### Order Management System (OMS)

*-A technical proposal by Steven Durko*

*“The race to zero is on and every microsecond of incremental latency is the difference between making a market and reading about it in tomorrow's paper. In your firm every microsecond counts, and is counted. Technologies that accelerate the movement of data create competitive advantage and drive increased market liquidity.” –Low Latency Engineer*

# Overview

An Order Management System (OMS) is the technology backbone of every competitive trading firm. The most familiar goal is to reduce round trip execution times as much as possible to capture opportunities in the market. Although it is important to reduce latency, there are other important factors that deserve consideration. This paper will explore all factors while recommending a design for an Order Management System intended to provide a competitive advantage to a trading firm.

# Assumptions

* Does not include budget or time constraints
* Will not focus on algorithmic logic, but will be limited to execution
* Will not focus on risk components in detail, but they will be mentioned
* Reduced latency, high frequency is implied (no use cases generated)
* Use Cases are “sunny day” (best case scenario), unless described otherwise

# Requirements

Functional requirements are necessary to capture the intended behavior of the system. For purposes of eliciting requirements for the proposed system, the following Use Cases have been created below:

Use Cases

|  |  |
| --- | --- |
| Basic   * Send an order to an execution venue * Modify an existing order * Cancel an existing order * Request status of an order * Cancel all orders * User should be able to see all orders by:   + Account   + Trader   + Instrument | Compliance   * Send an order from different accounts * Order does not exceed threshold of:   + Notional value   + Maximum quantity   + Risk thresholds (Delta/Vega)   + Average daily volume * Order has valid:   + Limit price   + Tick increment * Last 20 orders should not be identical * Firm should not trade with other accounts * System should be able to handle stock locates |
| By Functionality   * Manual orders * High frequency orders * Market making quotes * Spreads * COB orders * RFQ’s | By Product   * Trade stocks * Trade options * Trade futures * Trade fixed income * Trade spreads * Trade COB order |
| By OrderType   * Send an order with a type of:   + Market   + Limit   + Stop Limit   + Stop Loss | By TimeInForce   * Send an order with TIF of:   + DAY   + IOC   + FOK   + GTC/GTD   + MOO/MOC   + LOO/LOC |
| Session   * Sequence numbers should be unique and always incrementing on new orders * Messages should be sent in the order they are received * Client Order Ids must be unique for the day and session | Security   * Clients should be prompted to logon to the OMS with a username and password * Orders can only be created by trading entities (trader or algo) * Order should be rejected if placed outside trading hours |
| By Time   * Ability to send orders 23 hours 7 days per week * Order time should be the same regardless of location viewing it. (Sync to Coordinated Universal Time (UTC) | Algorithmic   * The system should be able to handle high volume of orders from strategies * A trader should be able to tell which strategy sent the order * A strategy should know which legs of a spread order were filled |
| Connection Failure   * Order Gateway disconnect from exchange should cancel all orders on exchange side * A 5 second gap with no heartbeats should trigger the OMS to automatically cancel all orders from a client session | Recovery   * If OMS goes down, we should be able to recover cleanly * If Order Router / Order Gateway goes down, it should recover messages lost from execution venue |
| Scalability   * The system should be able to handle the addition of new accounts dynamically * The system should be able to trade new products without shutting down * The OMS should be able to add more capacity by adding hardware | Reliability   * When sending an order, I should always receive a confirmation * Users should be able to view the connections are available and working correctly |

# Design

### Core Architecture Components

* Order Gateway
  + software library that connects to a particular execution venue and translates the communication protocol
* Order Router
  + Executable service that runs at least one to many Order Gateways
  + Receives order and forwards to execution venue b
* Order Manager
  + Maintains the state of all orders
  + Communicates with many Order Routers for order state information
  + Retransmits order state information to downstream clients
  + Incorporates the Order Compliance Library and checks orders against it
* Order Compliance Library
  + Software library that performs compliance and sanity checks on all incoming order flow before it is forwarded on to an execution venue
* Instrument Service
  + Responsible for knowing all the tradable instruments a firm requires
  + Uniquely identifies each instrument with a numeric id
  + Updates new products added intraday in real time
  + Maintains knowledge of expired products and deprecated products
  + \*Not shown in diagram
* Time Service
  + Synchronizes time for all client services
  + Broadcast updates to sync via UDP
  + Accuracy of time helps measure latency
  + \*Not shown in diagram

### System Diagram



Interfaces & Domain Objects

|  |  |
| --- | --- |
| Order   * Describes everything about an order necessary to send to an execution venue * Custom OrderId generated allows firm to track same Id through the life of the order * OrderId must be unique and is created from a combination of GUID + Date + Time + Account + Trader Acronym + sequence number. Always increments. |  |
| Execution Report   * Describes the typical information returned from an exchange venue regarding an order. * Status is primary to determining the how to deal with the order * Even though the exchange appoints its own Id, the same proprietary order Id that was generated on creation is used to identify the order |  |
| IObserver   * Contract used to notify “listeners” of data that an update has arrived. |  |
| IOrderGateway   * An interface design to capture the most basic functionality required for a gateway to an execution venue. All concrete Gateways will implement this basic functionality allow * No algorithmic logic here, very simple |  |
| IOrderCompliance   * A contract that incorporates compliance rules and sanity checks * Stock locate information available here as well * Expandable as rules are changed * Intended to be implemented as a library the Order Router can access, but it can be a standalone service (less performant) |  |
| IOrderRouter   * A simple interface designed to send and receive orders from the execution venue * Makes the working orders available to clients through use of API calls * No algorithmic logic here, very simple * Concretely implemented as a service with one or more Order Gateways |  |
| IOrderManager |  |

### How It All Fits Together

The Order Gateways are compiled as libraries and embedded into the Order Router services. The adapters are very simple and do not contain decision making logic. Responsibilities include communicating with the execution venue, authentication, and translating the proprietary order messages to the execution venue’s format and vice versa.

Execution venues can be exchanges, ECNS, Dark Pools, or third party brokers who route orders on your behalf.

The gateway will need to make a connection to the destination and communicate with the venue’s API for messages. Most of the time the FIX protocol is used to communicate which necessitates the need for a FIX Engine. The FIX engine would make the direct socket connection with the execution venue and handle the session management. The FIX engine library would be embedded in the Gateway Adapter to allow messages to be translated at this level in the software. It should be noted that each execution venue could be different in communication technology and format. As such, each new protocol would require a new Gateway Adapter.

The Order Routers will work in unison running the numerous gateways and forward exec report information up the application stack where the Order Manger comes in. The Order Manager (OM) subscribes to all data from each Order Router and maintains order state information for the entire firm. The OM forwards on the message upstream in the proprietary message format of the firm.

The Order Manager also acts as a liaison to the execution venues. All orders are sent to the OM regardless if sent by a trader or a strategy. Clients of the OM must logon with a username and password as a security measure. Communication with the Order Manager and clients is in TCP to preserve message order, provide error checking and delivery validation.

Once the OM receives an order, it immediately save the new order in it’s in memory database (Berkley DB) and begins the compliance checks. If the compliance checks pass, a new OrderId is assigned and the order is then routed over the wire to the appropriate Order Router connecting to the desired venue. The order goes through it’s typically lifecycle all over again until it is cancelled or filled.

*\*This is a standard implementation of an Order Management System many firms might implement. I have included an* [*alternative version of an OMS*](#_Appendix)*, which takes an even more aggressive approach to reducing latency and a preference towards algorithmic trading, located in the Appendix.*

# Performance

To reduce system latency, the entire stack must be taken into consideration when tuning for performance.

* Logical choices
  + Build or buy your FIX engine
    - Recommend purchase of Onyx FIX engine
  + Go DMA to venues (no middle man)
* Development choices
  + C++
    - Low level memory management allows for better performance
    - Compiles to native code
  + Use of Intel’s Thread Build Blocks (TBB)
    - Could be used to parallelize certain tasks
  + Pin threads for dedicated processing
  + Queue swapping for high frequency updates
    - Avoids context switching
  + Byte streams for communication protocol over xml or proto buffers/xml
    - More efficient then serialized data
    - Reduced marshalling costs
  + Combine order router and order gateway into a single process running on a single machine
    - Performance increased as data no longer has to travel over the communication layer
    - Order state / execution reported up to consolidated OM
      * [Reference Appendix 1](#_Appendix)
  + In memory database to save order info before going to order router
    - Low latency reads, non-blocking writes
    - Recommend [Berkley database](http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html)
  + Separate out accounts or limits (only consolidate for global view)
  + Stub orders
    - Pre create general order details and keep in cache for rapid sending of orders
* Hardware
  + Servers
    - Dell Power Edge M820
    - CPU
      * [Intel Xeon E7 family](http://www.intel.com/content/www/us/en/processors/xeon/xeon-processor-e7-family/intel-it-xeon-e7-4800-accelerating-silicon-design-brief.html) (Ivy Bridge -EX)
      * Increased performance over previous generations
      * Better cooling allows you options
        + Run with less power in data center
        + Over clock the cpu for more performance
  + Network Interface Cards
    - Solarflare
      * Kernel Bypass - Use [OpenOnLoad](http://www.solarflare.com/Content/UserFiles/Documents/Solarflare_OpenOnload_IntroPaper.pdf) technology to reduce latency by allowing application to directly access the network stack, bypassing the kernel.
* Operating System
  + Linux
    - Suse, Redhat or Ubuntu will be acceptable.
      * Since kernel bypass will be implemented using Solar Flare NIC cards, the choice becomes more preferential to a firm’s needs.
    - Thread affinity for a process
      * Reduces context switching and increasing performance by dedicating a thread to a process.
* Network
  + Colocation
    - Reduce the distant (latency) between execution venue matching engine and Order Router servers
    - Locations
      * Depends on products traded, but below are some examples of where New York based exchanges place their matching engines.
      * [Mahwah, NJ](http://www.wallstreetandtech.com/exchanges/NYSE-Datacenter/) – NYSE Euronext
      * [Weehawken, NJ](http://www.wallstreetandtech.com/slideshow/savvis-data-center) – Supporting data center
      * Carteret, NJ – Nasdaq OMX
  + 10gb Ethernet network to colocation facilities
    - Provide the speed and bandwidth necessary to accommodate transactions
  + [Arista Switches](http://www.aristanetworks.com/media/system/pdf/IBM_Arista_MellanoxWhitePaper.pdf)
    - Deterministic latency
    - [7150 Series](http://www.aristanetworks.com/en/products/7150-series)
    - High performance over reliability tradeoff
* Load Balancing
  + One Order Router/Gateway per execution venue recommended
  + It is possible to run multiple Order Gateways per Order Router if performance is not an issue.

# Order Compliance / Sanity Checks

Built into the Order Manager directly. Has performance hit, but necessary as order must be validated before sending to execution venues.

* Compliance
  + Recommend reviewing all required rules and the interpretations with firm Compliance Officer or legal counsel before accepting as solution.
  + Management of OMS Source Code
    - Usage of source control to ascertain changes to code and when they took place
    - Code reviews to help reduce defects to the system
    - Unit tests
  + Maintain order logs online for three years and an additional 4 years on tape
  + [Rule 15c-3-5](http://www.sec.gov/rules/final/2010/34-63241-secg.htm) compliant – Risk Controls
    - Trade size thresholds
    - Limit breaches (risk)
    - Prevent crossing of orders
    - Notional Value checks for derivatives
    - Order exceeds daily average volume
* Sanity Checks
  + when the market opens/closes
  + price increments (ticks) valid
  + buying an instrument above market
  + selling an instrument below market

# Handling Outages

Downtime and data loss are unacceptable events when dealing with execution systems. It is critical to report delivery or loss so actions can be taken and systems return to normal in an expedited fashion.

* General
  + Performed for both Failover and Recovery
  + Alert clients when connection is down
  + Cancel all orders on loss of top connection with execution venue
    - Setup with each destination
* Failover
  + Dual lines (primary & backup)
    - 10gb Ethernet
    - Dirty and clean lines available based on cost
  + Hot backup that listens to all the data, but takes no action.
    - If heartbeat not received in 5 seconds (configurable), it assumes primary responsibilities and starts to persist to the database and forwards on orders. Also sends a shutdown command to the primary instance to make sure duping does not occur in case primary comes back up.
* Recovery
  + Order Gateway to execution venue
    - Gaps in messages received are detected via sequence numbers.
    - Missing messages are requested on startup and replayed resulting in fully recovered order state
  + Order Router to Order Manager
    - Recommend using a 3rd party network communication software capable of persisting messages from Order Router.
      * Messages from Order Router are persisted until the OM acknowledges it has been received.
      * Upon OM acknowledging receipt of message, it can be removed from network layer data store
    - Missing messages are automatically replayed to Order Manager as it restarts resulting in fully recovered order state
    - Recommend 3rd party network communication software
      * [Solace Systems](http://solacesystems.com/) guaranteed messaging solution
  + OrderManager can recover from in memory database (Berkley Database)
    - Based on sequence numbers
  + Order manager and position service

# Testing

Modifying the OMS components will be common place to keep up with changing protocols, additional features, and bug fixes. As such, the need for testing is paramount to saving the firm from costly errors and fines without putting actual dollars at risk.

* Exchange simulator
  + Mimics different execution venues communication layer and protocols
  + Capable of providing playback of an actual day recorded order flow
    - Useful for back testing strategies
  + Ability to stress test systems by use of throttling
  + Useful for certification with exchanges
* Test Applications
  + GUI – allows manual input of orders, modifies, and cancels
  + Strategy – allows for high frequency load based on product/account
* Staging Environment
  + An environment for developing and testing new code, completely isolated from the Production environment
  + Hardware specifications less powerful then production, but allow for adequate testing and validation of candidate.
  + Connects to exchange simulator for controlled responses
  + Functional Testing occurs here
  + Execution venue acceptance testing occurs here
* Beta Environment
  + A controlled production environment
  + Hardware specifications, connections, and settings matching production systems
  + Limited to single venue/range of instruments/volume thresholds
  + Can be used by a small group of traders or a single algorithm
  + User Acceptance testing occurs here
* Unit Tests
  + Test the most basic of functionality (atomic)
  + Measureable code coverage numbers give you certainty when making release decisions
* Functional Tests
  + Integrated tests with all components of the system to verify behavior is as expected
* “Rainy Day” Use Cases
  + Order Compliance threshold breaches
  + Sanity Checks breaches
  + Risk Control breaches
  + Disconnects
    - Test Failover scenarios

# Monitoring

Rather than waiting for problems to occur, it is important to be proactive regarding system health. This provides confidence in the system and reduces costly outages. Below are recommended tool for monitoring the Order Management System.

* OMS Admin app
  + Custom – created in house
  + Real-time monitoring of
    - Orders
    - Fills
    - Cancel / Rejects
  + View order state
  + Ability to cancel orders
    - By Order ID
    - By Account
    - By Execution Venue
  + Connection status
    - Are we sending/receiving heartbeats?
* Server Management Software
  + Monitor health of servers running OMS
    - Immediate threat/failure detection with alerting
      * Memory
        + RAM
        + Disc
        + Virtual Memory
      * CPU
      * IO
      * Power Consumption
      * Temperature
  + Can embed client library into applications to help measure performance
    - Messages/sec
    - Queue sizes
  + Real time event coordination
  + Recommend [Nagios](http://www.nagios.org/about/overview/) or [Solar Winds](http://www.solarwinds.com/)
* Network Analyzer
  + Monitor congestion of network traffic in real time
  + Alerts
    - Impending bottles next arise
    - Message traffic threshold is breached
  + Ability to drill down into the data instead of broad updates
  + Recommend
    - [Latency Analyzer](http://www.aristanetworks.com/media/system/pdf/TechBulletins/Lanz.pdf) (Lanz) by Arista
    - Solar Winds

Appendix

# Figure 1 – Optimized Design



* This optimized system collapses the Order Gateway & Order Manager into a single library that runs in a process. The Order Router and communication hops have been completely removed. This approach is more stream lined for execution and provides less execution steps, and contention for resources.
* We now see an algorithmic trading server accessing market data and execution services directly. The algo server is much more efficient as it now contains everything it needs to trade and make decisions quickly. The algorithmic trading server runs its own compliance rules and is only concerned with the data it needs to work with, not a firm wide view.
* Firm wide view order state and position can be made available, but now the work has passed to downstream servers to collect order state and positions from these instances.

### Peripheral Components

### Figure 2 – Position Service Interfaces



* IPositionService
  + Maintains holds state for the entire firm’s positions in real time
  + Listens for “Fill” information from all possible sources to provide a complete picture of the firm’s current state
  + Handles “Position Adjustments” to correct firm positions on demand
  + Provides Position data to downstream clients by request and pushes updates to subscribers
  + Critical system feeds algorithmic trading servers, automated hedgers, firm risk infrastructure and traders on the desk
* Position
  + A Domain object that succinctly describes the relationship between an instrument, account, and quantity of the instrument.

### Figure 3 – Market Data Service Interfaces



* IDataFeedHandler
  + A contract designed to capture the basic functionality needed to obtain and utilize market data from an exchange
  + Provide Snapshot and Update functionality to keep the client updated with the latest information
  + Knows the implementation details for each venue it will emulate
* IMarketDataService
  + Represents the contract for an executing service which will reference a specific implementation of the IDataFeedhandler
  + Higher level functionality, the Market Data Service will know what to ask for, but will not know the details of how it receives the data
  + Able to build a custom NBBO based on data it receives from multiple exchanges
    - Ability to add or remove exchange from the feed
  + Offers ability to get Depth of Book updates (Level 2 quotes)
    - New Order Book object would accommodate this request (\*Not shown\*)
* Quote
  + A Domain object encompassing the entire definition of a quote from an exchange.
  + A lighter version should be used for passing data around the wire
    - \*Unless you use Google Protocol Buffers which will not populate the data unless it exists, saving you the cost of the entire message size.
  + Quote object does not