Financial Interpricing System for Fixed Income

Key Issues and Approach

Introduction

This document contains outline design principles for a Financial Interpricing System designed for a large Banking Environment, for use in the Fixed Income area.

Design of a complex Messaging System typically splits into two areas:

- Business Functionality – ensuring that the adaptors are in place to transfer messages in the correct format to the correct client, and
- Production Issues – ensuring that the design is reliable and scalable.

These notes concentrate on the Production Criteria, which can often make the difference between whether a system is workable or not.

Technical Design Issues

The System must be dynamic, to handle new business units, interoperable, to assist in the flow of information, and reliably real-time, to handle the constant change of the bank's financial status.

It should aim to provide a robust, global infrastructure which guarantees that information is passed in real-time between systems in a format which is understandable to all systems. At the same time, it must also accommodate Business changes (e.g. such as new financial instrument types, new systems, and replacement of existing systems).

At the message level, we need to define...

- an underlying transport mechanism
- a messaging layer
- a message structure definition
- Interfacing protocols.

Discussion of Financial Products is not included in this document. It is assumed that the purpose of this system is to forward the prices of Products from one part of the system to another, without necessarily needing to understand what the specific financial products actually are.

Interoperability and Simplicity

- Resist the temptation to go for a highly complex technical solution. In order to scale to the required throughput, it will be necessary to have a very simple “clean” interface between the different components.
- As a basic principle, the more processing that is required on a message, the longer the processing time. Since speed of delivery is essential, the messaging format processing should be as simple as possible.
- Latency time should be reduced in the design stage. Note that the key latencies are (1) Network Traffic, (2) JDBC and other Database connections, (3) Processing time for Parsing.

Scalability

- Able to add new “sinks” and “sources” very easily. The names etc. of sinks and sources should be configurable, and the specific message formats (adaptors) should be configurable.
- Message Transaction rate should be high enough for all possible throughput. Need to estimate rate, and to develop Load Test harness for the messaging system to prove throughput.
- Upscaling the system should be possible by just adding new hardware. Therefore, it should be possible to run the same application on high-end (multi-cpu) systems, just as easily as single-cpu low-end systems. This means that the application should be multi-threaded in all aspects.
- The Network impact of scaling up the number of clients should be considered – need to think about the advantages / disadvantages of multicasting, point-to-point etc.

Adaptability

- The system should use a standard messaging format which should be extendable, without having to rebuild adaptors. This implies the use of Fp XML as the key messaging format.

Reliability

- Messages must be guaranteed delivery, even if major parts of the infrastructure become unavailable (e.g. Failure of disk systems or servers). This means that all data must be in a persistant form as they are passed through the system.
- If part of the infrastructure fails, the system should be able to detect this, and re-route all activity(semi-) automatically. If this is not possible there should be a clear indication to the end-users that data is out-of-date and/or unavailable. It would not be acceptable for end-users to trade on out-of-date data.

Resilience

- Failure of some part of the infrastructure should not prevent messages from being forwarded. This means that the system needs to be “cluster aware”, that data can be picked up from a restart, from common cluster disks.
- Startup and Shutdown procedures should be clear. Systems should run as background jobs (“services” = no assumption that there is an OS console).

Speed of Operation and Serialisation

- We must be able to deliver messages at the correct speed. This should be verified by a Test Harness.
- It may be important that messages are received in the order which they are sent. If so, this implies some sort of time-stamping of messages, and serialisation routines.
- Network traffic should be optimised. This implies consideration about message size. Should all data be sent as a message, or should messages contain lookup references, which can be resolved at the receiving end?

Backup and Recovery

- It should be possible to backup the current persistant state at a given moment in time, in order to protect against failure of hardware (operating system, or disk, for example), and for the purpose of creating a test environment. This means that the system should have the ability of being “quiesed”, and all messages flushed, so that the environment is in a known state for backup.

- In the event of a total failure of a Datacenter, it should be possible to reconstruct the version of Operating System, Application software, and Configuration files very easily. Tests should be conducted to ensure that Backups can be recovered.

Audit Trail

- It should be possible to demonstrate that messages were sent to a specific target client, and that the target client received them. This implies some sort of audit log file.
- Archiving of history logs should be provided. Ensure that appropriate estimates are made for volumes of data required.

Security

- It should only be possible for messages to be received from trusted sources. This implies some mechanism for ensuring that, for example, it is not possible for developers to add messages to the live message queue.
- All endusers should be uniquely identified
- All messages should be uniquely identified, and their source should be verifiable.
- The architecture should be designed in such as way as to clearly differentiate the messages which are “internal” to the organisation, and those which may be passed externally, via firewalls, VPN etc.

Mobility

- Consider the possibility in future that data would be transferred outside the normal LAN / WAN environment.- e.g. Pricing information to PDAs with Bluetooth capability. Security implications also need to be considered. The transportation mechanism should be “light and portable”.

Compliance Issues

- Check whether any other compliance issues are not covered by the topics, above.

Terms and References:

FpXML: The Financial products Markup Language, based on XML, is an initiative to enable e-commerce activities for financial derivatives. The standard allows the electronic integration of a range of services, from electronic trading and confirmations to portfolio specification for risk analysis.

SwiftML: XML vocabulary for SWIFT messages.

ISO15022: One of the standard messaging formats used in SWIFTNet.

SWIFTNet: New Internet-like infrastructure messaging system created by SWIFT, to replace the old X25-based FIN.

FileAct: SWIFTNet product for secure & reliable transfer of any type of file - ideal for exchanging batches of structured financial messages, large reports, images, etc. used for bulk payments, securities value-added information, and reporting.

InterAct: SWIFTNet interactive transmission product – includes the ability to query another participant and receive an immediate response.

Technical References Suppliers:

Financial Information from FML eXchange.	http://www.fmlx.com/Financial_Information_Technology/index.shtml
ION Technologies Orbix:	http://www.iona.com/products/orbix.htm
IBM (MQSeries):	http://www.ibm.com/software/mqseries/
SunGard:	http://www.sungard.com/
TIBCO Messaging :	http://www.tibco.com/software/enterprise_backbone/messaging.jsp