Distributed Objects in a Server/Server Environment for Futures and Securities

Architectural Paper

3/8/98

James J. Geldermann
(847)686-8771
jim@geldermann.com

Table of Contents

Overview

The futures/securities execution and trading community is faced with the need to develop a data communications infrastructure that will support the transmission of time and mission critical information across programmatic and physical boundaries..

Todate solution architecture has been limited by restrictions imposed by hardware platforms, monolithic and introverted programming designs, and transport overhead.

The execution and trading community has a vested interest in expanding closed enterprises into a secure ubiquitous environment. For the clients of these firms' IT departments to be productive, information must be able to cross boundaries in a real-time, secure manner.

IT departments are further faced with the problem of supporting viable legacy systems in their current mission critical roles, while attempting to extend these systems into roles that they where neither design for or capable of performing.

These issues can be met by a distributed computing infrastructure, an underlying architecture that can securely cross programmatic and physical boundaries, bridging multiple language and hardware barriers.

This white paper will outline these boundaries and propose a non-linear, agile server/server solution that will handle these issues.

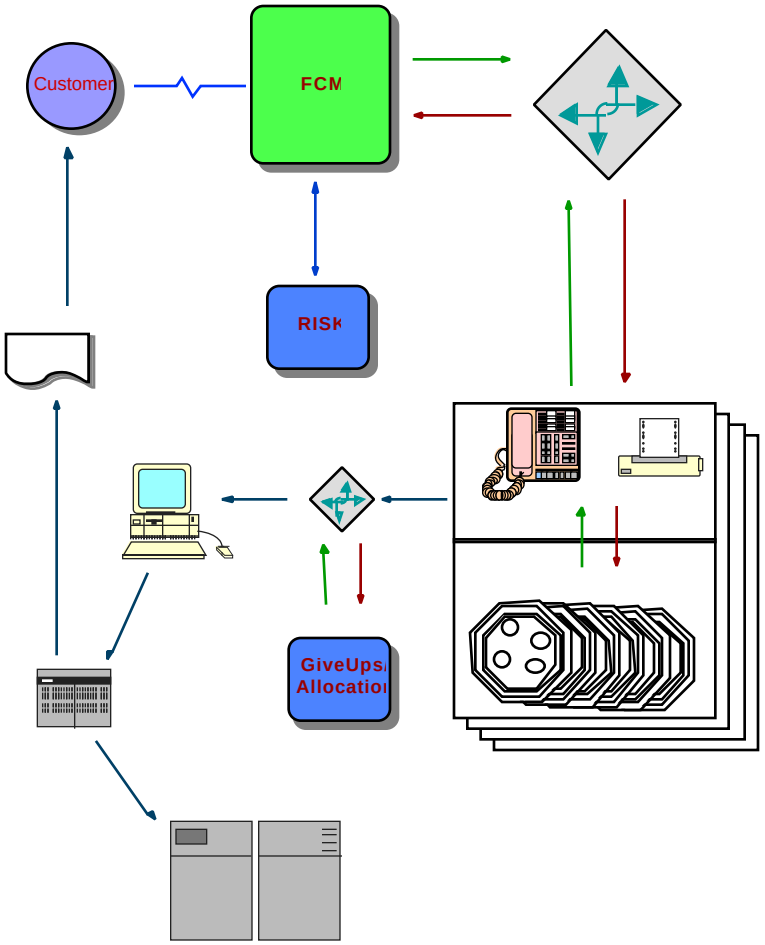
Programmatic and Physical Boundaries

Overview

Clearing, execution, and trading companies face difficult integration problems from both within and outside of their enterprises. Currently few clearing, backoffice, order management and trading systems communicate in a real time mode. Most attempts at integration have focused on file transfer schemes, clumsy APIs, and klugey screen scrapings. These solutions have been implemented because these systems support a monolithic design and are platform dependent. Solutions may also have closed transport protocols, unique data structures, dedicated hardware, and/or large maintenance overhead which further restrict their ability to interoperate.

A typical order management solution will incorporate an order delivery system, a backoffice system with a clearing house interface, and a report delivery system. In most cases these systems do not interoperate and the most common way data crosses these physical boundaries is via keyboard input.

An order generated by a customer may travel over voice to a broker or clerk who enters the order into the order delivery system. These systems may entail the clerk writing the order out on an order pad or entering the order into a terminal. In either case, a hard copy is picked up by a runner, either



from the clerk or from a printer, and delivered to the executing broker who files the order in his deck until market conditions require the order to be filled. Until the order is filled or canceled the customer may request status of the order from broker or wish to modify the order, via the same delivery system.

Once the order is filled a runner will pick up the order from the broker and return it to the desk. The clerk will check the order in, making sure that all information supplied by the broker, via a handwritten note, is complete and correct.

The order is then verbally reported to the customer and handed off to key-punching. At key-punching the broker's handwriting is deciphered and entered into the backoffice system. A preliminary run is batched to check for key-punching errors and at the end of the day a batch file is transmitted to the clearing house.

The clearing house will match the contra trades and return an exceptions report for trades that do not have a contra trade. These unmatched trades (out-trades) are checked for incorrect contra-brokers and/or houses, quantity discrepancies and execution price discrepancies.

Trades that can be reconciled are resubmitted to the clearing house and an edit report is submitted to the backoffice. All trades that cannot be reconciled will be delivered to the executing broker in the morning to be reconciled by the opening of the market.

The backoffice system will run a batch process at night to deliver to the customers a copy of all trade confirmations, open trade positions and cash positions. These reports are delivered to the risk management department (margin department), the account executive and the customer. Customers may receive their statements via mail, fax, and or electronic delivery.

The customer will then reconcile the report with their internal records and report any discrepancies to the firm.

Programmatic Boundaries

The programmatic boundaries experienced within the enterprise are extensive and are compounded by application languages and design. These applications utilize platform specific languages and are data-centric. The data center's mission is to first gather and validate data, second store data, third process the data, and forth report the data.

Legacy applications are designed to be self-contained and introverted. They were designed pre-ethernet and typically utilize SNA as the network protocol and IP through IP gateway controllers. Users are limited to inputting data and query for records. Processing is handled either by *Transaction Processing Monitors* (TP Monitors) or by scheduled batch processing. Client software provides a limited window into the system and are considered "dumb", meaning that they do not do any pre or post processing of data.

TP Monitors manage multiple processes within data centers by breaking complex applications into chunks of code called transactions. TP Monitors have the ability to get multiple transactions to run in unison. TP Monitors interject themselves between the users and the data center and can manage

the data center's processes by managing work load, route transactions across the WAN, monitor and adjust execution and restart applications after failure.

Legacy solutions are normally synchronous, which means that all parties are locked until all agree that the transaction has been successfully completed. Security concerns aside, it is not advisable to give up control of your data center's processing cycles to an untrusted member of a transaction. Further, transaction latency is increased and load bandwidth is decreased because of the physics involved with untrusted remote connections.

Legacy systems have limited communication tools that limit interoperability with PC clients. As an example on the Wintel platform there are no tools that allow developers to read the network card data buffer, and on the Macintosh there is only one. There is only one development language (HALAPI) that allows developer to talk to a 3270 or 5250 controller. There are several 3270 and 5250 OLE/COM objects, but they only allow the developer control of the VT session. Finally there are several proprietary legacy database tools, but they are also introverted and do not allow the developer to subsume them into a solution.

In the past few years SQL servers have been viewed as the answer to move legacy systems to client/server. Unfortunately SQL servers present severe limitations to secure enterprise to enterprise communications. SQL is too introverted and data-centric to provide the backbone to a transaction intensive solution.

SQL from different vendors does not interoperate. Each vendor employs different ODBC tools and a developer needs to have access to the enterprise's schema in order to pass data. That would mean that either a neutral SQL server, deployed and maintained by a third party, would act as an intermediary between enterprises or enterprises would have to grant access to each other.

SQL does not handle processes efficiently. It relies on non-standard extensions and procedural languages. A SQL server hides processes either within stored procedures or in front-end tools. Transactional applications require the use of triggers, which are basically polled batch applications that run continuously without regard to need.

Workflow/Groupware provides a good model of what a solution should be. Workflow is used within groupware to automatically route information to the people who most need it. It provides a good mission critical foundation for the business process, but it fails in the areas of time critical delivery, interoperability with legacy systems, and experiences an exponential decrease in performance as concurrent clients are logged on.

Groupware's middleware foundation is proprietary and the developer is forced to work within its framework. Groupware does not subsume legacy systems and does not scale well in the world of data rich transactions.

Physical Boundaries

Physical boundaries are both intentional and unintentional. Security and data integrity require enterprises to erect barriers that restrict access to applications and data. Data center managers take their mission as keeper of

the corporate jewels very seriously. People who live in halogen rooms don't let just anyone in, you must be one of the chosen few and you will require a "user name" and a secret password to gain access.

Of all of the barriers to overcome in deploying a solution that interoperates with extra-enterprise entities, the security committee is the most daunting. The security committee requires that any contact with untrusted entities follow strict protocols to ensure the integrity of the data center. Managers fear the breaching of their security systems will compromise data resources through corruption and/or replication.

For the security committee to approve contact with untrusted entities a architecture needs to be employed that allows the free exchange of messages without allowing direct access to the data center. It is not enough to erect firewalls, proxy-servers or router filters. With enough time and information these systems can be breached. The only truly safe way to protect the data center is to physically disconnect it from the rest of the world. This does not mean that the data center can not interoperate with untrusted entities, but it does mean that a system needs to be deployed that physically bridges intra and inter networks.

Unintentional physical boundaries include hardware incompatibilities, network protocols, and network configurations. IT managers have attempted to impose standards within their domains. They have experienced good success within work groups, but limited success through out the enterprise. Data centers typically require mainframe and mini computers because their missions employ schedules and large report generation capabilities. Trading centers have relied on the computing power of RISC architecture because of the processing speed required for complex algorithms that need to be displayed instantaneously. Sales and support areas require word processing and spread sheet platforms.

The diversity of these hardware platforms limits the amount of interaction and the means available to pass data in real time. Typically APIs and CTCIs have been developed to allow systems to cross physical boundaries. APIs and CTCIs require an in-depth knowledge of two platforms and systems. It requires a coordinated effort on both teams to discover, plan, test, and implement a solution. Each system is held hostage to the other's maintenance and upgrade schedules.

Intergalactic OrderNet™

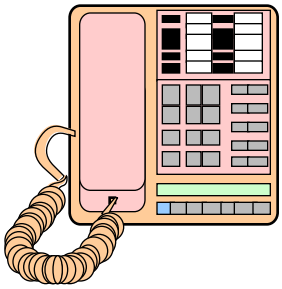
The Intergalactic OrderNet™ (ION) is a non-linear peer-to-peer secure communications architecture. It supports low bandwidth connections (9.6 kbps) to high bandwidth networks (100+ mbs). It is designed upon the Internet Protocol (IP) for the transport layer and will support either TCP or UDP at the control layer.

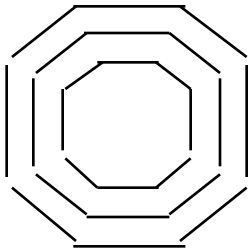
ION incorporates a packet based sessionless messaging protocol that substantially reduces network traffic. Because most order messages can be stuffed into one IP packet there is no need to open and maintain a traditional client/server dedicated session.

ION's architecture consists of a physical topography, software components, and metadata structures. Each of these elements are open by design, allowing any qualified exchange member to become a member of ION.

ION is designed to map directly to the current business practices members have employed for over 150 years in moving orders inter- and intra-domain. Although it is a subset of a holistic order management solution, its design allows easy integration into existing, non-ION solutions.

The mission of ION, using the exchange floor metaphor, is to supplant and/or augment the role of the runner. To best understand ION an understanding of the role of the runner is required.





The runner's role is to carry orders and messages between the FCM's desk and the pit broker. The runner needs to know which broker has been engaged to manage a particular type of order and the location of that broker. In the real world the runner operates in a synchronous mode, meaning that once the runner leaves the desk the FCM loses control of his actions until he returns. For instance the desk clerk may give the runner two orders (Cancel/Replace Sell 10 November Soybeans and 100 March/July Corn), check the status of an order (Buy 10 December Wheat), and get a quote (November Crush). In this case the messages are addressed by using message type(buy, sell, stop, spread, status, quote), commodity (soybeans, corn, wheat, oil, meal), and month.. Typically the runner will look at the message and map the address attributes to a particular broker's name and deliver the message to that physical location. If the runner cannot find the broker he may know the name of the an alternate broker, he may ask if

anyone knows the location of the broker, or he may go back to the desk and inform them that the broker is off-line.

It is the responsibility of the sender to notify the runner of any exceptions. For instance a customer may want his orders directed to a particular broker. In that case the account number of the customer is another attribute of the address and either the runner will know the correct address or the desk clerk will inform him.

To complete these tasks the runner only needs to know the location of the brokers and how to get there. It is the responsibility of the runner to determine the best way to arrive at his destination and he is responsible in seeing that the broker receives and accepts the message. The message the runner carries is formatted in a way that both the FCM and the broker can read. This allows each to translate the message into a format each has independently devised to manage their respective businesses. The runner's task is completed upon the successful delivery of the message.

In the real world the FCM's runner also acts as a messenger for the broker. The runner may wait for a reply to the message he has delivered or the broker may drop the order into the return queue (the pit's steps). The runner may notice the order sitting on the floor as a message for his FCM and will pick it up and deliver it to the desk clerk.

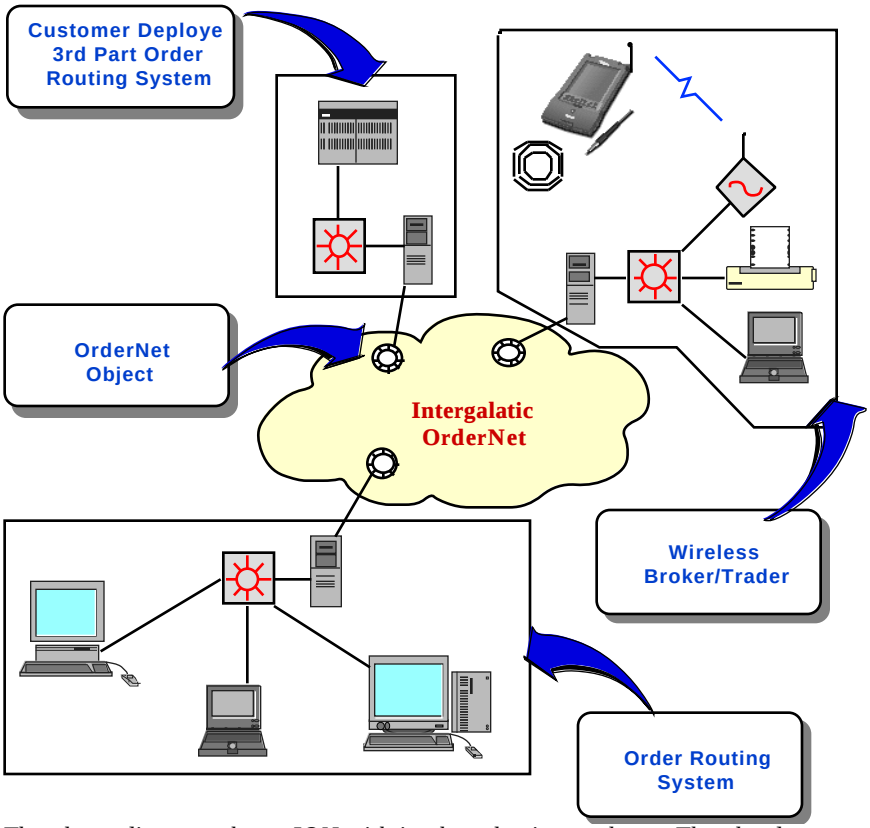
To complete the metaphor ION's topography emulates the FCM's runner, the software component emulates the desk clerk and the broker's clerk, and the metadata emulates the order or message.

Physical Topography

ION allows its members to securely link private enterprises and pass transaction based datagrams within a peer-to-peer sessionless connection. The architecture relies upon three physical transport zones, corporate enterprise networks, public/private networks, and exchange networks. For the most part these zones are in place, but not connected. ION requires that its members only have a point of presence (POP) on the exchange's ethernet, token ring, and/or wireless network. ION uses off the self network components (routers, DSU/CSU, hubs, switches, etc.) to route messages between members.

Since ION uses packet based messaging, as opposed to session based messaging, it is able to maintain message latency within a .3 and 1 second envelope.

A typical installation for an off exchange enterprise would consist of a ION server (RISC or Pentium), a dedicated router and a CSU/DSU. The modem



The above diagram shows ION with its three basic members. The cloud represents the exchange's network and the ION Object is the member's POP.

is connected to either a dedicated line to the exchange's DSU/CSU or to a private of public network that has a secure POP at the exchange.

A dedicated router is used to physically separate the member's network from ION. The ION server is used as a physical barrier between ION and

the member's network. The physical barrier is constructed by the use of two network cards registered on two non-logical IP network segments. This feature prevents untrusted clients from routing onto the members trusted side of the network. The ION server uses its ION Object to bridge packets between the exchange's and members networks.

Software Components

The ION software component consists of a member application framework containing an enterprise object, for enterprise session management and communication, and a CORBA/IIOP compliant object, called the ION Object. The ION software component has three core missions, 1) Secure communication with other ION Objects; 2) Secure communication with member enterprise applications and services; 3) Data mapping between ION Objects and member enterprise data structures.

Secure Communications with other ION Objects

ION Objects have three session states, 1) Unregistered; 2) Registering; 3) Registered.

An ION Object can only communicate with other authenticated ION Objects. For an ION Object to be authenticated it must first be pre-registered with the ION Registration Service (ION/RS). This service is a central depository of ION Object and their properties. It is used to initially authenticate a member's ION Objects and serve as a resource for other authenticated ION Objects to access for available services. It is accessed by ION Objects to authenticate other ION Objects during the challenge/acknowledgment phase of session registration. For an ION Object to acquire an unregistered state it first must be approved and registered with ION/RS. Registration with the ION/RS requires that the member supply the registry with the IP and MAC address of the member's ION server.

The ION/RS does not monitor or route traffic, it only acts as an authentication service and a registry database. Once a session has been registered between two ION Objects the ION/RS is not accessed during the normal course of business.

The ION/RS eliminates the need for peers to exchange object names, passwords, and locations. Once an ION Object has been registered all authentication issues are handled by the service.

An ION Object enters a state of registering by contacting another ION Object and requests a connection. The contra ION Object simultaneously contacts the ION/RS for authentication and sends the originating ION Object a contra request for connection which requires the originating ION Object to request authentication for the contra object from the ION/RS. If both ION Object have been authenticated they exchange unique session ids. Once these steps have been completed the two ION Object change state to registered. The registered state of an ION Object is an asynchronous IP session allowing the object to perform multiple pre-emptive tasks.

An ION Object in a registered state is able to send a pre-authenticated message to a contra ION Object. An ION Object receives a message from within its enterprise, it reads the destination of the message, and checks to see if the destination is registered. If the destination is registered, the ION

Object inserts the destinations session id, address the message to the physical address of the contra ION Object, opens an IP session, transmits the message and closes the IP session.

The contra ION Object is notified that there is a packet in the IP buffer. It extracts the message from the buffer, authenticates the session id and processes the message.

The IP control protocol, either TCP or UDP, guarantees the delivery and integrity of the packet. This eliminates the application layer's need to send redundant messages of receipt.

Secure communication with member enterprise applications and services

The ION Object provides a bridge between the enterprise's applications and services with the ION. A custom application is constructed that encapsulates the ION Object and code to allow the object to open and maintain sessions with the enterprise's applications and services.

For instance the member may have an existing order management system that is widely distributed and/or accepted within the member's enterprise. Instead of building a new system to route orders to the exchange, a simple application is built that acts as either a headless client on the order management system or as an integrated component. The software provides three critical functions; 1) Secure client session or interface with an enterprise order management system; 2) Data translation between the ION Object's metadata structure and the order management systems data structure; 3) Communication with other ION Objects on ION by encapsulating the ION Object.

The design allows IT departments to encapsulate one object for ION communication, thus focusing on communications and maintenance sessions with their known systems.

Since the ION Object handles all communication and security with other ION Objects, the integration of existing and future order management systems is simplified. Another benefit to this architecture is that upgrades can be managed independently of each other. It does not matter to either parties if an upgrade occurs, as long as basic method names, return values and data structures remain the same.

Data mapping between ION Objects and member enterprise data structures

The ION Object supports mapping between ION Metadata data types and enterprise data structures. This is an important feature because no two enterprises support the exact same data structure. An enterprise may use simple structures like strings with deliminators or more complex structures like data frames, maps, arrays, array of arrays, frames of frames, array of frames, etc.

A metadata data type model is used because parsing consumes cycles and is error prone. With a metadata model values are assigned to slots within the properties frame. For instance when a function needs to read the commodity symbol it only has to point to commodity symbol slot and read the symbol. This allows dynamic addressing of messages in a real time

mode. Which means that messages can be routed and rerouted quickly with little impact on latency.

Metadata Structures

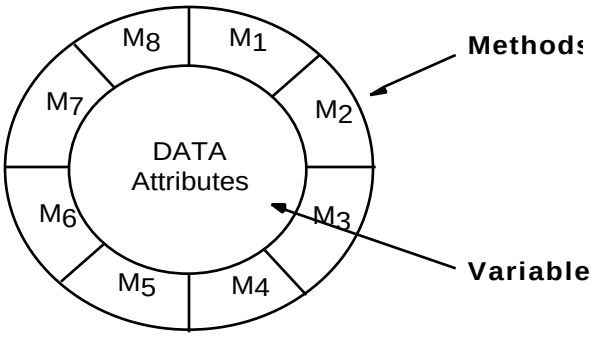
The incorporation of metadata makes an ION object self-describing and introspective. This allows the development teams to concentrate on the use of data as opposed on how to access data. Metadata is a critical component in any agile server/server solution. An agile system is self-describing, dynamic, and reconfigurable. ION's metadata structure is used to consistently describe all available ION services and data.

Server's developed independently of each other may register ION sessions with each other and securely pass data. The ION object handles all transactions within the ION cloud, relieving the developer of the task of writing communications session, matching data structures, and knowing another server's function calls. The ION object provides a secure and seamless interface from one enterprise to another.

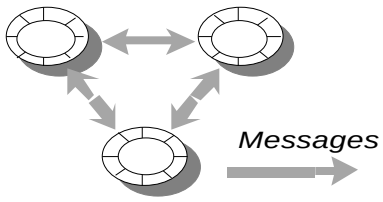
This also allows each enterprise to build their solutions to meet their business needs, instead of buying someone else's concept of how a FCM should run their order management solution.

Distributed Objects

As stated by more than one writer, object-orientated designs and solutions are the software equivalent of the Industrial Revolution. In the same way that modern factories assemble products out of prefabricated components rather than manufacturing every component from scratch, object-orientation allows programmers to build complex software by reusing software components called objects.



An object is a self-contained software package that encapsulates a collection of data and procedures. Objects can either act as stand alone applets or be encapsulated within larger packages. In either case objects act in the same way. An entity outside of the object sends a message to the object. The message will contain the name of the procedure that the object will act on and may contain data that will have an impact on the outcome of the procedure. The object will return some result of the procedure to the sender. The result may be either an error, the procedure did not execute properly, or a return value. These attributes allow software developers several advantages.



Reliability. By taking a large complex software structure and breaking it down into small, self-contained, manageable objects, object-orientated design ensures that changes in one portion of a solution will not have a cascading effect through out the solution.

The development cycle of an object is compact and can be accomplished independent of the solution. Because objects are mission specific they can be built, tested and certified outside of the solution's development team, thus the reliability of the overall software solution increases.

Maintainability. Objects are modular and constitute a small portion of the overall solution. Object teams need only know the functionality of the object as opposed to the business processes that incorporate the object. Object teams can upgrade an object and solution teams can change the implementation of an object without creating negative consequences throughout the solution.

Reuse. Objects are portable and are not application dependent, thus objects can be reused within and outside of solutions. One object can be used in many solutions. Through subclassing, specialized objects can be created by adding code unique to the new object. The new object inherits structure and functionality from the superclass, thus reducing coding and increasing reliability.