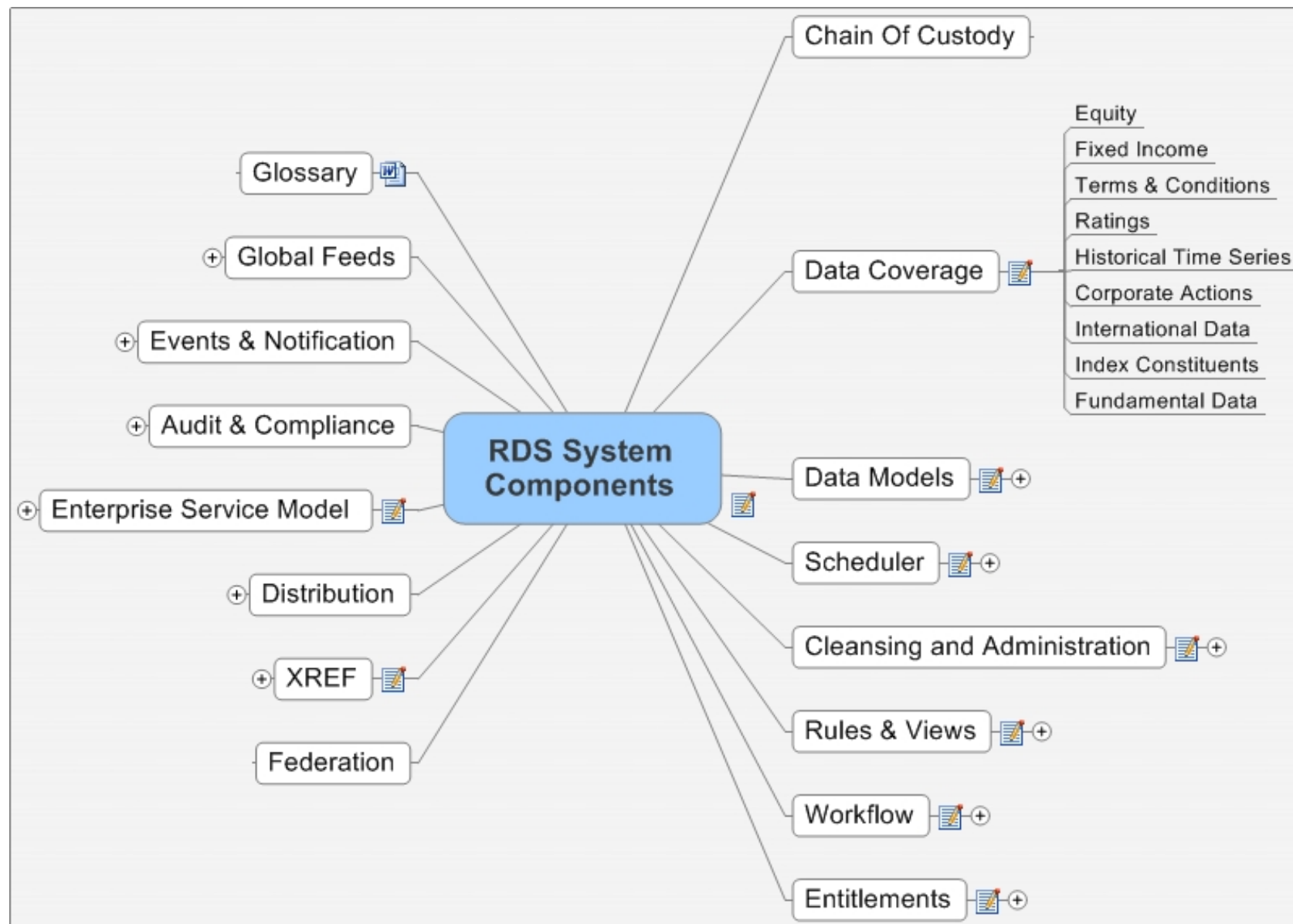
- A Merger!
- Vendor Contracts/Cost reduction - they cancelled what!?
- People/Skills Inventories - retention of key talent
- Systems of Record - where *should* you be getting this data?
- Feed Coverage - What do you leave on the cutting-room floor?
- Asset Classes Coverage - Linking the Silos
- Operational Pitfalls - reducing manual entry
- Events - What 'things' do you emit? - Capturing STP artifacts
- Shapes - What does your data look like?
- API's and Access Mechanisms

Build a Reference Architecture around what works today.

Guiding Principles

- Provide *professional management* of data assets - Chain Of Custody
- Create a dedicated reference data team - Data Stewardship
- Require *all data requests be driven through this team* - Control!
- Build components that are easy to deploy - Be Agile!
- Be *data-centric*, not database centric- Federate!
- Partner with Operations and create Governance around these processes - Automate!
- Replace high cost data acquisition methods with lower cost alternatives - Consolidate!
- Build systems so that you can *turn things off!* - Shelf Life
- Don't break production. Insist on *symmetrical environments*.
- Treat the cross reference construct as a first-class citizen - XREF

Regional Data Services™ (RDS)



The Big Pieces

■ The Loading Dock

- A file-based on boarding environment that 'mirrors' vendor data from FTP and Web Sites locally. Regionally deployed.

■ The Database

- A relational database that keeps and maintains historically all raw vendor data in its purest form. Not cleansed, not golden copy

■ The Engine/Interface

- A consumer-side data *HTTP interface* capable of being accessed via several end-points (telnet, Java, C#) with a flexible payload output
- Provides the capability to federate data from multiple data sources/repositories
- Supports both initial load and incremental updates.

■ API

- An easy binding to typical programming languages of choice (Java, C#, Groovy, Python...)

Terms Of Art

- API's
- Interfaces
- Data Providers
- *Implementations* versus API's
- Data Sources & Providers
- Chain-Of-Custody
- Loading & Cracking
- Federated Engines
- Data Models
 - Strong typing is for people with weak minds
 - Concrete types required for business objects
 - Business/Processing models
- Scrubbing (Golden *Copies*)
- Self-healing systems
- Event-driven systems
- Transports
- Wire Formats
- Payload
- The Loading Dock
- Shelf-Life
 - How do you turn things off?
- Rules
- Views & Shapes
- STP - Someone Typing Periodically

Terms of Art: Defined

- **Data Provider** - provides access to pools of various data across the enterprise
- **Federation Engine** - a DataProvider that can federate data from disparate systems
- **Views** - A projection of data into a certain 'shape', rectangular, or hierarchical
- **Authentication/Entitlements** - who are you and what are you allowed to see?
- **Wire Format** - XML, IBML, ASCII, EBCDIC, XDR, TDS, JSON
- **API** - Application *Programming* Interface. A high level construct to make things easy
- **Interface** - An access point to something. Generally lower level than an API
 - Ex. The *interface* is ftp, the *API* is WS_FTP. The *format* is mp3 the *API* is IPOD. A web service is an interface; generated code to access it is the API
- **Protocols** - FTP, Telnet, rlogin, ssh, Hayes modem, http, smtp, X.25, TCP/IP
- **Transports** - Socket, stream, queue, tin-can-and-string, sneaker-net, carrier pigeon
- **Namespaces** - 'coupon' versus 'cpn' versus 'sec_coupon'
- **Events** - Things that happen, not necessarily the buss that transports them
- **Metadata** - Data about data. *Discoverable!*
- **Chain of Custody** - When did my data arrive, where did it go, who has it now?
- **Discovery** - how do I find the data I am looking for?
- **Payload** - The thing *in* the message in some wire format, delivered over some transport

The RDS Engine™ - XML over HTTP

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV='http://schemas.xmlsoap.org/soap/envelope/'
>
  <SOAP-ENV:Header>
    <h:Header schemaVersion='3.0'>
      <h:Authentication>
        <h:UserID>ID_Demo</h:UserID>
        <h:Password>*****</h:Password>
      </h:Authentication>
    </h:Header>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:getViewData schemaVersion='2.0'>
      <m:ViewName>Equities</m:ViewName>
      <m:ViewDate>2009-05-07,2009-05-
08</m:ViewDate>
      <m:DataFormat>PIPE</m:DataFormat>
      <m:Keys>
        <m:Key>TICKER,JPM</m:Key>
      </m:Keys>
    </m:getViewData>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



```
<m:dataResponse >
  <s:statusType>SUCCESS</s:statusType>
  <m:viewData>
    <m:dataSet>
      <m:columns>
        <m:column name="TICKER"/>
        <m:column name="PRICE"/>
      </m:columns>
      <m:rows>
        <m:row>JPM|35.24</m:row>
        <m:row>JPM|38.94</m:row>
      </m:rows>
    </m:dataSet>
  </m:viewData>
</m:dataResponse>
```

A traditional (not quite!) web service over simple sockets (http).

Six lines of Java code to get reference data

```
1: Provider rdm = RDS.getProvider( "rds_dev", securityContext );
2: RecordReader equities = rdm.getData( new Request( "Equities" ) );
3: while ( equities.hasMoreRecords() ) {
4:     Record equity = equities.getNextRecord();
5:     writeEquity( equity.getString( "ticker" ), equity.getDouble( "price" ) );
6: }
```

Explanation of each line:

1. Lookup a DataProvider object by name (e.g., "rds_prod", "rds_qa", etc.)
2. Get the "Equities" data, which is returned as a RecordReader object.
3. Execute a while loop for each equity record.
4. Get the next equity record from the RecordReader.
5. Get two fields from the equity record and process them.

What to do next?

■ Take Advantage of “Significant Emotional Events”!

- “Be Pragmatic”, “Be Like Water” - John Bottega
- Leverage change to your advantage

■ Seek Ownership at Senior Management Levels

- This remains perhaps the most difficult aspect
- The fuss over ROI should be over, but it's not!
- Kill the Silo-mentality - we are all part of the solution

■ Build a quality Data Team

- You know the data better than they do...
- Retain a core team of veterans and train up the beginners
- Compensate appropriately

■ Keep It Simple - Don't get side-tracked by tools

- Use only enough of a particular tool to win