Screen Based Trading (SBT) Application Architecture

Chicago Board Options Exchange

20 May 2002
The Chicago Board Options Exchange

Change History:

Version	Date	Author	Reason for Change
0.1	06/29/98	TRC	Original document.
0.4	8/5/98	Khiem Luc Luis Benavides	Updated for changes from the application architecture discussion on July 98
0.5	8/14/98	Khiem Luc Luis Benavides	Updated for changes from the work-in-progress of application architecture review on 08/06/98
0.6	8/24/98	Khiem Luc Luis Benavides	Updated for changes from the work-in-progress of application architecture review on 08/17/98
0.7	9/2/98	Odalys Castro	Edit document for readability
0.8	11/9/98	Odalys Castro	Update document
0.9	12/30/98	Mark Woyna	Updated to reflect increment 1 changes
0.9	1/28/99	Odalys Castro	Final formatting for increment 1 changes
1.0	4/26/99	Mark Woyna	Updated to reflect increment 2 changes
1.0	4/29/99	Mary Ellen Waszak	Formatting for increment 2 changes
1.1	7/21/99	Mark Woyna	Updating to reflect increment 3 changes
1.1	7/29/99	Odalys Castro	Formatted and updated to reflect increment 3 changes
1.2	11/5/99	Mark Woyna	Updating to reflect increment 4 changes
1.2	11/23/99	Odalys Castro	Update to reflect increment 4 changes
1.2	12/7/99	Odalys Castro	Final formatting for increment 4
1.3	5/3/00	Mark Woyna	Update to reflect increment 5 changes
1.3	5/3/00	Mark Woyna	Convert to Word 97 Format
1.3	6/26/00	Odalys Castro	Final formatting for increment 5
2.0	5/20/02	Mark Woyna	Update to reflect Version 2.0 changes

Document Cross Reference:

No.	Title	Reference

Document Tracking Information

File Name: SBTApplicationArchitecture

Document Reference: Screen Based Trading (SBT)
Application Architecture/Document/SBT_AppArch Draft09/CBOE1/SBT

Version: Version 2.0

Status: Draft

Sign-off

Authorized by Mark Novak/CBOE
Dave Marchand/TRC

Signature _____

Contents

Figures

Foreword

The application architecture is a blueprint for what the developers will build. Like a building blueprint, it outlines the major features both physical and functional. It details the type of hardware, if not the specific hardware. It outlines the dimensions of the architecture by specifying the boundaries. It builds-in any required security, capacity, performance, maintenance, and operational needs. It specifies all external and user interfaces to the application. The application architecture is the vision for what the built system should look like. Although the architecture may change slightly over time, it gives the designers and developers a view of what the final product should be.

Scope

The application architecture covers the high level view of the final system. It will provide the vision for other more detailed documents. It will cover both the logical, physical and functional aspects of the application. Although the application architecture is not enough for a developer to begin building the application, it is necessary for providing a common vision and determining synchronization points for multiple teams.

Audience

The system owner should read this document to ensure it captures the owner's vision for functionality and presentation. The system architect should read this document to ensure it adheres to the enterprise architectural vision and philosophy. The information architect should read this document to ensure it properly captures the business functionality. The infrastructure architect should read the document to ensure that the application uses the infrastructure items wherever possible. The technology architect should read this document to ensure the application properly uses the appropriate technologies. The system engineer should read this document to understand the vision and constraints placed on the system. The system engineer will reference this document when building the system requirements and performing system analysis. The designer should read this document to ensure his design captures the architect's intent. The developer should read this document to ensure the implementation realizes the architect's vision of the application.

Prerequisites

Readers should be familiar with the system architecture, the information architecture, the infrastructure architecture, and the technology architecture for this project.

Organization

1. The Introduction section contains background information, the business driver force and the technical driver force for the system, and an overview of the system context.
2. The Use Case View section provides the motive and describes the system needs in concrete term.
3. The Logical View section describes the static and dynamic aspect of the system in terms of packages, subsystems, components, and their responsibilities.
4. The Process View section represents the decomposition of components in terms of execution flows. Process View also represents the synchronization between the components and the availability and reliability of them at the process level.
5. The Deployment View section describes how the processes are deployed at the processor level. It discusses the scalability and fault tolerance features of the system. The hardware and software tools are also documented here.

Open Issues

The following open issues still exist within this document.

- None

Conventions

Special terms defined in the Glossary begin with uppercase.

Commands you need to key in and programming code appear in `Courier New`.

Menu items appear in Title Case. Menu paths follow in order Main Menu>Menu or Submenus>Command. For example, saving a file is File>Save.

Keys or keystroke combinations appear in **bold**.

Files and directory names appear in *italics*.

Notes look like:

Note

Description of the hint, problem, or precaution.

Related Documents

1. *Next Generation Computing*, The Technical Resource Connection, Inc.
2. *Screen-based Trading Functional Requirements*, Chicago Board Options Exchange (CBOE)
3. *SBT Software Requirements Specification*, Chicago Board Options Exchange (CBOE)
4. *SBT Analysis Model*, Chicago Board Options Exchange (CBOE)
5. *CBOE Integration Architecture*, Chicago Board Options Exchange (CBOE)
6. *CBOE Technology Architecture*, Chicago Board Options Exchange (CBOE)
7. *Directory Service Requirements*, Chicago Board Options Exchange (CBOE)
8. *System Management Requirements*, Chicago Board Options Exchange (CBOE)
9. *Log Service Requirements Document*, Chicago Board Options Exchange (CBOE)
10. *SBT Security Policy*, Chicago Board Options Exchange (CBOE)
11. *CBOE Security Requirements*, Chicago Board Options Exchange (CBOE)
12. *Messaging Service Requirements*, Chicago Board Options Exchange (CBOE)
13. *Foundation Framework Requirements*, Chicago Board Options Exchange (CBOE)
14. *CBOE Application Programming Interface*, Chicago Board Options Exchange (CBOE)

1. Introduction

This document addresses the application architecture of a Screen Based Trading (SBT) system that CBOE will deploy as part of its option trading business.

Note: As of Version 1.0, the SBT system has been renamed to CBOE*direct*™.

1.1 Business Drivers

SBT is necessary for CBOE to remain competitive in the industry. Other exchanges already offer some form of SBT to their member firms. If CBOE does not also provide SBT capabilities, it stands a great risk of losing its current members to other firms and will be unable to attract other firms to replace them. In order for CBOE to remain competitive, it must offer SBT capabilities by mid-summer of 1999.

1.2 Technology Drivers

With the growing popularity of the Internet, the desire for real-time trading from computer users has grown. SBT will provide an investor either direct connection to other investors or indirectly through a member firm to perform cyber-trading without the current need of a trading floor or brokers.

In order for CBOE to provide SBT capabilities by the target date, it must leverage the capabilities of the existing system. CBOE does not have the time to totally re-engineer the existing system into the long-term architecture of post-centricity. For the initial release, CBOE will "wrap" its legacy TPF based trading engine, to allow the SBT to communicate with the trading engine using the target distributed architecture.

1.3 Overview

The Screen Based Trading (SBT) system is an attempt of CBOE to provide a real-time trading system without the "open outcry" trading crowd. The system is currently used for an extended trading session (ETH) during the hours of 7:00 a.m. to 8:15 a.m. In the future, the system may be utilized for trading during the regular trading session, during which a product would be traded in the trading crowd or on the SBT system. SBT does not support trading a product on both sides at the same time. SBT information is not broadcast to the existing crowd trading system (TPF).

The SBT system will allow for the routing of orders to TPF for products traded in open outcry during regular trading hours (RTH). In this fashion, SBT provides an alternative interface to external systems wishing to enter orders into the exchange for processing.

CBOE*direct*™ Version 2.0 adds support for Single Stock Futures (SSF).

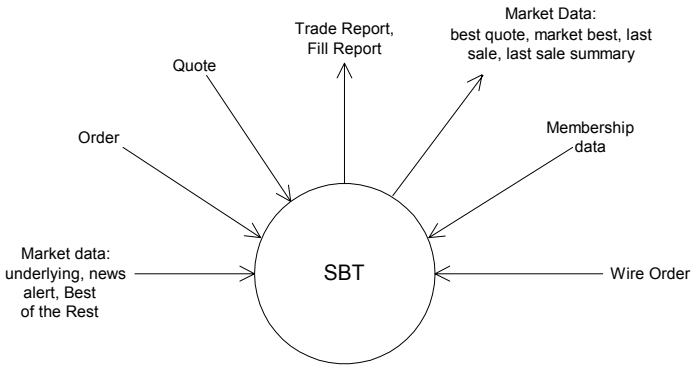


Figure 1 SBT System Context

The system processes wire orders (orders that are accepted using the existing CBOE trade infrastructure), SBT orders and quotes. It receives the supporting data: underlying price, news alert, Best of the Rest and membership information. It produces fill reports, cancel reports, trade reports and market data (best quote data, last sale summary). Due to the time frame of development and deployment of the system, it relies on the existing external systems capability to provide the described data and to disseminate the produced data to the proper authority.

2. Use Case View

This section provides an overview of the use cases that the SBT System must support. It is not intended to be a substitute for more detailed descriptions found in the Analysis Model (see use case documents in the Analysis Model), but only provides the background and context for the rest of the Application Architecture.

2.1 SBT User Logon

This use case models the required interactions between a Screen Based Trading User and the Trading Engine software when an SBT user logs in to an SBT terminal. Any actor (human or automated) must first be authenticated by the System before being allowed to use it. This use case (UC) comprises not only the initial actor authentication, but also the necessary initialization tasks in preparation for interacting with the System (i.e. downloading cached data, establishing a session, subscribing to receive events).

Actors using this use case:

- Trader
- Market-Maker
- Clearing Firm Broker
- Firm
- Class Display (Market Data)
- Operations/Help Desk
- BackOffice Application

2.2 SBT User Logout

This use case models the required interactions between a Screen Based Trading User and the Trading Engine software when an SBT user logs off an SBT terminal.

Logging off from the system involves disconnecting the session, unsubscribing from events, et cetera.

Actors using this use case:

- Trader
- Market-Maker
- Clearing Firm Broker
- Firm
- Class Display (Market Data)
- Operations/Help Desk
- BackOffice Application

2.3 Enter SBT Order

The System accepts orders from an actor for execution. The System accepts the following types of orders:

1) Limit
2) Market
3) All or none (AON)
4) Fill or kill (FOK)
5) Immediate or cancel (IOC)
6) Stop (STP)
7) Stop Limit (STP LIMIT)
8) Spread (Combination/Strategy)

Each type of order has a specific set of execution rules. Upon receiving an order, the system attempts to execute it according to the execution rules of its type. Depending on the execution rules and whether or not the order was (partially) filled, this UC may trigger (use) other use cases, including Display Fill Report, Disseminate Trade Report, Send RFQ, etc.

Actors using this use case:

- Trader
- Market-Maker
- Clearing Firm Broker

2.4 Cancel Order

An actor may cancel the unfilled portion of any of his/her resting orders.

Actors using this use case:

- Trader
- Market-Maker
- Clearing Firm Broker

2.5 Update Order

An actor may update an order, or cancel an order and replace it with another otherwise identical order but with a different price or increased quantity. An update request that includes a different price or an increased quantity results in a cancel/replace operation. All other changes to the order result in an update or revision of the order.

Actors using this use case:

- Trader
- Market-Maker
- Clearing Firm Broker

2.6 Hit the Bid / Take the Offer

An actor may electronically 'hit the bid' or 'take the offer' in a selected product market.

Actors using this use case:

- Trader
- Market-Maker
- Clearing Firm Broker

2. Display Fill Report

Upon filling an order (partially or completely), the System reports the fill to the originating actor.

Actors using this use case:

- Trader
- Market-Maker
- Clearing Firm Broker

2.8 Display Order Status

An actor may request a display of the current status of his/her filled and unfilled orders. The System will update the status of the orders in real time as long as the actor maintains the display active.

Actors using this use case:

- Trader
- Market-Maker
- Clearing Firm Broker

2.9 Display Market Best

An actor may request a display of the Market's Best Bids and Offers for a selected product. This information includes price and aggregate volume for the best bids and best offers, and an indication if the prices represent the National Best Bid and Offer (NBBO). The System will update the data in real time as long as the actor maintains the display active.

Actors using this use case:

- Trader
- Market-Maker
- Clearing Firm Broker

2.10 Display Last Sale

An actor may request a display of the last sale for a selected product. The System will update the figure in real time as long as the actor maintains the display active.

Actors using this use case:

- Trader
- Market-Maker
- Clearing Firm Broker

2.11 Display Underlying Market Data

An actor may request a display of the market data (status, last sales, and news alerts) for a selected product. The System will update the data in real time as long as the actor maintains the display active.

Actors using this use case:

- Trader
- Market-Maker
- Clearing Firm Broker

2.12 Display Trader's Activity Log

An actor may request a display of his/her trading activity, including orders entered, cancelled, replaced, and filled during a selected period of time.

Actors using this use case:

- Trader
- Market-Maker
- Clearing Firm Broker

2.13 Query Market History

An actor may request a display of the quote and trade history of a selected product. This information includes the best bids and offers, last sales, and their times and dates disseminated by the System since a specified time and date.

Actors using this use case:

- Trader
- Market-Maker
- Clearing Firm Broker

2.14 Enter Market-Maker Quote

A market-maker may submit a Market-Maker Quote representing his/her intention to trade a product. The System treats a Market-Maker Quote as a pair of limit orders, and as such, it attempts to execute them immediately or otherwise book them for later execution.

Actor using this use case:

- Market-Maker

2.15 Cancel Market-Maker Quote

A market-maker may cancel the unfilled portion(s) of his/her quote in a selected product.

Actor using this use case:

- Market-Maker

2.16 Display Book Depth

A market-maker may request a display of the book depth of a selected product. This display is only a snapshot that needs to be refreshed manually.

Actor using this use case:

- Market-Maker

2.17 Display Market-Maker Quotes

A market-maker may request a display of his/her outstanding quotes. The System will update the data in real time as long as the market-maker maintains the display active.

Actor using this use case:

- Market-Maker

2.18 Enter RFQ

An actor may submit a Request for Quote (RFQ) to notify market-makers to provide quotes. An RFQ can also be generated automatically, while executing a market order, for example. The System will maintain a log of market-maker quotes submitted in response to RFQs to periodically evaluate the fulfillment of the market-makers' obligation to provide liquidity to the market.

Actors using this use case:

- Trader
- Market-Maker
- Clearing Firm Broker

2.19 Disseminate Best Quote to TPF

The System sends a best quote to OPRA (via TPF) each time the best bid and offer in the book falls below the Exchange Prescribed Width (EPW) defined for the product. Conversely, the System sends an empty best quote each time the best bid and offer in the book outdistance the EPW.

Actor participating in this use case:

- OPRA (via TPF)

2.20 Disseminate Reports to TPF

The System sends a Fill Report(s) and/or Trade Report to TPF each time it matches a set of orders. The report(s) contains trade information for all the parties to the trade

Actor participating in this use case:

- TPF

2.21 Start-up System

This UC comprises all the steps that are necessary to prepare the System for operation.

Actor using this use case:

- Operations/Help Desk

2.22 Shutdown System

This UC comprises all the steps that are necessary to shutdown the System.

Actor using this use case:

- Operations/Help Desk

2.23 Pre-Open Product

The System pre-opens a product following the procedure described in section XIII.A of the *SBT Functional Requirements*. This use case may be initiated manually by an Operations/Help Desk person or automatically at a pre-defined time dependent on the opening time of the underlying asset. Since all the series in a class share the same underlying, all are pre-opened, opened, halted and closed simultaneously.

Actor using this use case:

- Operations/Help Desk

2.24 Open Product

The System opens a product following the procedure described in section XIII.A of the *SBT Functional Requirements*.

Actor using this use case:

- Operations/Help Desk

2.25 Halt Product

The System halts trading of a product automatically when one of a number of events occurs. The most common is when the primary exchange has halted trading of the underlying asset. An Operations/Help Desk person may also manually halt trading of a product.

Actor using this use case:

- Operations/Help Desk

2.26 Close Product

The System closes a product following the procedure described in sections V and XIII.B of the *SBT Functional Requirements*.

Actor using this use case:

- Operations/Help Desk

2.27 Post-Close Product (End of Day Process)

The System post-closes a product following the procedure described in section V of the *SBT Functional Requirements*.

Actor using this use case:

- Operations/Help Desk

2.28 Set Trading Parameters

An Operations/Help Desk person may set system-wide trading parameters such as the exchange minimum widths, minimum market-maker order default size, time for market-makers to respond to RFQs, etc.

Actor using this use case:

- Operations/Help Desk

2.29 Set Trader's Defaults and Preferences

An actor using a workstation may set defaults and preferences applicable to any trader in order to customize trading as well as the user interface to his/her needs.

Actors using this use case:

- Trader
- Market-Maker
- Clearing Firm Broker

2.30 Set Market-Maker's Defaults and Preferences

A market-maker using a workstation may set defaults and preferences specific to market-makers in order to customize trading as well as the user interface to his/her needs.

Actors using this use case:

- Market-Maker

3. Logical View

The diagram below shows the top-level logical structure and context of the SBT System.

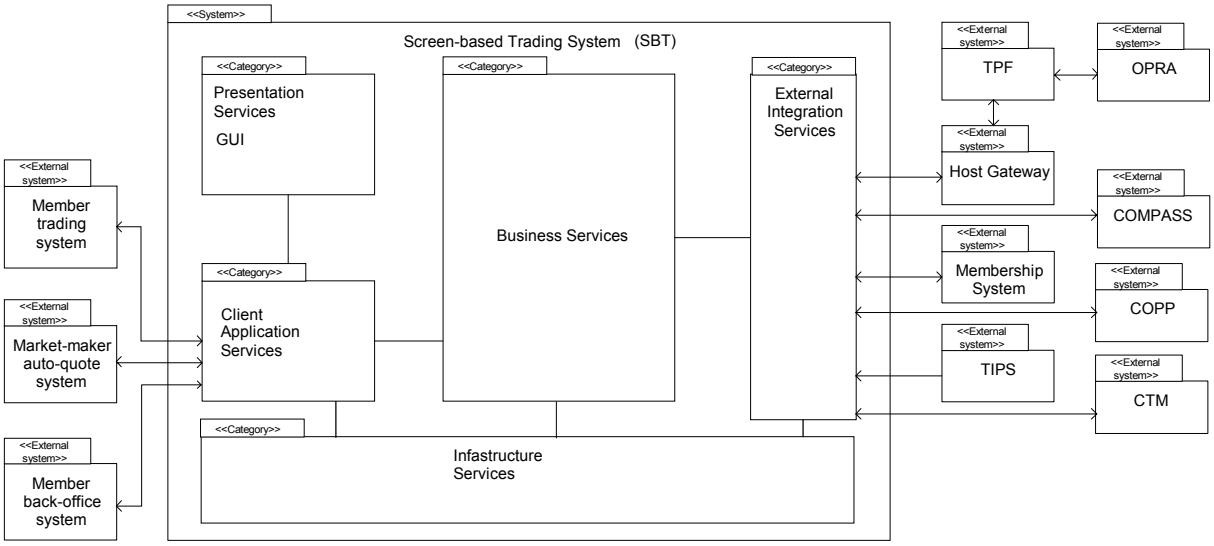


Figure 2 System Logical Components and Context

The SBT System is comprised of the following five logical packages of functionality ("Class Categories" in UML terminology):

- Presentation Services (GUI)
- Application Services (Client Application Server, Gateway)
- Business Services
- External Integration Services
- Infrastructure Services

The Presentation Services (GUI) package is constituted by applications that interact with the SBT system via the CBOE Market Interface (CMi) API described later. There are two types of client applications, those that provide a GUI to allow user interaction with the system directly and applications that automate trading functions.

The Application Services package contains subordinate packages that forward requests initiated by human or automated actors, to be executed by the appropriate Business Services package(s). In addition, the Application Services packages maintain instantaneously updated views that reflect the prevailing state of each actor's information in the Business Services package.

The Business Services package contains the core functionality of the SBT system. It includes components that correspond to the key business object model entities of screen-based trading such as members, orders, books, products, quotes, et cetera. In addition, it includes components to administer and operate the SBT system.

The External Integration Services package consists of adapters that map the interaction paradigms of external systems to the ones in the SBT system architecture.

Finally, the Infrastructure Services package contains Commercial Off-the-Shelf Software (COSS) and extended infrastructure services that provide enterprise-wide support to CBOE systems.

Figure 3 SBT System Logical View below shows the detailed logical components of the SBT System and their key interactions within the system and with external systems.

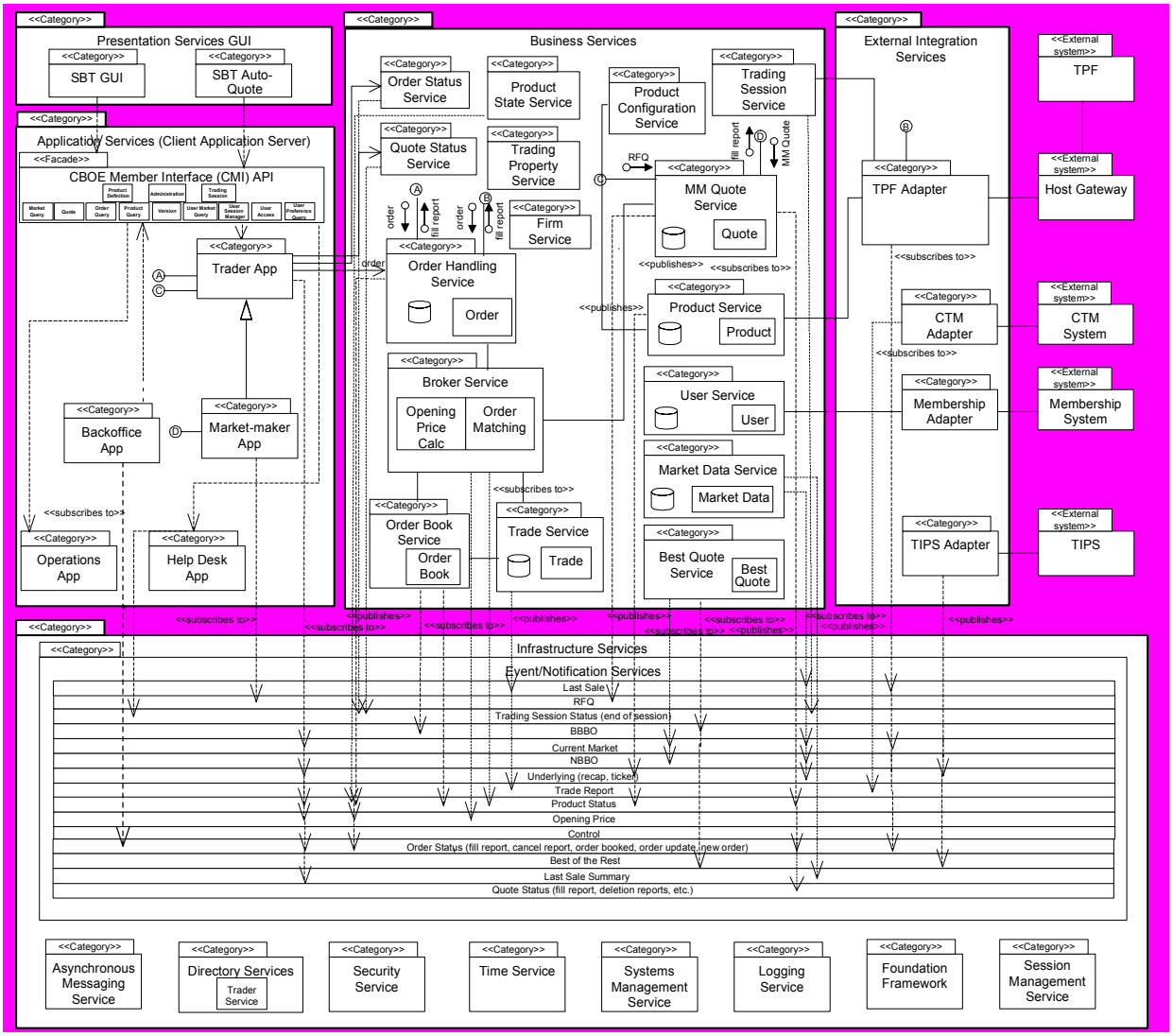


Figure 3 SBT System Logical View

The following section describes the responsibilities of the subordinate packages in each of the top-level packages.

3.1 Logical Packages

3.1.1 Presentation Services Package

The Presentation Services package contains six subordinate packages:

- SBT GUI – contains GUI applications to allow human actors interaction with the SBT system
- Models – contains MVC models
- Internal Event Channel
- Registered Callback Listeners
- API Translator

3.1.1.1. SBT GUI

The SBT GUI display is based on the model-view controller pattern. The SBT GUI Display is responsible for displaying the contents of a particular model to the screen and updating the display if the model's contents change. This package contains several GUI applications, one for each of the major classes of human actors that use the SBT System: traders, market-makers, clearing firm brokers, firms, and system operators.

- Trader GUI – is used by regular traders. It consists of several GUI's for displaying and entering orders, and market data.
- Market-Maker GUI – is an extension of the Trader GUI and is used by market-makers. It consists of several GUI's for displaying and entering orders, quotes, and market data.
- System operation GUI – is used by system operators and help desk operators
- Class Display GUI – is a subset of the Trader GUI that provides access to market data only.

A GUI framework is implemented here to promote the consistency in look and feel and speed up the GUI development work.

3.1.1.2 Models

Certain models including product class, product and order/quote models are singletons within a distinct GUI application and thus will be shared among multiple frames and panels. For the purpose of speed of display, each market data display panel/frame has its own copy of a market data model and underlying model for those products in which a particular frame is interested. Each model that can be updated by a remote event implements the *EventChannelListener* interface.

3.1.1.3 Internal Event Channel

The internal event channel is a pattern that is based on Java custom events. For each type of custom event that can be registered, a model (*EventChannelListener)* can be added as a listener to the Event Channel. When a remote event is received, it is routed to the correct channel and all listeners of that channel are notified of the event and passed any user defined data that wants to tag along with the event. The internal event channel exists in its own thread and has a queue.

3.1.1.4 Registered Callback Listeners

When a remote method is called on the CAS/CMi, an optional callback object can be passed which then registers the Presentation Services for any update "callback" messages from the CAS/CMi. There is one listener for each type of message that can be received. When a listener receives a message from the CAS/CMi, it will forward the message to the appropriate "Internal Event Channel" where it will then be forwarded to any interested event channel listeners. Each callback listener lives in its own queued thread.

3.1.1.5 API Translator

The API Translator is responsible for hiding the complexities of communication with the CAS from the GUI and Models. All remote requests by the GUI go through the translator. The translator will take a request for data, register the Event Channel Listener, and create and pass the callback listener. It also keeps track of the user data so that the GUI does not need to supply it when making remote calls.

3.1.1.6. SBT Auto-Quote

This package contains a single application that duplicates the functionality of the auto-quote generation engine currently used by market-makers in CBOE's trading floor posts.

The SBT Auto-Quote application will run remotely on the market-maker's computer to comply with the "fairness in trading" requirement – if it ran centrally, market-maker's would have an advantage over other traders as a result of reduced latency. Only the parameters for calculating auto-quotes are saved centrally to allow market-maker's to move from one workstation to another. The SBT Auto-Quote will use the CBOE Market Interface API and will input quotes to the Market-Maker Application.

SBT's own auto-quote facility should not be confused with the proprietary auto-quote facilities of market-makers, which will also utilize the CBOE Market Interface API. The SBT System is not responsible for storing the calculation parameters used by these proprietary auto-quote facilities, but only to accept orders, and quotes.

Note: SBT Auto-Quote will not be available in SBT Version 2.0.

3.1.2 Application Services Package

The Application Services package contains an application for each human actor in SBT. These applications submit requests to Business Services components, notify clients of business events, and maintain user-specific views of information in the Business Services.

This package encompasses a CBOE Market Interface (CMi) API that provides a single entry point to the system exposing the applications in the Application Services Package (i.e., Trader, Market-Maker). CBOE has chosen to support two different application programming interfaces. The first is a robust, full function API using the CORBA/IIOP protocol. This version of the API is intended for market makers and traders that plan on using high bandwidth functions, such as market data, and in the case of market makers, auto-quotation facilities.

Retail firms and some institutional customers will be interested in adopting the CBOE API FIX (Financial Information Exchange Protocol) Protocol version. CBOE is providing this protocol based upon the wide acceptance of FIX (4.2) as a standard for order entry, request for quotation (indication of interest), and execution reporting. The FIX Protocol is an industry standard created by retail and institutional investors for the trading of securities. The FIX protocol was extended to support derivatives in Version 4.1. The FIX protocol was originally designed to automate much of the trading that was done over telephone by providing messages for indications of interest, quotations, orders, and execution reports. The protocol was extended to the point where it is feasible to use this protocol for electronic trading from a retail or institutional perspective.

The FIX Protocol is implemented over the TCP/IP protocol. CBOE intends to support future versions of the FIX protocol - including the FIXML a standard from the FIX protocol working group implemented using the Extended Markup Language (XML).

Depending on existing systems, some firms may choose to adopt both APIs for different applications.

See *CBOE Application Programming Interface* for more information on the CBOE Market Interface.

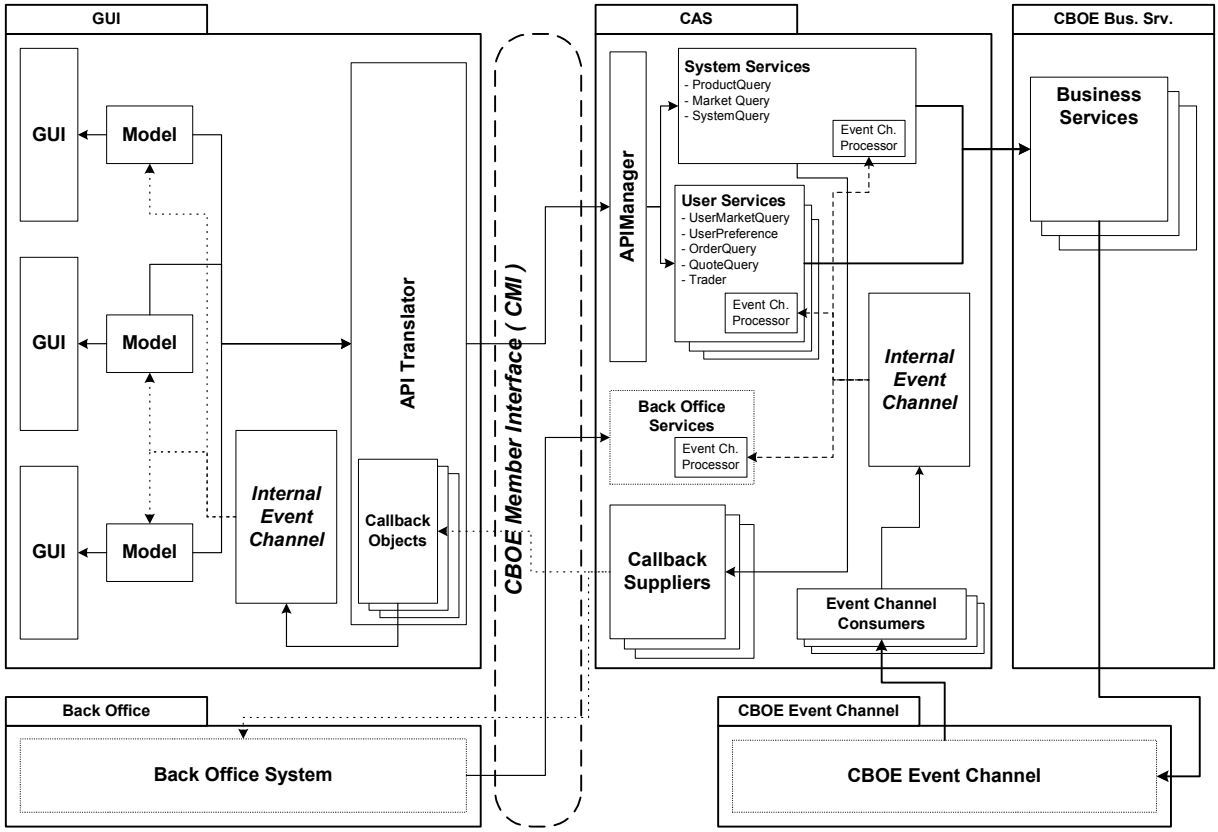


Figure 4 Presentation and Application Services Components

3.1.2.1. Trader Application

This package has the following specific responsibilities:

- Submit, cancel, update, and cancel/replace orders
- Submit requests for quotes
- Present the current status of the trader's orders
- Present fill and cancel reports
- Present Market Best Bids and Offers for selected products
- Set the trader's defaults and preferences
- Present underlying quotes/last sales and news alerts

3.1.2.2. Market-Maker Application

This package inherits the Trader App package's responsibilities and adds the following:

- Submit and modify market-maker quotes
- Present requests for quotes

- Set the market-maker's defaults and parameters
- Set auto-quote parameters
- Submit auto-quotes
- Present Book Depths for selected products

3.1.2.3. BackOffice App

This package is responsible for reporting order status information. This can include fill reports, cancel reports and new order notifications.

3.1.2.4. Operations App

This package has the following responsibilities:

- Start and shutdown the SBT system
- Start and stop trading of a product
- Change the status of a product's market (pre-open, open, close, halt, etc.)
- Present logged system events
- Maintain SBT-specific trader information
- Maintain SBT-specific product information
- Maintain trading parameters (spread width, minimum market-maker order default size, required percent of responses to an RFQ, maximum response time to an RFQ, etc)

3.1.2.5. Help Desk App

This package has a subset of the responsibilities of the Operations App. The specific responsibilities have yet to be defined.

3.1.2.6. CBOE Market Interface API

The functionality of the Trader, Market-Maker, and BackOffice packages is exposed by a façade, the CBOE Market Interface (CMi) Application Programming Interface (API). The CBOE Market Interface exposes different subsets of functionality depending on the user that logged on to the system. The intention behind sharing a common API among the different trader classes is to allow workstations installed on the floor to service all of them.

The CBOE Market Interface API supports both SBT client applications and external applications owned by members. Members use the CBOE Market Interface API to link their existing computer systems to SBT, to submit orders electronically and to automate trading. Likewise, market-makers use the API to generate auto-quotes employing their proprietary systems, instead of the default auto-quote application provided by SBT.

A major requirement of the Screen-Based Trading system, as specified in the *SBT Software Requirements Specification*, is to provide an Application Programming Interface (API) to enable members' computer systems to interact

with the SBT system. The targeted users of this API are member firms, non-market-maker professional broker/dealers (BD), and market-makers (MM). Although these types of users have different capabilities, they are sufficiently similar to suggest the use of a single API with usage restrictions according to the type of user, as opposed to using multiple APIs.

The following SBT system functions must be accessible through the API:

- Session logon and logoff
- Market state inquiry and change notification
- Connection status inquiry and change notification
- Order entry, cancellation, and replacement
- Quote entry, cancellation, and replacement
- RFQ notification
- Order status inquiry and fill notification
- Subscription to product markets
- Best market quotes notification
- Book "depth" inquire and change notification

The API is not intended for automating administrative or configuration functions; members will have to use a SBT Workstation to manually perform these functions with proper authorization.

The API leverages the distributed infrastructure utilized by CBOE to ensure a reliable, efficient and secure transmission of messages to and from the SBT System. In addition, this leverage will substantially simplify the development of the API.

CBOE will provide a single portable API or a set of functionally equivalent APIs for deployment in members' platforms. In addition to the API, CBOE will provide the required libraries for the distributed infrastructure components such as Object Request Broker (ORB), messaging service, event service, security service etc. With the API, members will develop applications to automate trading or gateways to allow other computers to automate trading. These applications or gateways will run in member's computers connected to CBOE via leased lines.

CBOE will make special simulation server components available to members to facilitate the development of applications. The simulation server components will support all the functionality of actual servers and provide a realistic environment for stress testing the member's application.

The API's functionality will be exposed as a set of CORBA IDL interfaces to facilitate its use from any language with a CORBA mapping. Alternatively, the API may be exposed as a framework in a particular language, such as Java, to be linked directly with an application, thus avoiding inter-process communication and consequently increasing response time.

To simplify application development and maximize performance, the API will be multi-threaded and thread-safe. Also, the API exposes the public interface of the underlying application (i.e., Trader, Market-Maker, UserAccess, etc.).

Replicated objects are persisted by the API—if the client application fails or the connection to server components is dropped, upon re-starting or reconnecting the API restores the objects to its prevalent state. The API only presents a snapshot of the member's current trading state at any given time; the application is responsible for storing relevant transactions to reconstruct history. Whenever possible, data validations will be performed locally using cached data to avoid unnecessary network traffic.

The API conforms to the event-based programming model—the API calls back a registered *event handler* upon occurrence of an event, passing the details of the event to the event handler. For example, to receive notifications every time an order is (partially) filled, the application first needs to register an application object which implements an API-defined interface with the API. Once a fill occurs, the API will invoke a method defined in the interface on the application object.

Events are fired asynchronously as soon as they occur, however, an event may be delayed if the application is not ready to process it. This will depend on the threading support of the application development environment used.

Invocation of critical methods is synchronous in order to let the application know of the success or failure of the call. Armed with this information, the application can decide whether to continue normal processing or take corrective actions immediately, instead of having to wait for an asynchronous reply. Order and quote submissions are examples of synchronous calls.

To simplify the development of the application and the API, transaction boundaries will not span more than a single method invocation.

As described in Section 3.1.2, CBOE will support a FIX protocol in addition to the IDL-based interface.

3.1.2.6.1 UserAccess

The User Access service provides for authenticated login to the Application Server. The User Access service is the first and last service accessed by an application. The logon method returns a reference to the API Manager service - which provides access to the application server services.

3.1.2.6.2 UserSessionManager

The User Session Manager service provides access to the services provided by the Application Server. Access to the User Session Manager is returned after a valid user login. It is through the User Session Manager that access to all other services is obtained.

3.1.2.6.3 OrderEntry

The Order Entry service accepts orders, updates to orders, and requests to cancel orders.

3.1.2.6.4 MarketQuery

The Market Query service provides access to market data, and market data history. The query methods are used to retrieve data in snapshots - the currently available information being returned upon invocation. The MarketQuery service also is used to subscribe for market data. The Market Query service is used to subscribe to ticker data, sales summary, and last sales information. Methods are available to unsubscribe.

3.1.2.6.5 UserPreferenceQuery

The User Preference Query service is used to store and retrieve user preferences within the User Service.

3.1.2.6.6 Quote

The Quote service provides the following: Accepts one or more quotes for the client application, the ability to cancel the current quote for a product, retrieval of Request for Quotes ("RFQ"), and retrieval of quotation history.

3.1.2.6.7 OrderQuery

The Order Query service provides for the retrieval of orders. The order queries return the initial snapshot of orders meeting the selection criteria and at the same time subscribe the caller to subsequent updates to the orders that meet the criteria.

3.1.2.6.8 ProductQuery

The Product Query service provides for the retrieval of product information.

3.1.2.6.9 ProductDefinition

The Product Definition service provides the ability to request markets for new products by specifying contract terms and underlying products.

3.1.2.6.10 Version

The Version service provides a constant that identifies the CMi version.

3.1.2.6.11 Administrator

The Administrator service provides access to text messaging operations.

3.1.2.6.12 TradingSession

The Trading Session service provides users with information on trading sessions.

3.1.2.6.13 UserHistory

The User History service provides for the retrieval of a user’s trading activity.

3.1.2.6.14 UserTradingParameters

The User Trading Parameters service is used to store and retrieve market maker's Quote Risk Management (QRM) preferences.

3.1.2.7 Application Services Components

The Application Services package contains the following types of components:

- Business Services Proxy/Cache
- Callback Suppliers
- Remote Event Listeners
- Remote Event Processors

3.1.2.7.1 Business Services Proxy/Cache

The Business Services Proxy/Cache provides the connection between the API and the CBOE business services. Any caching of remote data is accomplished in internal models owned by the Business Services Proxies.

3.1.2.7.2 Callback Suppliers

The callback suppliers are responsible for sending updated data through the Common Client API to a remote listening application. Each type of callback has its own supplier object. Similar to the Internal Event Channel used by the Presentation Services, a callback object is registered with a particular channel or set of channels. Each callback object then serves multiple remote callback clients and manages which types of data to send to which remote clients. The callback supplier object is threaded and queuing. Synchronous callbacks are used to communicate remotely in order to insure that the Presentation Services layer receives the message.

3.1.2.7.3 Remote Event Listeners

The Remote Event Listeners are objects that listen to the Event Service (event channels). The Remote Event Listeners are responsible for registering any needed filters on the event channel and collecting any events in which the Application Services layer is interested. When a remote event is received off of the event channel, it is passed to a Remote Event Processor object to be processed before sending to the business services proxy/cache layer. A single Remote Event Listener object can have multiple registered Remote Event Processors. This allows multiple Business Services Proxies to listen for the same event. There is one Remote Event Listener object for each type of event in which the Application Services tier is interested. The Remote Event Listener object is threaded and queuing.

3.1.2.7.4 Remote Event Processors

The Remote Event Processors are owned by each of the different business services proxies. They are registered with the Remote Event Listeners and have the responsibility of manipulating any data received off of the event channel into the format in which the particular Business Service Proxy is interested. Different Business Services Proxies can have personalized Remote Event Processors that listen for the same event or to the same Remote Event Listener. Remote Event Processors are threaded and queuing.

3.1.2.7.5 Business Services Emulators

The Business Services Emulators provides simulations of the CBOE business services. These components are utilized by the CMi SDK to provide a stand-alone test platform for developers.

3.1.3 Business Services Package

3.1.3.1 Order Handling Service

The Order Handling Service package maintains the current state of all orders persistently. Specific operations may be exposed directly by Order objects, bypassing the Order Handling Service. Logically, Orders are components of this package. Specifically, the Order Handing Service and Order components are responsible for:

- Receiving and storing incoming orders (from SBT clients or TPF)
- Forwarding incoming orders to the Broker package for execution
- Receiving order state change notifications from the Broker and Order Book packages and updating stored orders with this information. The functionality is provided by exposing Orders as first-class objects, allowing the Broker and Order Book components to directly update the orders.
- Sending fill reports to originating traders upon receiving fill notifications from the Broker and Order Book packages
- Receiving order cancellation requests and forwarding them to the Broker and Order Book packages. Upon confirmation of a cancellation, notifying the originating trader of the result of the request and updating the stored state of the order
- Receiving order cancellation/replacement requests and forwarding them to the Broker and Order Book packages. Upon confirmation of the cancellation/replacement, notifying the originating trader of the result of the request and updating the stored state of the order

Note: includes the Order Maintenance Service, and Order Routing Service.

3.1.3.2 Broker Service

The Broker Service package is responsible for executing the following types of orders:

1) Limit
2) Market
3) All or none (AON)
4) Fill or kill (FOK)
5) Immediate or cancel (IOC)
6) Stop
7) Stop Limit
8) Spread

Upon trade execution, the Broker Service is responsible for notifying the Trade Service package of all the orders matched (all parties to the trade) in the trade. It is also responsible for notifying the Order Handling Service (i.e. Orders) and MM Quote Service (i.e. Quotes) of the fills.

The Broker Service is also responsible for routing orders for open-outcry products to TPF.

3.1.3.3 Order Book Service

The responsibilities of the Order Book Service package are:

- Cooperate with the Broker in calculating the opening price during a products pre-opening period
- Acknowledge that an order was accepted by publishing an event consumed by the Trader application which originated the order
- Cancel and cancel/replace resting orders
- Upon changes to the top of the book, publish the new BBBO, book depth (top N), and last sale

3.1.3.4 Trade Service

The responsibilities of the Trade Service package are:

- Receiving trade notifications from the Broker Service
- Formatting trade reports
- Storing trade reports
- Forwarding trade reports to CTM (via TPF)
- Receiving Trade Bust requests
- Receive Block Trades and Exchange for Physical (EFP) trades.

Note: includes the Trade Maintenance Service.

3.1.3.5 Market Maker Quote Service

The Market Maker (MM) Quote Service package is responsible for:

- Receiving requests for quotes (RFQs) from traders
- Submitting RFQs to market-makers assigned to the product for which the quote was requested (by publishing in the RFQ event channel)
- Receiving and logging market-maker responses to RFQs (market-maker quotes)
- Upon receiving a market-maker quote, saving it persistently and submitting them to the Broker Service package for execution.
- Sending fill reports to originating market-makers upon receiving fill notifications from the Broker and Order Book packages
- Canceling or updating a MM quote upon receiving a request from the originating market-maker by submitting the request to the Broker/Order Book.
- Upon inquiry, providing the history of the quotes submitted by a market-maker
- Monitoring a market-maker's fill rate, and submitting a cancel request to the Broker when the rate exceeds the user's fill rate limit (QRM - Quote Risk Management).

3.1.3.6 Product Query Service

The Product Query Service package maintains all product-related information. In order to perform these responsibilities, the Product Service package downloads and caches product information from TPF and TIPS (options only). The service also allows for the creation and maintenance of a variety of derivative products, e.g. equity futures. Note: includes the Product Maintenance Service.

3.1.3.7 User Service

The User Service package maintains all user-related information, both specific to SBT and contained in CBOE's Membership System and Market Planning and Procedures (MPP) System. It provides a unified interface to SBT components accessing user information, hiding the actual location of the maintained data, thus simplifying client logic and facilitating replacement of the Membership System in the future.

The User Service maintains the information of:

- traders
- market-makers
- clearing firm brokers
- operators

- help desk personnel
- back-office personnel

Note: includes the User Definition Service, User Maintenance Service, User Maintenance Event Service, and User Trading Parameter Service.

3.1.3.8 Firm Service

The Firm Service package maintains all firm-related information, both specific to SBT and contained in CBOE's Membership System. It provides a unified interface to SBT components accessing firm information. Note: includes the Firm Maintenance Service.

3.1.3.9 Exchange Service

The Exchange Service package maintains all exchange-related information, both specific to SBT and contained in CBOE's Membership System. It provides a unified interface to SBT components accessing exchange information.

3.1.3.10 Trading Session Service

The Trading Session Service package maintains all business day and trading session-related information and manages the different states of a trading session, e.g. open, closed, and halted. What products are processed/traded in each trading session is also kept at this service. In order to perform these responsibilities, the Product Service package needs to download trading session and product information from TPF, as well as monitor events that affect products traded within a session. Note: includes the Trading Session Maintenance Service, and Trading Session Event State Service.

3.1.3.11 Product State Service

The Product State Service package is responsible for coordinating product state changes for all products, e.g. pre-opening, opening, trading, halting, closing, and post-closing. It works closely with the Broker Service to insure that state changes occur in a timely fashion. The service monitors events that affect products traded, such as monitoring the underlying market to detect when the primary exchange opens, closes or halts trading a product.

3.1.3.12 Product Configuration Service

The Product Configuration Service package is responsible for assigning a product's trading/processing location. The information is used primarily by the Product Routing Service to route product-specific requests (e.g. orders) for processing. Additionally, the PCS can be used to create and populate logical groups of products for a variety of purposes.

3.1.3.13 Product Routing Service

The Product Routing Service package is responsible for providing the location of where a product is processed/traded. This information is primarily used to route product-specific requests (e.g. orders) for processing.

3.1.3.14 Order Status Service

The Order Status Service package provides subscription and notification services related to orders (i.e. fill reports, cancel reports, order accepted by book, etc.) Note: The service replaces the use of event channels for order status reporting, providing a more secure mechanism for status delivery, i.e. point-to-point, using the Order Status Subscription Service.

3.1.3.15 Quote Status Service

The Quote Status Service package provides subscription and notification services related to quotes (i.e. fill reports, deletion reports, etc.) Note: The service replaces the use of event channels for quote status reporting, providing a more secure mechanism for status delivery, i.e. point-to-point, using the Quote Status Subscription Service.

3.1.3.16 Trading Property Service

The Trading Property Service package maintains system wide trading parameters (e.g. exchange prescribed width, fast market spread multiplier) for use by various business services. The service supports hierarchical relationships for certain parameters that allow exchange-wide parameters to be overridden at the trading session, or product class level.

3.1.3.17 Market Data Service

The Market Data Service package maintains a current snapshot of market data, in addition to publishing market summary data. The package also provides an interface to clients to query historical market data (MDR). Note: includes the Market Data Summary Service.

3.1.3.18 Text Messaging Service

The Text Messaging Service package provides messaging between the help desk and other users.

3.1.3.19 User Activity Service

The User Activity Service package provides the ability to query for a user's trading activity, which unifies order, RFQ, and quote-based activity.

3.1.3.20 Best Quote Service

The Best Quote Service package is responsible for calculating the market best (best quotes) for each product and sending them to COPP or TPF (which in turn forwards them to OPRA) for public dissemination. In addition, it is responsible

for calculating and disseminating the NBBO. In order to perform its responsibility, the Best Quote package subscribes to the Best of the Rest event channel to receive the best bids and offers for all products and publishes the results to the Best Quote event channel, of which the COPP and TPF Adapters are subscribers.

3.1.4 External Integration Services

The External Integration Services package contains subordinate packages implementing the Adapter Pattern that "adapt" (or "wrap") the native legacy interfaces to interfaces appropriate in the DTE/SBT environment. These new interfaces are designed in such a way that no changes will be necessary to SBT when DTE components eventually take over TPF's responsibilities. In other words, DTE components will simply re-implement these interfaces and probably extend them with trading floor-specific functionality.

3.1.4.1. TPF Adapter

This package contains the adapter to allow SBT and TPF to interact. TPF data is received, repackaged, and broadcast/delivered to the appropriate components within SBT. Conversely, SBT data is received, either through direct invocation or event consumption, repackaged, and sent to TPF using its native interface. The TPF Adapter always maintains an active HGW connection. When failure is detected, it should reestablish the connection with the backup HGW.

The TPF Adapter handles the following data flows:

- Order (TPF to SBT) – TPF sends a member's wire order received from COMPASS to SBT
- Cancel Order (TPF to SBT) – TPF sends cancellation of a member's wire order received from COMPASS to SBT
- Cancel Request Status (SBT to TPF) – SBT either confirms the cancellation of a wire order, or responds that the cancellation was received after the order was filled, to TPF
- Order Fill Report (SBT to TPF) – SBT reports when a wire order is (partially) filled to TPF
- Best Quote Report (SBT to TPF) – SBT broadcast quotes for products traded in SBT to TPF for external dissemination via OPRA
- Order (SBT to TPF) – SBT sends a CMi trader's order received from the CMi to TPF
- Cancel Request Status (TPF to SBT) – TPF either confirms the cancellation of a CMi entered order, or responds that the cancellation was received after the order was filled, to SBT
- Cancel Order (SBT to TPF) – SBT sends cancellation of a trader's order received from CMi to TPF
- Order Fill Report (TPF to SBT) – TPF reports when an open-outcry routed order is (partially) filled to SBT

- TPF time (TPF to SBT) – TPF provides its time upon request, so SBT server nodes can synchronize their clocks
- Product related flows – TPF notifies SBT of intra-day events affecting products such as new series added, stock splits, et cetera

3.1.4.2. Membership Adapter

The Membership Adapter translates requests for member information received from SBT components into requests to the Membership System and returns the results after re-formatting. Note: includes market-maker assignments from the Market Procedures and Planning (MPP) system.

3.1.4.3. TIPS Adapter

The TIPS Adapter subscribes to TIPS to receive the external market data needed in the SBT environment, including underlying market data and the Best of the Rest of options listed in SBT. It is necessary to determine whether the Events Service is capable of notifying the TIPS Adapter of consumer subscriptions so that it can propagate these subscriptions back to TIPS.

Once subscribed, The TIPS Adapter reformats the market data received from TIPS and publishes it for consumption by SBT components.

Another responsibility of this adapter is to publish product state events when external markets change their states, for instance when they open, halt, close, etc.

3.1.4.4. COMPASS Adapter

This package contains the adapter to allow SBT and Compass to interact directly in support of single stock futures. Compass data is received, repackaged, and delivered to the appropriate components within SBT. Conversely, SBT data is received, repackaged, and sent to Compass using its native interface.

The Compass Adapter handles the following data flows:

- Order (Compass to SBT) – Compass sends a member's wire order received from the firm to SBT
- Cancel Order (Compass to SBT) – Compass sends cancellation of a member's wire order received from the firm to SBT
- Order Fill Report (SBT to Compass) – SBT reports when a wire order is (partially) filled to Compass
- Order Cancel Report (SBT to Compass) – SBT reports when a wire order is (partially) cancelled to Compass

3.1.4.5 COPP Adapter

The COPP Adapter receives SBT market data, e.g. last sales, and forwards it to COPP for dissemination to various data providers, e.g. OPRA.

The COPP Adapter handles the following data flows:

- Best Quote Report – SBT broadcast quotes for products traded in SBT to COPP for external dissemination via OPRA
- Last Sale Report – SBT broadcast last sales for products traded in SBT to COPP for external dissemination via OPRA

3.1.4.6 CTM Adapter

The CTM (CBOE Trade Match) Adapter receives SBT data and forwards it to CTM.

The CTM Adapter handles the following data flows:

- Trade Report (SBT to CTM) – SBT reports all the parties to a trade to CTM

3.1.5 Event Flows

One key mechanism by which SBT components interact with each other is by supplying and consuming events, implemented as a publish/subscribe pattern. The following list provides a brief description of the significant event flows/notification services (messaging services) depicted in the Events Service within the Infrastructure Services package in *Figure 3 SBT System Logical View* as well as their corresponding Quality of Service (QoS).

- RFQ – the MM Quote Service supplies Request for Quote (RFQ) events consumed by the Market-Maker Application. QoS: Store and forward, FIFO, time to live, filtered.
- BBBO – the Order Book supplies Book Best Bid Offer (BBBO) events consumed by the Best Quote Service. QoS: Best effort*, FIFO, Publisher transactional, low priority delivery, filtered.
- Book Depth – the Order Book supplies Book Depth (top N prices and quantities) events consumed by the Trader Application. QoS: Best effort*, FIFO, Publisher transactional, low priority delivery, filtered.
- Current Market – the Best Quote Service supplies Current Market Best and National Best Bid Offer (NBBO) events, containing a product's best quote, consumed by the Market Data Service and Trader Application. The best quote indicates if the exchange has the best quote. QoS: Best effort*, FIFO, Publisher transactional, medium priority delivery, filtered.
- Best of the Rest – the TIPS Adapter component supplies best-of-the-rest events consumed by the Best Quote Service. QoS: Best effort*, FIFO, filtered.
- Logging – the Logging Service Proxy component supplies Log Service events consumed by the Log Service component. QoS: Best effort, priority, time to live.
- System Management – the Foundation Framework supplies System Management events consumed by the System Management component. QoS: Store and forward, FIFO.

- Instrumentation – the Instrumentation Service component supplies Instrumentation events consumed by both the System Management component and the Log Service component. QoS: Store and forward, FIFO.
- Ticker – the TIPS Adapter supplies underlying ticker events (prices, quotes, last sales, news alerts) consumed by the Trader Application and the Product Service. The Trade Service supplies ticker (last sale) events consumed by the Market Data Service and TPF Adapter. QoS: Best effort*, FIFO, low priority delivery, Publisher transactional (Trade Service), filtered.
- Recap – the TIPS Adapter supplies underlying summary events (high and low prices, volume) consumed by the Market Data Service and Trader Application. The Market Data Service supplies summary events, based on last sale, consumed by the Trader application. QoS: Best effort*, FIFO, low priority delivery, Publisher transactional (Market Data Service), filtered.
- Trade Report – the Trade Service supplies Trade Report events consumed by the TPF Adapter. QoS: Stable store and forward, FIFO, Publisher and Consumer transactional, high priority delivery.
- Product Status – the Product Service and Product State Service supply Product Status events (State, Price Adjustment, Configuration, and Update) events consumed by the Trader application, Order Handling Service, and TPF Adapter. QoS: Stable store and forward, FIFO, Publisher and Consumer transactional, high priority delivery, filtered.
- Trading Session Status – the Trading Session Service supplies Trading Session State events consumed by the Operations and Help Desk application. QoS: Stable store and forward, FIFO, Publisher and Consumer transactional, high priority delivery.
- End of Session Summary – the Trading Session Service supplies End of Trading Session Status events. QoS: Stable store and forward, FIFO, Publisher and Consumer transactional, high priority delivery.
- Opening Price – The Broker Service supplies Opening Price events consumed by the Trader Application. QoS: Best effort*, FIFO, medium priority delivery, filtered.
- Control – the Operations and Help Desk applications supply Control events, possibly through the System Management component, consumed by Business Services and External Integration Services components. QoS: guaranteed, ordered, high priority delivery.
- Order Status – the Order Handling Service (Order) supplies Fill Report, Cancel Report, Updated Order, New Order, and Order Accepted by Book events consumed by the Order Status Service, and TPF Adapter. QoS: Stable store and forward, FIFO, Publisher and Consumer transactional, high priority delivery, filtered.
- Quote Status – the MM Quote Service (Quote) supply Fill Report, and Delete Report events consumed by the Quote Status Service. QoS: Stable store and forward, FIFO, Publisher and Consumer transactional, high priority delivery, filtered.

Note: * refers to a snapshot

3.1.5.1 Product Groups

The Notification Service provides the capability for a consumer to apply filters to an event channel to limit the number of events received, since many consumers are only interested in a limited subset of events within an event channel. This capability greatly reduces the messages sent by the system by eliminating the publication of unwanted events.

Many of the event channels handle events related to products, both classes and series (e.g. Trading Session Status). A typical consumer will use filtering to receive only a subset of products. Since the number of products in a class can be large, and the overhead of applying a large number of filters to an event channel can reduce performance, many event channels provide the capability to filter at the product group level. A product group defines a set of related products (e.g. all products trading in a virtual pit). Publishers of events involving products will include a set of product groups in the event. Consumers can reduce the number of filters by filtering on a product group, rather than each product in the group. The Product Configuration Service provides product group information for both consumers and publishers.

3.1.6 Infrastructure Services

Overall there are four major tiers of the application that can be identified. The business services handle all the SBT trade matching, execution and reporting functionality. It provides the repository for all SBT information data. The application services handle the application presentation and act as the application front end to the business services. Different views of the business services and collaboration of business objects are grouped together and are presented to the user based on logon authentication and authorization level. The two tiers communicate to each other by two supported tiers: the infrastructure services and external integration services. The infrastructure services take care of the plumbing underneath to provide a seamless integration between the application services and business services. The external integration services provide the access to the external system.

Most of the common services are grouped together inside the Infrastructure Services. The services are defined when possible using standard service specification (CORBA services) and technology engines to minimize the cost of development and deployment. At the same time, this approach also ensures the openness of the application architecture without locking the architecture into any one vendor product and technology. Below is the high level description of the common service and usage pattern of how the business object interact with the common services

3.1.6.1 Directory Services

Directory Services provide the ability to register an object and its properties, and the ability to locate them in the distributed environment. Directory Services does

not include a Naming service. The Trader service will be used for the Naming service functionality.

The Directory Service (consisting of Trader Services) will be constructed following the CORBA specification. This will give a consistent structure to the layout and implementation of the supporting interfaces.

Due to Directory Service's importance in providing clients location data, its availability is paramount. At this time, there is no specification for levels of resilience within the Object Management Group (OMG) specification. In addition, no commercially available implementation of the Trader Service adequately supports CBOE's needs. For this reason, the functionality will have to be developed in-house.

The Directory Service will need to persist the data registered with the service, and ensure that it can be recovered in the event of a failure. This may be accomplished by a replication strategy utilizing multiple instances of the Directory Service.

The Directory Service will utilize a commercial Lightweight Directory Service Protocol (LDAP) server to store persistent data, and provide replication and redundancy.

See *Directory Service Requirements* for more information on the Directory Service.

3.1.6.1.1 Trader Service

Objects advertise their service with the Trader Service. The object service consumer can query the Trader Service to locate the service. When registering for a service, objects can specify the properties of the service. There can be multiple registrations of services with different properties. When querying the Trader, the object service consumer can query the Trader using the service type and property(s) to narrow down the service for which it is looking.

Application Service objects are the primary user for the business object registration. An application like the Trader application would establish a session with the Client Session Management, which in turn, contacts the Trader Service to retrieve the available services.

3.1.6.2 Messaging Services

The CBOE Integration Architecture identifies two paradigms for messaging: Publish and Subscribe (P&S), and Asynchronous Messaging (AM). The first addresses a model in which the sender knows nothing of the receiver or target of the messages that it is sending. Likewise, the receiver does not know the origin of the messages it receives. The result is a powerful multi-point to multi-point mechanism, which supports graceful system evolution.

The Asynchronous Messaging model extends the basic point-to-point synchronous invocation model of CORBA to support asynchronous invocation, independent of client or server related failures.

See *Messaging Service Requirements* for more information on the Event/Notification and Asynchronous Messaging Service.

3.1.6.2.1 Event/Notification Service

The Event Service provides a publish and subscribe mechanism for delivering and receiving messages where the message supplier does not have the knowledge of the message consumers. CORBA Event Service Specification translates the message into event channel object. It specifies two models for providing the delivery of the message. With the *push model*, the supplier sends a push method invocation on the event channel object. The event channel object, in turn, pushes the event data to the consumer objects. With the *pull model*, the consumer issues a pull method invocation on the event channel object. The event channel object, in turn, pulls the event data from the supplier. Event data can be *typed* to allow business objects to describe the contents of events.

The CORBA Notification Service, recently adopted, extends the Event Service specification with the additional capabilities, including:

- The ability for clients to specify exactly which events they are interested in receiving, by attaching filters to each channel
- The ability for event suppliers to discover the event types requested by consumers, avoiding the transmission of events in which no consumers have interest
- The ability to configure various quality of service properties on a per-channel, per-proxy, or per-event basis.

Within SBT, the interface and processing model using the Event Service is sufficient to provide the messaging service, but the quality of service such as guarantee message delivery, priority event, message queuing will borrow from the Notification Service specification. Performance is also a concern in the SBT environment where certain messages (like quotes) can be extremely high in volume. Using a normal communication transport layer where messages are sent round-robin to each of the consumers is not acceptable. The messaging service infrastructure encapsulates all the quality of services while preserving the simple but flexible interface of the Event Service.

In SBT, the business object services and the application object services delegate the message delivery responsibility to the messaging infrastructure by specifying the quality of service when creating the event channel object. The mechanism of invoking and receiving all events is the same.

The following are the different Quality of Service (QoS) that are provided by the Event Service infrastructure:

- Transport Properties
- Ordering Properties
- Lifecycle Properties.

Transport Properties

Transport properties specify the guarantee vis-à-vis messaging handling, by the elements of the messaging services infrastructure, and between Suppliers and Consumers. Transport properties are set as an administrative attribute of the Channel.

- Best Effort – this QoS transport property expresses that the publish and subscribe messaging service should attempt, within implementation-dependent bounds, to transport a message. If these bounds are exceeded, the service drops the message.
- Store and Forward (Guaranteed) –this QoS transport property specifies that the publish and subscribe messaging service must keep one or more copies of a given message within the infrastructure elements so as to tolerate failure of one of those components.

In addition, this property can support disconnected usage.

- Stable Store and Forward –this QoS transport property specifies that the publish and subscribe messaging service must store each message on a persistent storage device at elements within the infrastructure so as to tolerate failure of any or all of those elements.

- Transactional Distribution –this QoS transport property specifies that the publish and subscribe messaging service must receive from Suppliers, and deliver to Consumers, under a transaction. This allows the messaging service to coordinate message state handling with application processing to assure that messages are properly delivered and processed.

Ordering Properties

Ordering Properties specify constraints on the sequence in which messages are delivered relative to one another.

- FIFO –this QoS ordering property specifies that messages should be delivered based on source ordering.

- Priority –this QoS ordering property places an absolute priority on processing a message relative to other messages.

- Temporal –this QoS embodies the concepts of applying a time-based ordering scheme to determine delivery sequence. Older publications are handled before more recent publications.

- Deadline –this QoS embodies the concepts of applying a deadline-based scheme to determine delivery sequence. The publications with deadlines occurring first are handled before publications with deadlines further out.

Lifecycle Properties

Timeliness Properties are concerned with how quickly publications are processed. These properties are of interest to suppliers and consumers in time sensitive applications. There are several potential schemes that the Publication Service should support.

- Time to Live –this QoS is indicative of the length of time a publication can exist in the system and still be considered valid. Suppliers and consumers who support time critical applications need the ability to define how long a publication can be considered valid.

In the process of creation of an event channel, multiple QoSes can be specified. (Only some QoSes are mutually exclusive to the others: Best Effort and Guaranteed, Ordered and Last Only Ordered)

Private Event Channels

In certain situations, a private event channel is required. All the listeners and publishers can communicate through the event infrastructure with all the benefit of the push and pull model between the two but in a secure and private conversation. For example, when the order is booked at the business service side, the Trader application needs to be notified. The information is intended for the sole use of the trader who originated the order. While the publication of the event would use the Order Status event channel, the Event Service infrastructure must ensure that only a single authorized consumer receives the event. This can be accomplished by using restrictive filtering that would apply an appropriate identity-based filter to outgoing events to ensure that only the original sender of the order was notified.

While the concept of private event channels appears to be a natural extension of the Event/Notification Service, the OMG specification makes no mention of the capability. In addition, the nature of the publish subscribe model, which isolates a consumer's identity, and the security ramifications of broadcasting private data on a multi-point channel, may require a alternative mechanism for secure multipoint-to-point communication.

An alternative to private event channels may use a combination of Publish and Subscribe and Asynchronous Messaging. A client requiring a private channel could register with a trusted mediator component residing on the business service side. The mediator would register with the Event Service as a normal consumer, receiving events destined for all registered clients. The mediator would then forward the event to the appropriate client using the secure, point-to-point, Asynchronous Messaging Service.

Note: Private event channels are not implemented at this time. A combination of publish and subscribe, and synchronous messaging is used where appropriate.

3.1.6.2.2 Asynchronous Messaging Service

The Asynchronous Messaging Service provides an interface (AMI) that allows point-to-point, or point-to-group, communications where the client or the server

are potentially disconnected when the other is processing a request or response. When using the AMI, a Callback Handler is passed from the client to the server, and the server returns the results of the operation by invoking an operation on the handler. The use of AMI is completely transparent to the server, allowing the server to provide both synchronous and asynchronous operations from a single interface. In addition, the service provides various Qualities of Service (QoS) with timeout notification.

3.1.6.3 Security Service

The Security Service provides the mechanism to control and to monitor access to the resources (objects) of the system. The following functional areas are addressed:

- Authentication
- Authorization (Access Control)
- Integrity and Confidentiality Protection
- Non-repudiation

With authentication, the user and server components are securely identified. It guards the SBT from the security threat of someone impersonating an authorized user of the system in order to perform actions, and have these actions attributed erroneously to other authorized users.

Authorization provides the specification and enforcement of access rights to the system resources (objects) based on the authenticated user or group. It guards the SBT from the security threats of someone invoking operations on the system without authorization. Access can be defined on a per-object or per-operation basis.

Integrity and Confidentiality Protection provides assurance of the accuracy and secrecy of the transmitted information. It guards the SBT from the security threats such as: someone eavesdropping to gain access to confidential information, someone gaining malicious or accidental access to information that should be hidden, someone tampering with the messages while in transit through the corporate network in order to alter their contents.

Non-repudiation provides the irrefutable evidence of actions such as proof of data sent and data received. It guards the SBT from the security threat of denying usage action.

In a CORBA environment, Security Service “policy” and “role” software agents will be defined, built, and distributed across multiple servers. These agents will be implemented as CORBA objects, allowing security administrators to control their operation remotely and change the set of policies enforced.

Secure CORBA ORBs can provide authentication, access control, auditing, and message protection. The CORBA philosophy is to protect objects that are not security-aware in such a way that the ORB can perform security policy enforcement automatically. Two "interceptors," the access control function and secure invocation, are the mechanism by which the ORB can invoke security operations (e.g. policy agents) on behalf of CORBA objects.

The environment will enable fully specified policies defined at the operation level.

The Security Service will utilize a commercial Lightweight Directory Service Protocol (LDAP) server to store persistent data, and provide replication and redundancy.

See *CBOE Security Requirements* and *SBT Security Policy* or more information on the Security Service.

3.1.6.4 Logging Service

The Log Service (Logging) provides clients the following functionality:

- System Monitoring: Logged messages alert system administrators of abnormal events and enable the general "health" and performance of the system to be monitored.
- Debugging: Logged messages allow system administrators to detect, trace, and diagnose system problems.
- Auditing: Logged messages provide an audit trail that enables system administrators to verify the valid use of system resources and assign accountability for actions
- Non-repudiation: Logged messages allow system administrators to confirm that a given transaction occurred and cannot be disputed by the client that initiated the transaction

In addition, the Log Service must provide these services in such a way that system performance, scalability, extensibility, and resilience are not compromised.

The Log Service is responsible for the creation and delivery of log messages intended for human viewing. It is not intended to provide inter-process communication or archiving and retrieval of data.

The Log Service is comprised of three basic components:

- The Logging Server process(es)
- Client-side proxies
- Frameworks or API's that facilitate monitoring and management of the Log Service by system administrators

The Logging Server provides a central Log Service, accessed indirectly by clients through a proxy. The central Log Service will have a repository for storing all log messages. A system may be comprised of a number of Logging Servers, providing for increased scalability and resilience.

The Client-side proxies provide a local object that directs log messages to an appropriate destination. The destination of the log message is unknown by the client, providing for a simple interface. The client-side proxy can be statically or dynamically configured to route various log messages, based on type or priority, to a variety of destinations (e.g. standard error, a local file, central Log Service). This approach provides an easily configurable, extensible framework for handling the variety of log messages during the various phases of the system lifecycle (e.g. development, testing, deployment).

The Control API allows control parameters to be initialized, queried, or set by an outside entity (e.g. system administrator). It provides for the static initialization of the Log Service through a configuration file, and the runtime configuration of the service through an exposed interface.

See *Log Service Requirements Document* for more information on the Logging Service.

3.1.6.5 System Management Service

System Management Service addresses the aspect of SBT system management, system monitoring and configuration. Each component in the business services and infrastructure services has to participate by exposing an administrative interface that will allow for remote management.

The System Management Service is based on the CORBA Management Facilities Architecture, which supports the X/Open Systems Management Reference Model. The specification defines four services:

- Managed Set Service: Organizes Managed Objects, resources that need to respond to management interactions in order to be managed, into hierarchical structures
- Policy-Driven Base Service: Allows Managed Objects to be managed by policy regions
- Instance Management Service: Provides basic creation and management capabilities for all types of Managed Objects
- Policy Management Service: Used to establish policy regions, assign Managed Objects to policy regions, report policies, and define policies

The services included in the CORBA Management Facilities form a powerful framework for implementing an object model of distributed, policy-driven Managed Objects representing resources in an installation. The object model is shared among all system management applications, thus promoting consistency and enforcement of enterprise-wide management policies.

Patrol has been identified to provide the user interface for monitoring and controlling SBT resources.

Application configuration is responsible for maintaining system-wide configuration parameters not associated with users, members or products. These parameters include, but are not limited to the following:

- Maximum time period for market-makers to respond to RFQs
- Percent of responses required to continue execution of market orders
- Current version and location of executables or dynamically loadable classes to support automatic component deployment

See *System Management Requirements* for more information on System Management.

3.1.6.6 Persistence Service

Persistence Service provides the ability to "persist (store)" objects data beyond the application that creates the object or the client that uses it. Persistence is provided by using a Persistence Framework within the Foundation Framework.

At the SBT Business Service tier, all business objects that require persistence use the Foundation Framework to store their state in an underlying persistent store. The Persistence Framework supports multiple products that provide persistence, such as a relational database system (Oracle), or a file system.

At the SBT Application Service tier, application objects are constructed by connecting to the business object service. After initialization, the application service builds a cache of all necessary application objects. Object state changes are propagated from the business object to the application service cache object through the event infrastructure.

3.1.6.7 Session Management Service

The Session Management Service manages user session by providing the facilities to create user session via logons. The service manages and monitors all user connections, and provides session clean up when the client process terminate (both gracefully and ungracefully). Business services can register with the service to be notified when a user session is terminated, allowing them to apply application specific clean up.

3.1.6.8 Time Service

Time Service provides the global time synchronization between the components within SBT from TPF. The server processes synch up the time with TPF in the morning and whenever they restart. The service is available as a global service. In addition, the Time Service provides timer functionality, allowing clients to be notified at the end of a specified time interval.

An alternate implementation, utilizing NNTP, is also available.

3.2 Processing Paradigm

This section describes the common processing paradigm using software design patterns. It details the collaboration between objects in a certain context to achieve a generic software construction.

3.2.1 Foundation Framework

The Foundation Framework is an application server framework consisting of a set of Java classes that enables using certain SBT infrastructure services in a consistent and effective manner for both the Application Service developers and Business Service developers.

Overall, the Foundation Framework functionality provides a default standard usage of the SBT infrastructure service. The developers will not be affected by the changing infrastructure technology, and they are not required to work with the infrastructure services directly. However, it will not force the developers to use the framework to get to the infrastructure services.

The framework provides an application or business server process with the ability to have multiple execution contexts where many service objects can be deployed using specified execution policies and a mechanism to provide authorization validation at the method invocation level. A mechanism is provided that collects and instruments the service statistical profile data by intercepting the invocation of the business service object, capturing the necessary information, then delegating the work back to the business service object. The service process is consistently managed by exposing an administration interface to allow an external application management process to have reactive or proactive control over the process operation. It facilitates logging integration, allowing a developer to statically define all log information at the development phase, and dynamically control the amount of log information to propagate to a central log repository. It also provides a standard interface to use the Persistent Framework/Service. The framework specifies creating, deleting, and finding operation interfaces. The implementation of the operation for each business object and the actual operation of maintaining the mapping between the object and the persistent layer is outside the scope of the Foundation Framework. The framework simply provides a façade interface to enable an underlying Transaction Framework/Service. There is no additional functionality of transaction management to be implemented at the framework.

This list details the Foundation Framework's specific functionality:

- Thread management—provides the following threading policy:
  - Reuse threading—uses the same thread as the caller. In this model, the framework relies on the developer to provide a threading model (the developer can implement the threading at the business object implementation, at the ORB, or at the Event Infrastructure).

  Other threading policies that can be implemented in the future include:

    - Shared thread policy—all incoming invocations are handled by a single shared thread. Service invocation is serviced one at a time by the shared thread.
    - Thread per request policy—for each incoming service invocation a thread is created and services the request. Service invocations are serviced in parallel.
- Directory Service Integration—provides an easy interface to the Directory Service, simplifying its usage.
- Security Integration—provides the security mechanism to validate authorization with the Security Service at the service invocation level. It collaborates with the Security Service to provide the authorization.
- Session Management Integration—provides an interface to the Session Management Service to establish and terminate user sessions. In addition, the interface provides the capability to register for notification of terminated user sessions.
- Application management interface:
    - Provides a mechanism to collect and calculate the service statistical data (Instrumentation)
    - Provides an interface to process administration and operation commands from an external management process (Command Callback)
    - Provides a standard interface to allow access and control application/service properties (Admin)
- Logging Integration—provides an easy access entry point to the Logging Service. From this log access, the developer can control the logging and the amount of log information to be propagated to the Logging Service.
- Event/Notification Integration—provides an easy access entry to enable the use of the Event/Notification Infrastructure.
- Generic Exception Facility—provide a default implementation of generic exception to interface with the Logging Service.
- Time Service Integration—provides an easy access entry to enable the use of the Time Service.
- Persistent Service Integration—specifies the factory, finder, load and store interfaces a developer needs to implement to. The underlying persistence framework is JGrinder. The developers implement the business objects using "wrapper" classes for the underlying JGrinder interfaces.
- Transaction Management— provides the common access to the session object in the JGrinder transaction framework. From this session object the developer can start a transaction, commit, and rollback transactional business objects. They can allow a thread to join or leave a session using the same exposed session object. The specific behaviors of these operations are details in the JGrinder documentation.

See *Foundation Framework Requirements* for more information on the Foundation Framework.

3.2.1.1 Components and Concepts

The following diagram details the overview components and concepts of how the Foundation Framework operates.

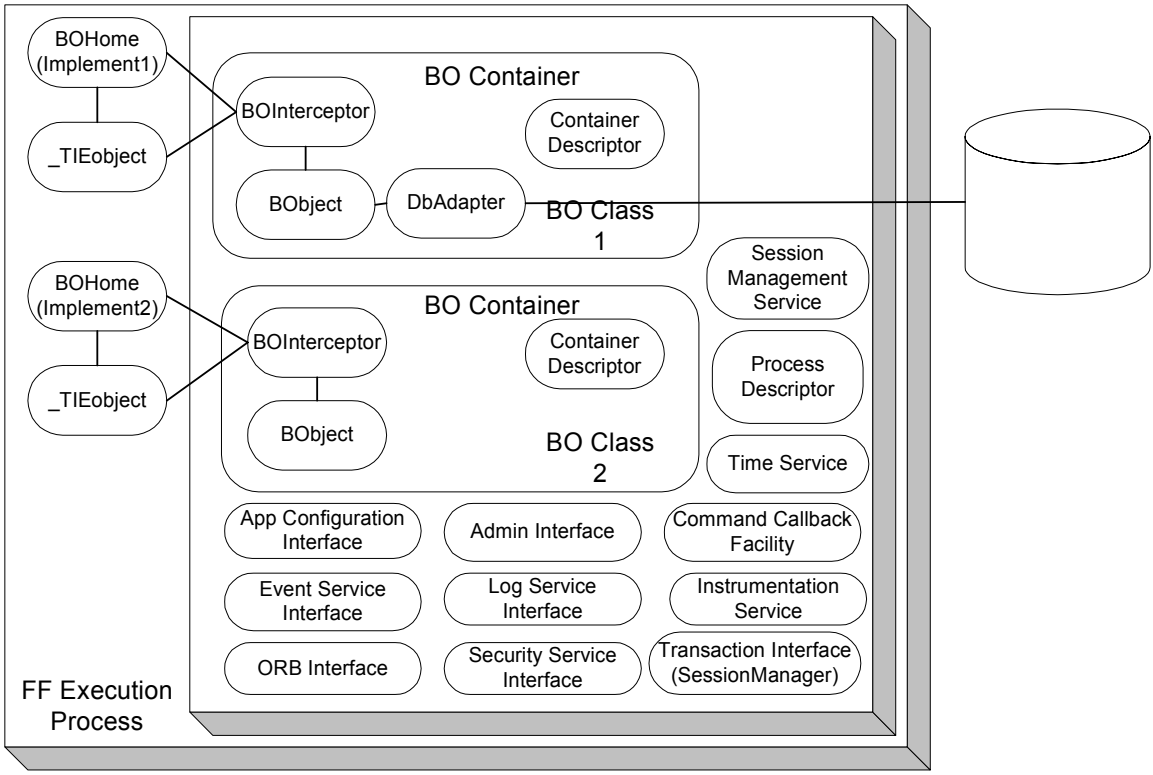


Figure 5. Foundation Framework Components Overview

3.2.1.1.1 Foundation Framework Execution Process

The Foundation Framework Execution Process is a singleton object that encapsulates all the infrastructure functionality and exposes an easy-to-use interface for the underlying infrastructure services. It contains a list of execution BOContainers, a Process Descriptor, the interface to the Application Configuration Service, ORB Service, Log Service, Event Service, Security Service, Session Management Service, Time Service, Instrumentation Service, Transaction Service (SessionManager), an Admin interface, and a Command Callback Facility. The developer simply instantiates the Foundation Framework Execution process in the main Java VM program to deploy business service objects to operate in the specified execution containers.

Each of the BOContainers provides a different execution container for a group of business service object instances (BObject). A service method invocation is always invoked on a BOInterceptor, a peer object to the BObject. The Foundation Framework code to implement the functionality is interjected at the

interceptor level and the actual work to service the request is delegated back to the BObject.

At this level, all the process level information that is shared between BO Containers is specified and exposed in the **Process Descriptor**. It contains:

- The process security identity—the identity of the whole process.
- A list of execution containers.
- A list of home implementations.
- An environment property provides the customizable name pair values, which can be defined and retrieved using the standard Properties object.

Within the process boundary, the framework exposes at this level the following interfaces:

- The Application Configuration Service interface – through this interface the start up default properties of the process can be accessed.
- The Security Service interface – provides the access to the Security Service
- The Session Management Service interface – provides the access to the Session Management Service
- The Transaction Service interface – provides the interface to the JGrinder SessionManager object. From this SessionManager, Session can be created to groups a number of threads together working on a set of transactional objects. The transactional objects will be commit and rollback together using session commit and rollback operations.
- The ORB Service interface – through this interface the basic Object Request Broker functionality like network parameters (request queue, time-out. The Trader service can be accessed through this interface.
- The Event Service interface – provides a simple access to the Event Service.
- The Log Service interface – provides the access to the Log Service. From this interface, the control of propagation level and the logging functionality is delegated to. The Logging Service defines the following log types: DEBUG, INFORMATION, ABORTION, THROTTLE, SYSTEM, RESOURCE and EXCEPTION. Each log type has ten levels of priority ranging from the one (lowest) to ten (highest). Along with the priority log level, a quality of service can be specified. The quality of service used to control the amount of messages to package and send at the sender type depends on the repetition of the message sent. Refer to the Log Service for the detailed quality of service.
- Instrumentation Service interface –provides the mechanism to actively collect the service statistical profiles.

The framework also exposes an Admin interface beyond the process boundary so that it can participate in the system control and management.

3.2.1.1.2 Business Object Container (BOContainer)

A business object container provides an execution container for a group of business service objects. Each service invocation is intercepted and controlled by the policies in the **BOContainer Descriptor**. The following policies are shared at the container level:

- A Transaction Policy: OBJECT_MANAGED and CONTAINER _MANAGED. With OBJECT_MANAGED, the business service object will make calls to the Foundation Framework Transaction Management directly. With CONTAINER_MANAGED, the transactional begin demarcation and commit demarcation will be provided at the peer interceptor object.

  Only the OBJECT_MANAGED is available for the first release of SBT. The framework will expose a JGrinder session object per container. The transactional object will manage its transaction begin and end explicitly.

- Thread Policy: REUSE_THREAD, SHARED_THREAD and THREAD_PER_REQUEST. The REUSE _THREAD policy executes the task in the same thread of the caller. The SHARED_THREAD policy uses the single shared thread in the execution container to process tasks one at a time. Incoming tasks are blocked in this model until the current task is finished. The THREAD_PER_REQUEST will execute the task in a separate thread. For each task, a new thread is created and the dispatching of the task happens within the new thread. Tasks are dispatched in parallel.

  Only the REUSE_THREAD is available for the first release of SBT.

- Instrumentation Policy: ON and OFF. ON policy starts the process of collection and instrumentation of the statistical profile data of service invocations. The running average timing and the number of invocation at the request level will be collected and stored in memory. OFF policy turns the processing off. The instrumentation data will be available when request and also available in the public broadcast event channel.

- Security Policy: PROCESS_IDENTITY and SPECIFIED_IDENTITY. The PROCESS_IDENTITY policy uses the process level security identity as the identity to validate the authorization. The SPECIFIED_IDENTITY policy allows the container to control the identity at the invocation level (in this case a client identity can be used instead of the process identity).

- Exception Policy: THROW_BACK, ABORT_CURRENT_TASK, and ABORT_PROCESS. TRHOW_BACK policy propagates the exception thrown to the caller. ABORT_CURRENT_TASK policy dictates the execution container to abort the current task, log the exception, and continue to the next task. ABORT_PROCESS policy allows the exception to be logged and other execution containers to be notified that shutdown is in process, and finally terminates the process.

  Since the Foundation Framework will not have any threading implementation for the first release of SBT, the only exception policy support is THROW_BACK. The developers have to code the EXCEPTION event log directly at the time of exception raise and it will be raised back to the caller the normal way.

3.2.1.1.3 Business Service Object Home (BOHome)

For each business service object class in the system, a Business Object Home is available to provide the finding and creation of a service object instance and its peer interceptor. In the case of implementing a CORBA object the TIE object can be managed by the BOHome as well. A Business Service Object Home is shared between the execution containers. There is only one Business Service Object Home per class of business service object in the VM process.

There can be multiple BOHome objects. Each one can handle the creation and finding of a selected implementation of a business service object. One BOHome object can be responsible for the CORBA proxy object (which in turn communicates with a server business service object). Another BOHome object can be responsible a local stubbed out version of a business service object; and one can take care of the actual business server service object at the server side. These multiple homes facilitate a consistent usage pattern and promote the iterative development effort.

3.2.1.1.4 Business Service Object Interceptor (BOInterceptor)

For each business service object instance (BObject), a peer service interceptor object is created. It is a one-to-one relationship between the peer interceptor object and the instance object. This peer object exposed all the service that a BObject provides. At this interceptor level all Foundation Framework features are provided. The interceptor delegates the actual work to the business service object instance.

3.2.1.1.5 Business Object Instance (BObject)

BObject is an instance of a business object. A Business Object is defined as a large-grained object that provides some business or application service, or represents a significant entity in the business domain.

There are two types of business objects. A service object is a large-grained object that provides some business or application service. There are a small number of service object instances in the process. The main process usually creates the service object instance at the initialization stage. The service object is running until the process is terminated. There is no activation and passivation of a BObject by the framework. The service object instance is typically a transient object with no persistent data. However, a service object can manage a collection of persistent objects.

Entity objects represent significant, small-grained domain objects (e.g. Order, Quote). These objects are typically persistent, outliving the process in which they were created, with cardinality greater then one.

When implemented with the JGrinder persistence framework, a JGrinder DBAdapter will be associated with the object to provide the persistence capability. There are different DBAdapters to persist data to different persistent medium: one for a relational database, one for a text file, and one for a binary file. When implemented with the JGrinder Transactional interface, a BObject will implement the interface to support commit and rollback of the data (the in process memory data attributes)

A business object can be a CORBA IDL implementation object or an object intended to be used exclusively within a process.

3.2.1.1.6 Service Invocation Dispatching

When the service invocation (a business method) arrives to the peer interceptor object, the interceptor accepts the invocation. Before submitting the service invocation and processing the reply invocation, the interceptor object enables the Foundation Framework functionality.

To enable the Foundation Framework functionality, the service invocation only needs to go through the interceptor object. To the caller, the syntax and calling signature is the same between the service instance and its interceptor. If the service object is a CORBA object, use the interceptor object as the implementation object instead of the service object to inherit the Foundation Framework functionality.

When using the TIE approach to provide the CORBA object, the tie object (server side IDL generated object) can be associated with the interceptor object to enable the Foundation Service functionality the same way. Dispatching the CORBA request to the interceptor using the TIE mechanism and the dispatching between the interceptor and the instance object should proceed normally.

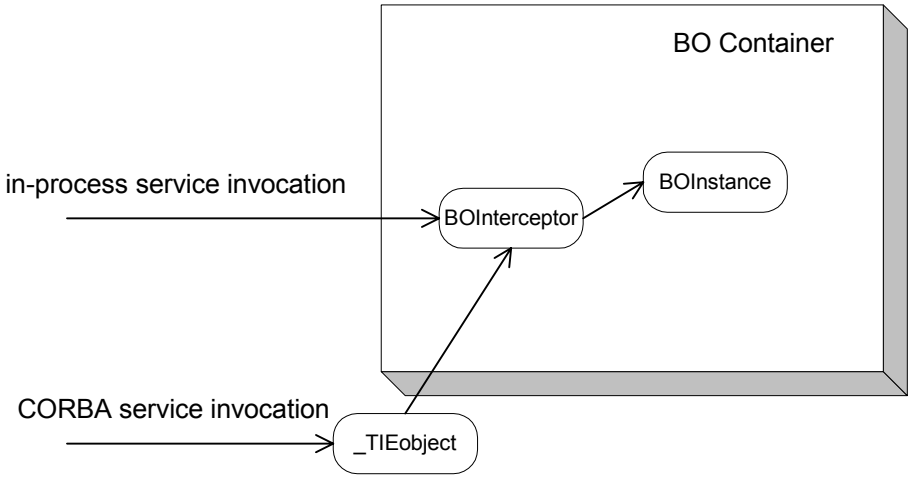


Figure 6. Service Invocation

3.2.1.1.7 Application Process Management

At the process level a **Command Callback facility** is exposed. It contains a map of registered callbacks, allowing the Foundation Framework to enable the process to be controlled and managed from an external process. A developer register "command" as a string key name to an action at initialization stage and the call back will be invoked from an IDL **admin interface**.

An external management process binds to the admin interface to retrieve process configuration information, to change process behaviors, and to retrieve the statistical instrumentation profile data.

3.2.2 Internal Event Channel

The internal event channel is a pattern that is based on Java custom events. For each type of custom event that can be registered, a model (*EventChannelListener)* can be added as a listener to the Event Channel. When a remote event is received, it is routed to the correct channel and all listeners of that channel are notified of the event and passed any user defined data that wants to tag along with the event. The internal event channel exists in its own thread and has a queue.

3.2.3 Message Translator

The API (Message) Translator is responsible for hiding the complexities of communication with the CAS from the GUI and Models. All remote requests by the GUI go through the translator. The translator will take a request for data, register the Event Channel Listener, and create and pass the callback listener. It also keeps track of the user data so that the GUI does not need to supply it when making remote calls.

3.2.4 Thread Pool Framework

The Thread Pool Framework provides a lightweight, asynchronous implementation of the Command pattern. The framework accepts commands, implemented as subclasses of an abstract ThreadCommand class, and dispatches them to the next available worker thread. If no thread is available, the command is queued until a thread becomes available. Upon completion, the command can be programmed to update zero or more objects, simulating a return value.

The framework is used within the Broker Service for order and quote processing, as well as the Internal Event Channel.

3.2.5 Callback Framework (Observer Pattern)

The GUI Client Application is a thin client that only provides the presentation and the input controller of the SBT Application Services. The observer pattern is nicely fit to provide the presentation need for the GUI Client Application. Each of the object data can be mapped to form a one-to-many dependency with the GUI widgets. When the object data changes state, all the GUI widgets are notified and updated automatically.

When a remote method is called on the CAS/CMi, an optional callback object can be passed which then registers the Presentation Services for any update "callback" messages from the CAS/CMi. There is one listener for each type of message that can be received. When a listener receives a message from the CAS/CMi, it will forward the message to the appropriate "Internal Event Channel" where it will then be forwarded to any interested event channel listeners. Each callback listener lives in its own queued thread.

Within the CAS, the callback suppliers are responsible for sending updated data through the CBOE Market Interface to a remote listening application. Each type of callback has its own supplier object. Similar to the Internal Event Channel used by the Presentation Services, a callback object is registered with a

particular channel or set of channels. Each callback object then serves multiple remote callback clients and manages which types of data to send to which remote clients. The callback supplier object is threaded and queuing. Synchronous callbacks are used to communicate remotely to insure that the Presentation Services layer receives the message.

3.2.6 Adapter

The intent for the adapter pattern is converting the interface of a class into another interface clients expect. In SBT all the external interfaces can be grouped into this category (e.g. TIPS Adapter, TPF Adapter). An object wrapper is created for data that needs to be accessed from the external system. The object wrapper converts requests to external native function calls to fulfill the request.

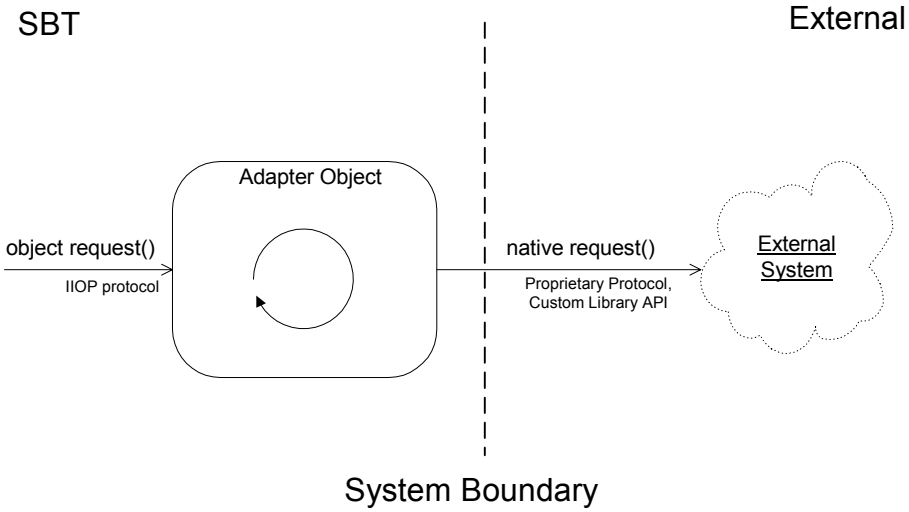


Figure 7 Adapter pattern

3.2.7 Static/Dynamic Dispatching (Dispatcher/Concentrator/Broker)

In a dispatching pattern, fault-tolerance and scalability of service are addressed by providing multiple registered services with a dispatcher. The dispatcher receives a service request and dispatches the request to the registered service. More than one instance of service can register to provide reliability and/or load balancing. The information needed to dispatch the request to a pool of services is available in the Trader Service (all the services advertise their capability using service type and properties with the Trader Service.) The concentrator uses the information in the Trade Service to route an incoming request to the appropriate object. The mapping process can be lazy instantiated (only maps out the service that is required at the time of execution) or fully instantiated for performance depending on the service usage pattern.

The Dispatcher can use a simple round robin algorithm for load balancing among the registered services. This method of load balancing is appropriate when the service is stateless, and any instance of the service can satisfy the request. An alternate strategy is used when the service instances are not identical, or contain

state that must be retained from one invocation to the next. In this case, the concentrator must use some mechanism to determine which service instance can handle the request. An example is a concentrator for the Order Handing Service. There may be many instances of the service; each instance assigned a subset of the all the products. This approach maximizes load balancing and reliability by partitioning the products across a number of service instances, typically running on different platforms. The concentrator is responsible for determining the correct service instance for an incoming request. This is accomplished by examining one or more fields/parameters in the request, along with information obtained from the Trader Service about the properties of the service instances, to determine the appropriate destination.

With static dispatching, the client requests an instance of the service from the dispatcher. All further communication is performed directly between the client and the service. This style of dispatching is good for a stateful connection between a client and a service provider. With dynamic dispatching, the client always sends the request to the dispatcher and relies on the dispatcher to relay the message. This style of dispatching is better at stateless communication between a client and a service provider.

Routing Proxies utilized by the Front End hosts implement the dispatcher pattern, forwarding product-specific requests (e.g. orders) to the appropriate business server, locating business objects (e.g. orders) when a product is unknown, and coordinating federated queries which require all business servers (e.g. getAllOrders).

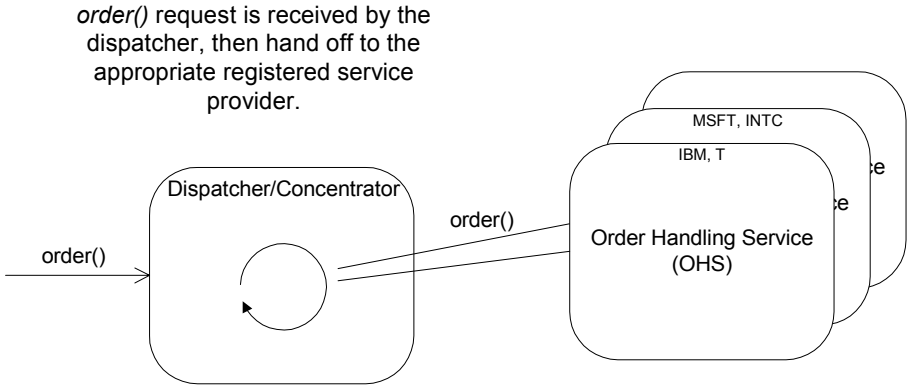


Figure 8 Dispatcher pattern

3.2.8 Smart Proxy

A standard proxy provides a placeholder for another object to control access to it. A typical usage is in a distributed application where a local proxy for a remote object is provided to a client. The proxy appears to the client as the remote object, forwarding requests to the remote object and returning results to the client.

A smart proxy extends the standard proxy with additional capabilities. Within SBT, smart proxies can be used to provide transparent fault tolerance to clients.

Each smart proxy will contain references to two remote services (e.g. Order Handling Service). One reference will be designated as a primary, and the other as a secondary, or backup. Upon receiving a request, the smart proxy will attempt to forward the request to the primary reference. If a failure occurs, the smart proxy can transparently retry the request, or attempt to use the secondary reference. The number of retries can be configurable. The smart proxy can be implemented to always try the primary reference first, or completely bypass the primary after a number of failures. If a failure occurs due to a lost connection, the smart proxy can attempt to reestablish the connection on the primary reference while servicing the request through the secondary reference.

Use of smart proxies simplifies the development of client applications, by isolating complex error handling and fault tolerance code in a common implementation.

The smart proxy object can optionally cache the data for performance purpose.

The CAS uses a smart proxy for each Business Service (Order Handling Service, Order Book Service, Market Data Service, Market-Maker Quote Service, and Product Service). In addition, the Product Service proxy implements caching, which reduces the number of calls to the remote service. Also, each server process is configured with smart proxies for all services that reside out-of-process. For example, the Trade Server and the TPF Servers use caching proxies for the Product Service.

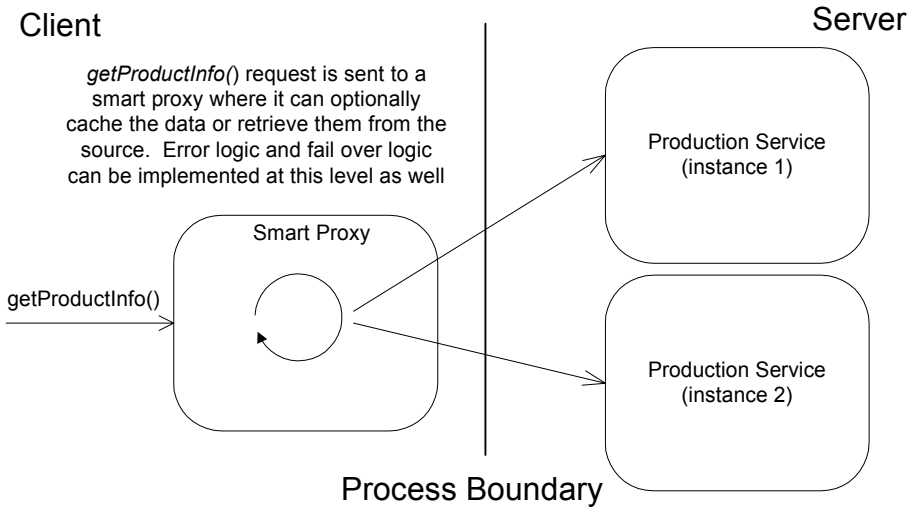


Figure 9 Smart Proxy pattern

3.2.9 Snapshot and Changed Events

Caching data on the client is an excellent way to increase system performance by reducing the number of calls to the server to retrieve data. The difficulty lies in ensuring that the cached data accurately reflects the state of the data on the server. There is little difficulty when the data is read-only. The data can be retrieved at one time (snapshot), or fetched on an as-needed basis (lazy instantiation).

When the cached data can potentially change at the source level (server), a notification mechanism needs to be in place to propagate the changes to the client. Given the SBT environment, a solution can be provided using event infrastructure to propagate a changed event (the delta data) to the local cached object. Every snapshot and changed event has to be time stamped or sequenced before sending out. At the client side, the proxy object should start subscribing to the changed event before getting the snapshot. A snapshot is then retrieved and the proxy start applies the delta change to the local snapshot. The changed event that arrived before retrieving the snapshot should be discarded. This approach will ensure that the cached object is synchronized with the object on the server.

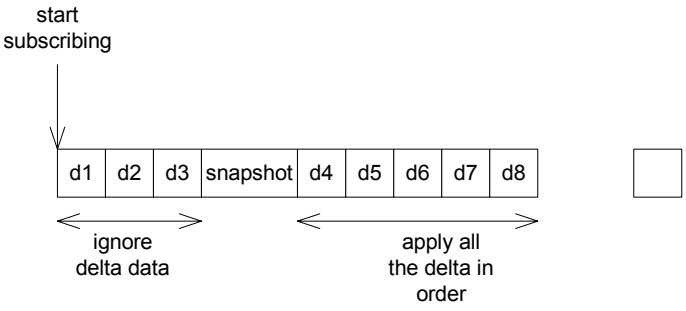


Figure 10 Snapshot and Changed Events

The CAS uses the snapshot pattern for maintaining the consistency of orders and quotes. The Order Status Service is used to propagate changes (e.g. fills, cancels) to the CAS whenever the actual object residing on the server changes.

3.3 External Interfaces

The SBT interfaces to the following systems:

- TPF – provides the wire orders and product data (options only), and processes quote data. Also accepts orders for open-outcry products entered through the CMi.
- COMPASS – provides wire orders (single stock futures only).
- TIPS – provides the market data: underlying price, news alert and Best of the Rest.
- CTM – processes all trade reports.
- COPP – processes outgoing market data.
- Membership System – provides the membership and firm data
- Member's System – member's proprietary systems will interface with the SBT system using the Client API. The interface will provide all capabilities of the CBOE provided GUI, which will allow member firms to develop their own user interface and/or back office system.

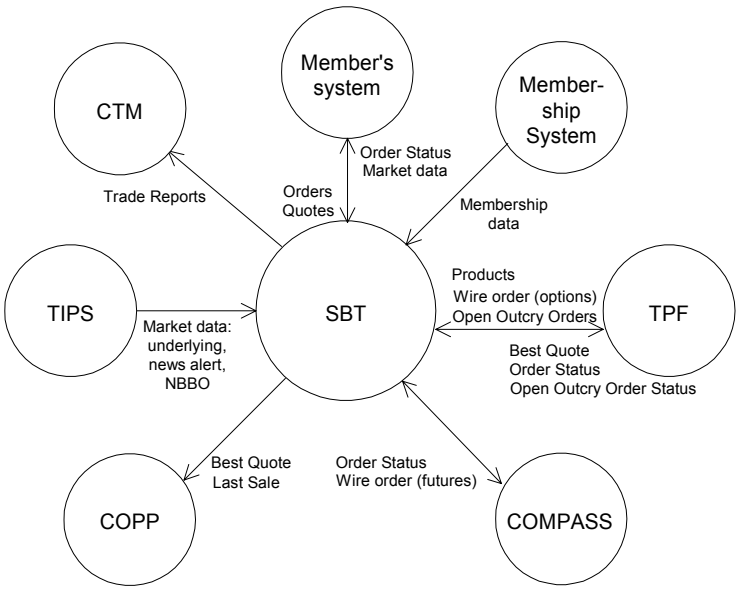


Figure 11 SBT External Interfaces

3.4 User Interfaces

This Presentation Services (GUI) package contains two GUI applications, one for each of the two major classes of human actors that use the SBT System: traders and system operators.

- Trading GUI – is used by regular traders, and market-makers. It consists of several GUI's for displaying and entering orders, quotes, and market data.
- System operation GUI – is used by system operators and help desk operators

Packaging the GUIs in single applications shared by all users in a class is intended to reuse a substantial number of common GUI components and maintain a consistent look and feel.

In the context of the Model-View-Controller design pattern, the SBT GUI applications act as views and controllers, while the applications in the Application Services Package, described in section 3.1.2, act as models. A GUI framework is implemented here to promote the consistency in look and feel, and speed up the GUI development work. The SBT GUI will utilize the CBOE Market Interface (CMi) Application Programming Interface (API).

The following are general design requirements or principles that have to be considered in designing the various components of the SBT system.

- The user interface is designed to be flexible and configurable by the user. The user is able to specify the columns of data he/she wants to see in displays such as the Market Display, the Order Status Display, the Trades and Quotes Log, the AutoQuote Setup Display, etc. The user is also able to specify the background colors, the font size, etc.
- The use of popup windows is minimized to provide the trader a constant, unobstructed view of his/her trading screens.
- Speed in interactions with the system is of the utmost importance. Mouse and keyboard operations for each transaction, especially the most frequently used transactions, are kept to a minimum.
- Keyboard operations equivalent to the mouse operations for certain critical transactions, for example, Delete All Quotes transaction, are available for use in case of mouse failure.

A prototype of the SBT system was developed to assist in defining the requirements and look-and-feel of the user interface. The prototype interfaces served as the basis for the design of the SBT user interfaces. The individual user interfaces are detailed in the set of *CBOE Use-Case Reports*. Examples of two SBT GUI windows are given below.

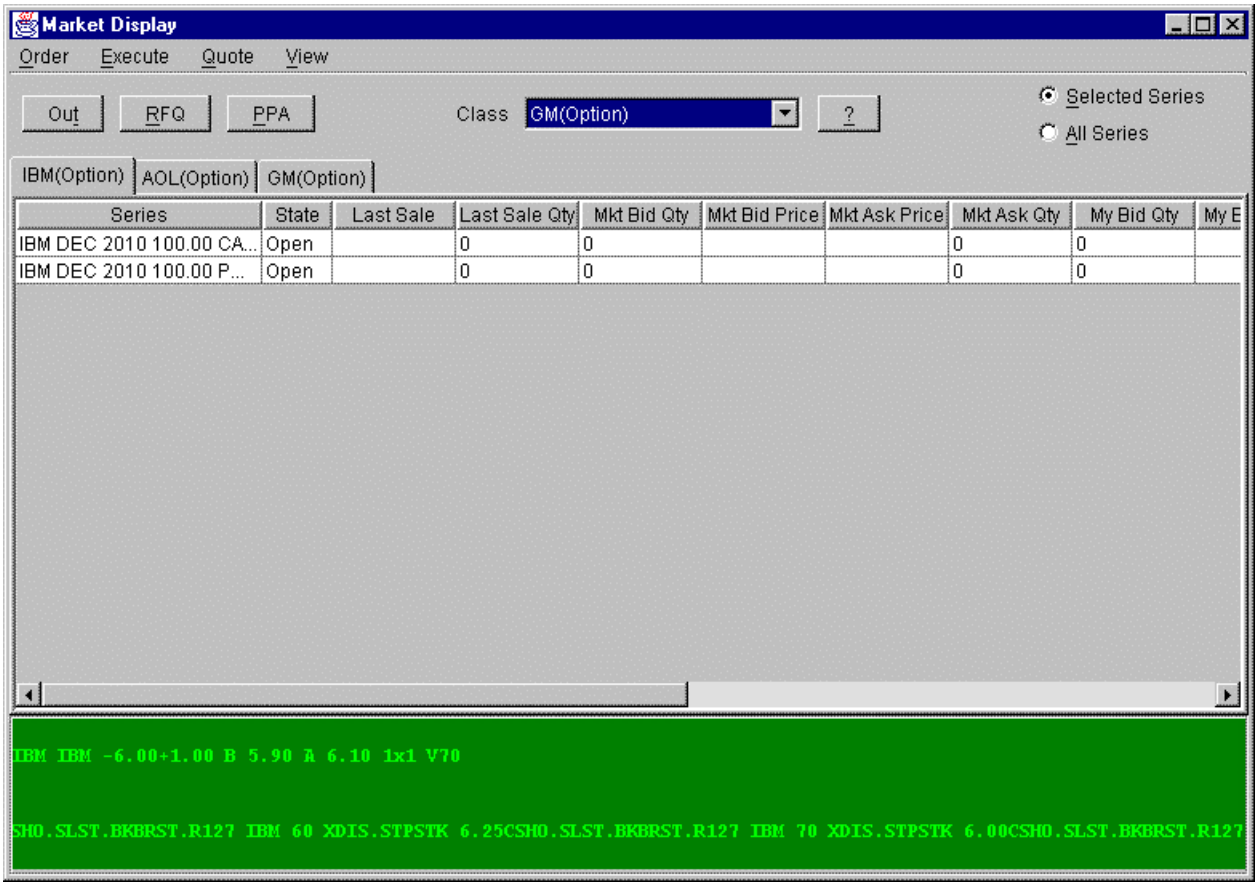


Figure 12 Market Data Window

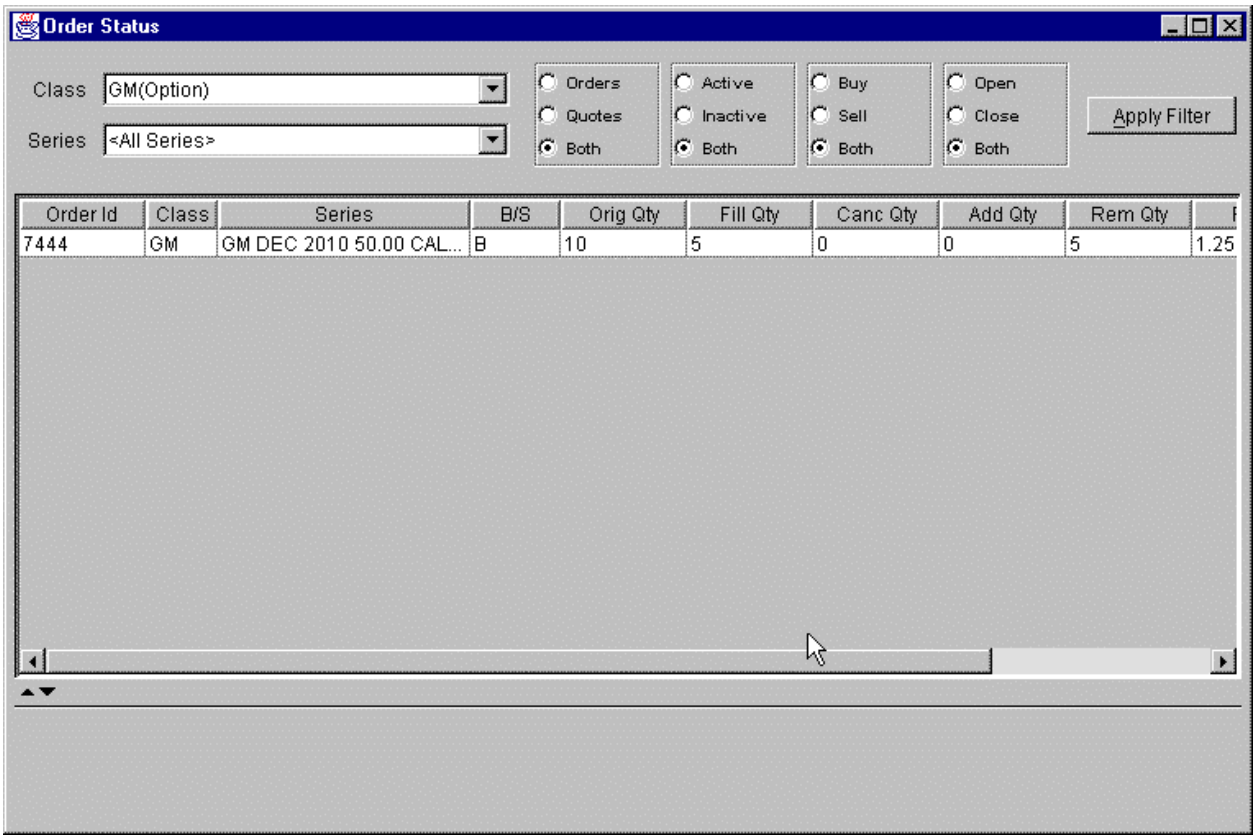


Figure 13 Order Status Window

3.5 System State

The system will support the following states of operation for each product class. The system should be designed to support both manual (i.e. administrator controlled) and automatic state transitions at the appropriate time (i.e. system controlled).

- Pre-opening
- Opening
- Trading
- Halted
- Closing
- Post-Closing (End of Day Processing)

3.5.1 Pre-opening

The system will accept orders and quotes at this state, but no trading will take place. The system will display on Market Displays any resting orders from the previous day and orders submitted before the opening. See section XIII.A of the *SBT Functional Requirements* for the pre-opening procedure.

3.5.2 Opening

When the primary market disseminates the underlying security's opening trade or opening bid and ask, the class goes into the Opening state. The system will send out an Opening Notice to Market-Makers who are assigned to that class. The system will continue to accept orders and quotes at this state. At the end of this opening time period, the system will establish an opening price for each series, complete the opening trade, if any, and then change the state of the class to Trading. See section XIII A of the *SBT Functional Requirements* for a more detailed description of the opening procedure.

3.5.3 Trading

At this state the series will trade freely. Orders and quotes will be accepted.

3.5.4 Halted

At this state trading is halted. The most common reason will be the primary exchange has halted trading of the underlying security, or no underlying security prices or quotes are being received. The system will send OPRA the appropriate quotes for a product that is halted. At this state orders are accepted by the system. The product will have to go through the pre-opening and opening procedures before it reverts to the state of Trading.

3.5.5 Closing

The system changes the state to Closing at a pre-determined time period, e.g., 1.5 minutes, before the closing time. Orders and quotes will be accepted until the system changes the state to Post-trading at closing time.

3.5.6 Post-Closing (End of Day Processing)

Orders and quotes will not be accepted at this state. The system will perform its post-trading functions such as the passing of eligible SBT orders to the Regular Trading Hours (RTH) book and other systems, purging of SBT day orders, and reporting of Nothing Done order status to member firms. At the end of RTH trading and before SBT goes into Pre-opening, eligible RTH orders in the RTH book will pass into the SBT book.

3.6 Significant Scenarios

This section provides an overview of several important scenarios (functions) that the SBT System supports. It is not intended to be a substitute for the more detailed descriptions found in the Analysis Model, but only provides the background and context for the rest of the Application Architecture.

3.6.1 Enter Order

The System accepts orders from an actor for execution. The System accepts the following types of orders: Limit, Market, All or none (AON) , Fill or kill (FOK), Immediate or cancel (IOC), Stop (STP), Stop Limit (STP LIMIT), and Spread. Each type of order has a specific set of execution rules. Upon receiving an order, the system attempts to execute it according to the execution rules of its type. Depending on the execution rules and whether or not the order was (partially) filled, this UC may trigger (use) other use cases, including Display Fill Report, Disseminate Trade Report, Send RFQ, etc.

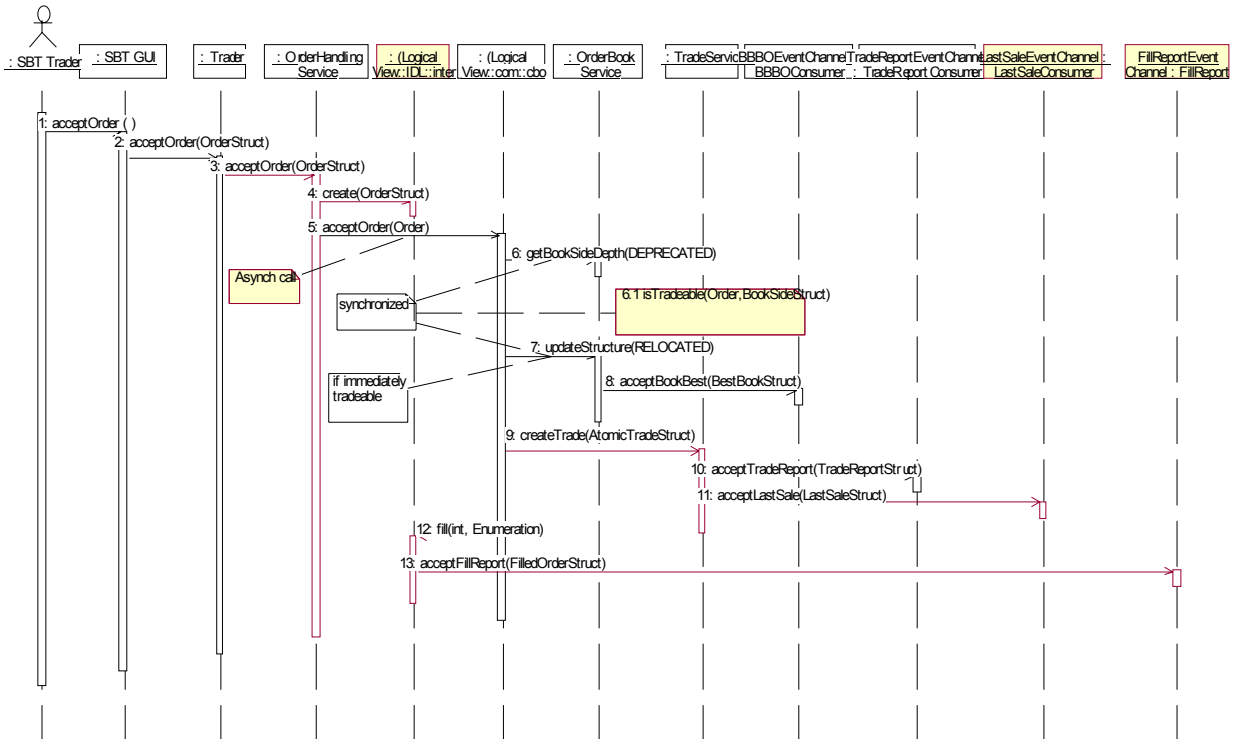


Figure 14 Enter Limit Order

3.6.2 Display Fill Report

Upon filling an order (partially or completely), the System reports the fill to the originating actor.

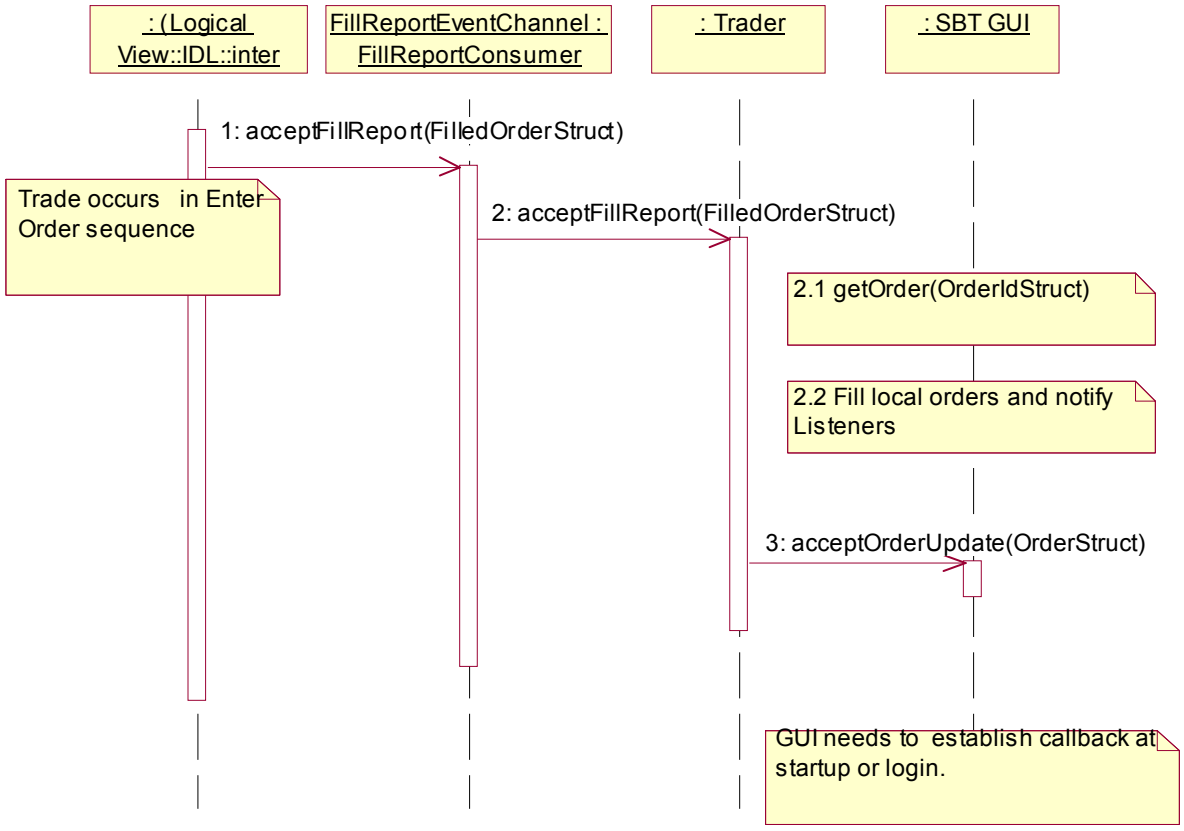


Figure 15 Display Fill Report

3.6.3 Display Book Depth

A market-maker may request a display of the book depth of a selected product. This display is only a snapshot that needs to be refreshed manually. The ability to dynamically refresh the book depth display is also available.

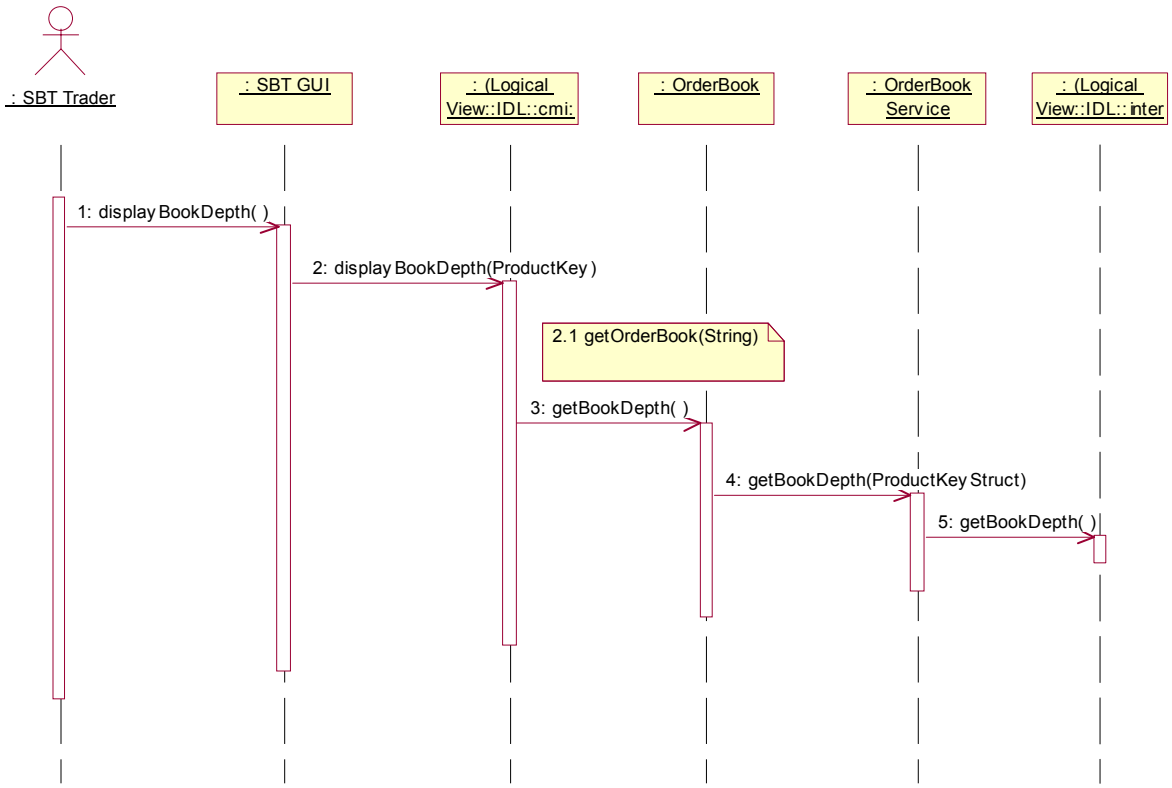


Figure 16 Display Book Depth

3.6.4 Disseminate Fill Report to TPF

The System sends Fill Report(s) to TPF each time it matches an order originating in TPF. The report(s) contains trade information for all the parties to the trade.

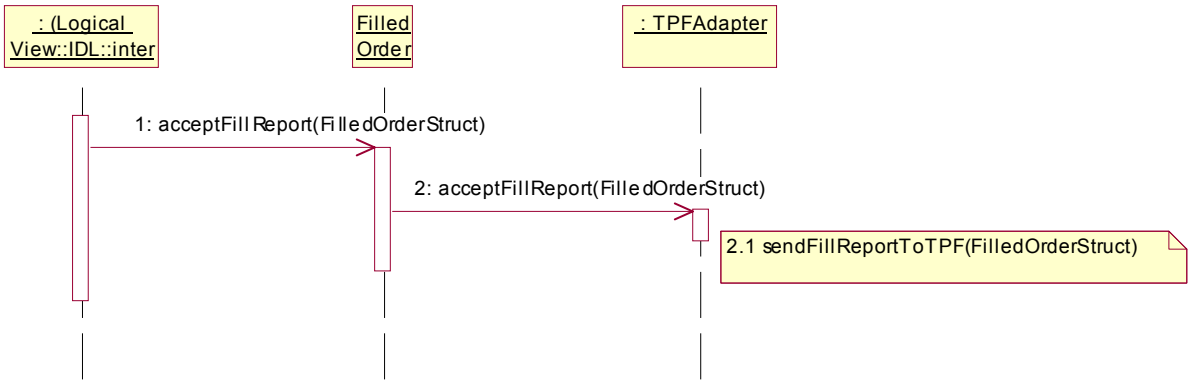


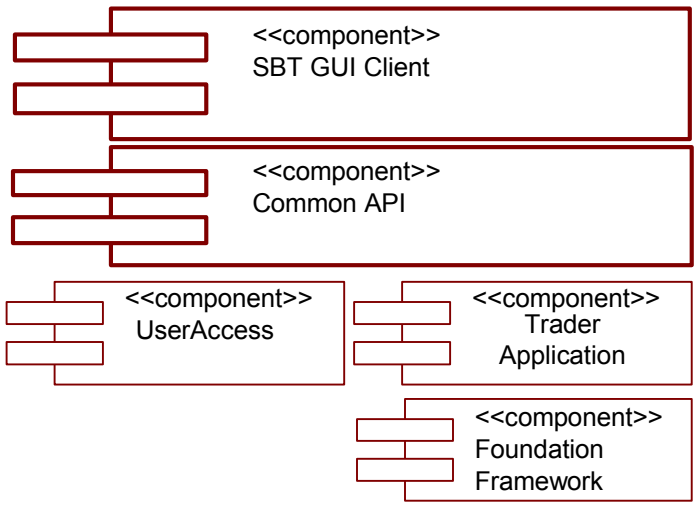
Figure 17 Disseminate Fill Report to TPF

4. Process View

SBT is a distributed application based on an Abstract Integration Model. The fundamentals: uniform type, reference, and invocation system in a distributed environment are the key ingredients to the integration view. The Common Object Request Broker: Architecture and Specification (CORBA) is selected as the technology for the Abstract Integration Model. It provides the basic elements: type, interface, object, reference, operation, parameter, context and exception needed to define a flexible architecture that is able to adapt to changing technology. Using CORBA, business services are implemented as CORBA object servers. The business data and functionality are described in a development language neutral syntax called Interface Definition Language (IDL). The business objects are designed to be deployed in a distributed environment for performance, scalability, and high availability. The following process views represent the decomposition of SBT components in terms of execution flows (process and threads of task), the synchronization between flow, and the allocation of components within the various flows.

The process grouping below is established and is guided by the frequent interaction between the components. It is leaning toward minimal physical I/O transaction and network traffic. Some of the independent process should be analyzed further to see if they can be deployed together in a single process to minimize the system usage (physical memory, virtual memory, swap space, paging, etc.) and network usage (socket, connection to global service server, etc.).

4.1 Trader Workstation (GUI and Client Application Server)



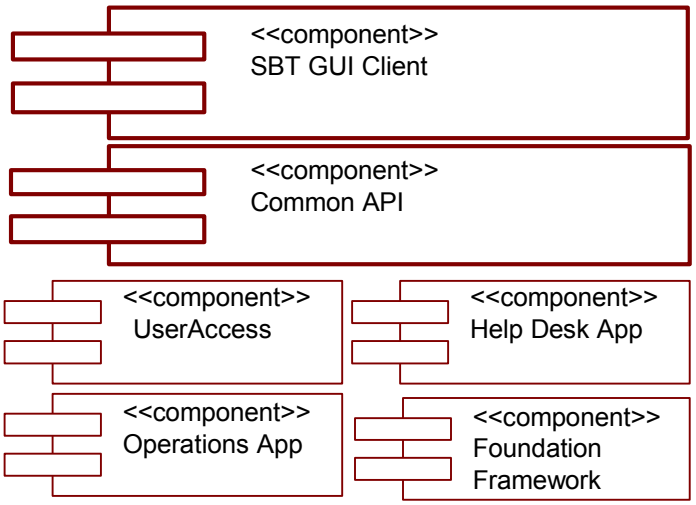
The Trader workstation supports the market-maker, broker, class display, and firm roles. Operations are restricted to those defined for the role.

The SBT GUI Client component provides the interaction mechanism between the Trader Workstation process(es) and the SBT system. The Trader Workstation is comprised of two parts: a GUI, and a Client Application Server (CAS). The two components are defined as CORBA objects, and as such, can be executed in a single process or multiple processes. In the two process model, the CAS is started first, and publishes an Interoperable Object Reference (IOR) for the SBT UserAccess object in a location that the GUI can access. (Note: the CAS currently incorporates a web server that provides the IOR via an HTTP request). When the GUI process starts, it obtains the IOR for the UserAccess object, and opens a GUI screen prompts the user for logon information. The entered information passes through the Common API component to the UserAccess component. Upon validating the authentication and authorization, the CAS returns a reference to the UserSessionManager object (i.e. Trader App), and starts

- receiving and processing Trade commands through the SBT GUI component
- monitoring and processing events from the Event Infrastructure

The Trader Workstation process contains a trader view of the orders and the top of the book (Current Market). The view of the data should be retrieved when the process is started up and cached locally for performance. The view needs to be updated as the data changed at the business services. There is no need for transaction management in this process.

4.2 Operator/Help Desk/Product Maintenance Workstation



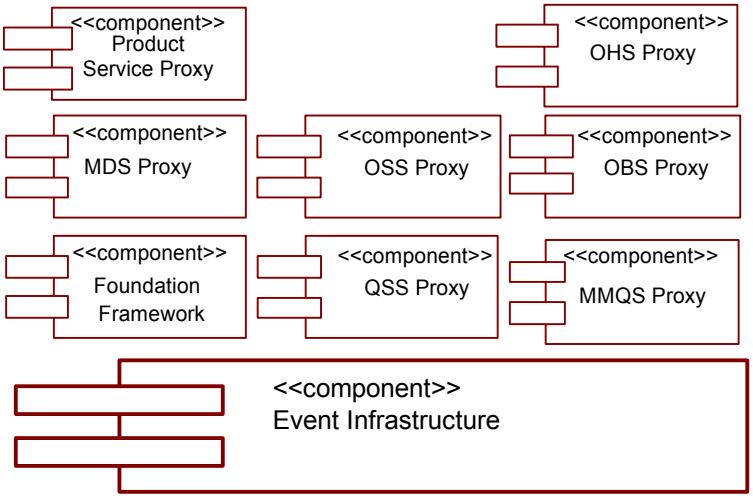
The SBT GUI Client component provides the interaction mechanism between the Operator/Help Desk/Product Maintenance Workstation process(es) and the SBT system. The workstation is comprised of two parts: a GUI, and a Admin Client Application Server (ACAS). The two components are defined as CORBA objects, and as such, can be executed in a single process or multiple processes. In the two process model, the CAS is started first, and publishes an Interoperable

Object Reference (IOR) for the SBT UserAccess object in a location that the GUI can access. (Note: the CAS currently incorporates a web server that provides the IOR via an HTTP request). When the GUI process starts, it obtains the IOR for the UserAccess object, and opens a GUI screen prompts the user for logon information. The entered information passes through the Common API component to the UserAccess component. Upon validating the authentication and authorization, the CAS returns a reference to the UserSessionManager object (i.e. Help Desk App), and starts

- receiving and processing Help Desk, Operations and CFB commands through the SBT GUI component
- monitoring and processing events from the Event Infrastructure

The Operator/Help Desk Workstation process(es) contains all the administration and operation views of the data. The views of the data are retrieved when needed and cached locally for performance (most of the data for this view are read-only for the day). There is no need for transaction management in this process.

4.3 Front End (Routing) Server



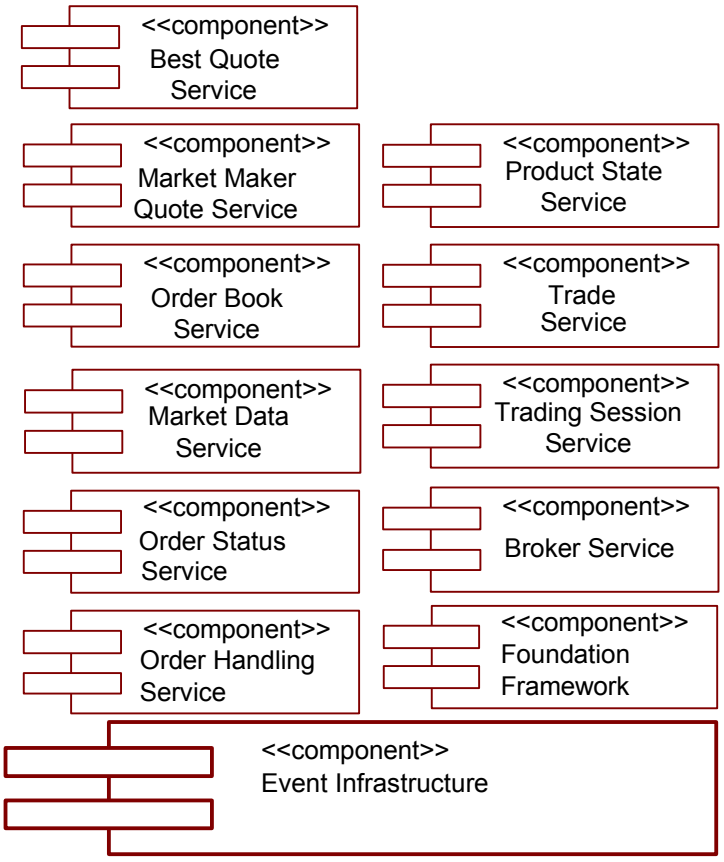
The Front End Server process starts when a system administrator manually brings up the server or it is started by a monitoring process. When started by a monitoring process, the process relies on the Foundation Framework to set up the threading policy, to establish connection with the Infrastructure Services , and to set up itself for system management. The process then initializes the routing proxies (concentrators) for the various Business Services (e.g. Product Service, Order Handling Service, Order Book Service, Order Status Service, Quote Status Service, Market-Maker Quote Service, Market Data Service, etc.), and prepares to accept requests. The process goes to the normal server operation mode to

- receive and process service entry requests

- receive and process Business Service requests from the Client Application Server client.
- monitor and process Application Management requests

There is no stateful information about the connection kept at the Front End Server process.

4.4 Distributed Trade Server



The Distributed Trade Server process starts when a system administrator manually brings up the server or it is started by a monitoring process. When started by a monitoring process, the process relies on the Foundation Framework to set up the threading policy, to establish connection with the Infrastructure Services, and to set up itself for system management. Next, each of the business components (OHS, Trade Service, Order Book Service, Broker Service and MM Quote Service) goes through its own initialization, preparing to accept requests. The process finally goes to the normal server operation mode to

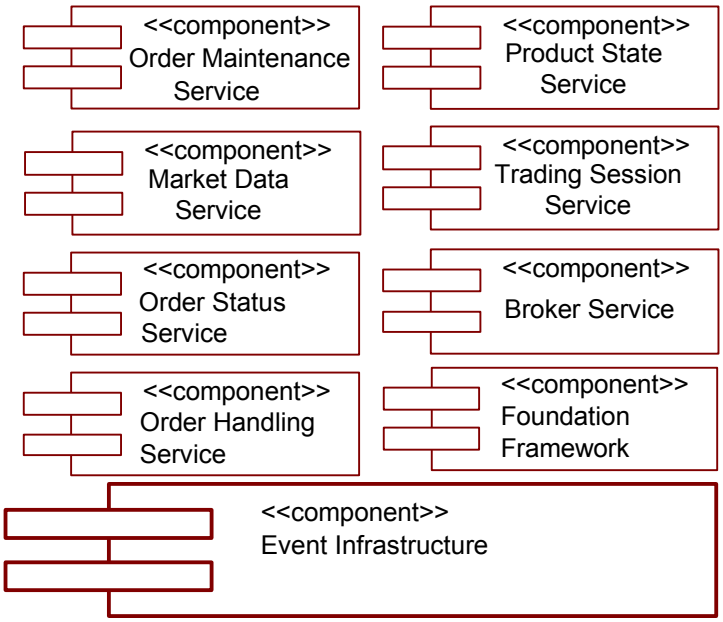
- receive and process orders from the Trader Workstation process
- monitor and process wire orders from the Wire Order Server process

- receive and process MM Quote Service requests
- receive and process Order Book Service requests
- monitor the Event Infrastructure for Best Book Bid Offer (BBBO), and Best of the Rest, and disseminate Current Market and NBBO information via the Event Infrastructure.
- monitor and process Application Management requests

The collaboration of business services components for this process is highly active. There is need for transaction management at each of the components. The purpose for the grouping of them on a same process is to minimize the physical I/O transaction with the data repository and eliminate the need for the distributed transaction coordination.

Because of the nature of the business, the processing of a single product (i.e. series) is synchronous. For example, an IBM Jan Call 125 order request has to be completed before processing the next IBM Jan Call 125 request. Different product requests can be processed simultaneously.

4.5 External Trade Server



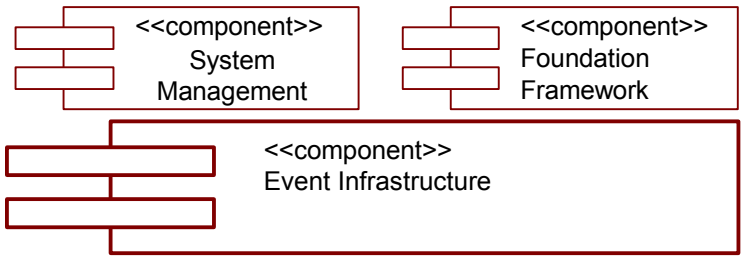
The External Trade Server process starts when a system administrator manually brings up the server or it is started by a monitoring process. When started by a monitoring process, the process relies on the Foundation Framework to set up the threading policy, to establish connection with the Infrastructure Services, and to set up itself for system management. Next, each of the business components (OHS, and Broker Service) goes through its own initialization, preparing to accept requests. The process finally goes to the normal server operation mode to

- receive, process, and route (RTH) orders from the Trader Workstation process to TPF via the TPF Adapter
- monitor and process Application Management requests

The collaboration of business services components for this process is highly active. There is need for transaction management at each of the components. The purpose for the grouping of them on a same process is to minimize the physical I/O transaction with the data repository and eliminate the need for the distributed transaction coordination.

Because of the nature of the business, the processing of a single product (i.e. series) is synchronous. For example, an IBM Jan Call 125 order request has to be completed before processing the next IBM Jan Call 125 request. Different product requests can be processed simultaneously.

4.6 Systems Management Server

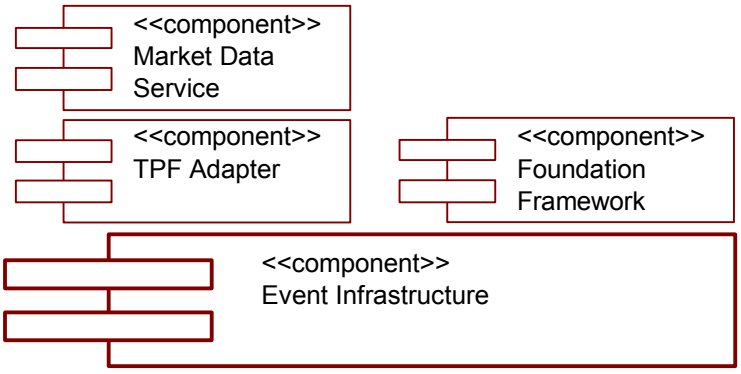


The System Management Server process starts when a system administrator manually brings up the server or it is started by a monitoring process. When started by a monitoring process, the process relies on the Foundation Framework to set up the threading policy, and to establish connection with the Infrastructure Services. The process then, initializes the System Management components and prepares for accepting requests. The process goes to the normal server operation mode to

- receive and process system management requests from the Administration/Help Desk application
- monitor and process events from the other applications through the Infrastructure Services

The Systems Management Server process needs the normal transaction management to retrieve and store application configuration data stored in collections of Managed Objects.

4.7 Market Data Service

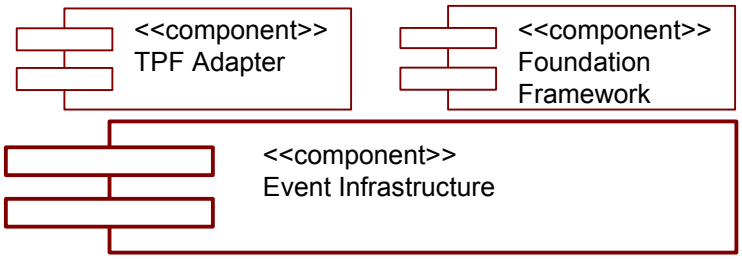


The Market Data Server process starts when a system administrator manually brings up the server or it is started by a monitoring process. When started by a monitoring process, the process relies on the Foundation Framework to set up the threading policy, to establish connection with the Infrastructure Services, and to set up itself for system management. The process then, initializes the TPF Adapter component and prepares for accessing market data (MDR) from TPF. The process goes to the normal server operation mode to

- monitor and process Market Data Service requests from the workstation process
- monitor and process Application Management requests

The Market Data Server process needs normal transaction management to retrieve and store historical Market Data in a certain caching option.

4.8 TPF Adapter Server

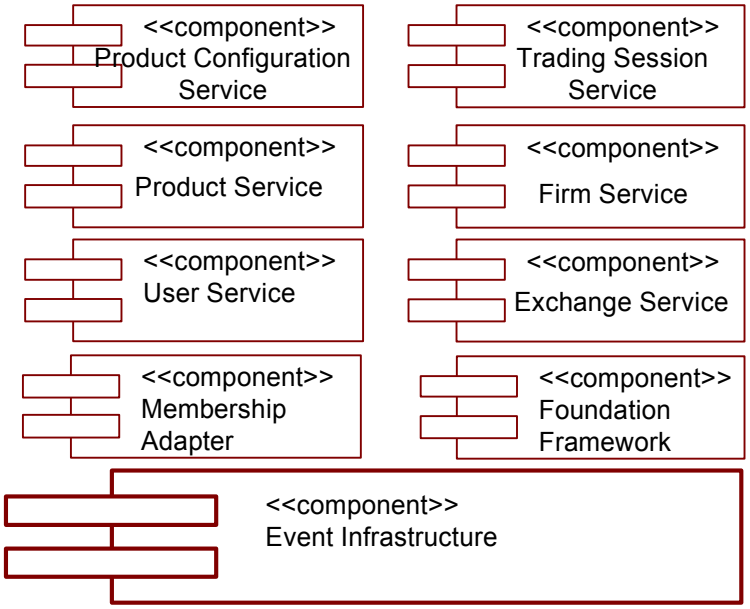


The TPF Adapter Server process starts when a system administrator manually brings up the server or it is started by a monitoring process. When started by a monitoring process, the process relies on the Foundation Framework to set up the threading policy, to establish connection with the Infrastructure Services, and to set up itself for system management. The process then establishes the connection between the TPF/Host Gateway and the TPF adapter. The process goes to the normal server operation mode to

- monitor and process incoming wire order requests from TPF
- monitor and process wire order status events from the Event Infrastructure

The wire order data is managed at the TPF side. Order requests are processed as they come to the process. There is no transaction and recovery needed at this process.

4.9 Global Server

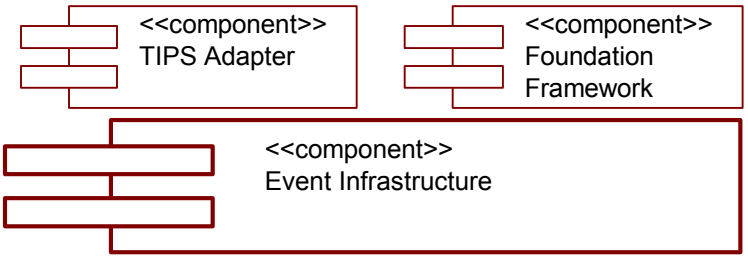


The Global Server process starts when a system administrator manually brings up the server or it is started by a monitoring process. When started by a monitoring process, the process relies on the Foundation Framework to set up the threading policy, to establish connection with the Infrastructure Services, and to set up itself for system management. The process then, initializes the Membership Adapter component and prepares for retrieving data from the Membership System. The process goes to the normal server operation mode to

- monitor and process User, Firm, and Exchange Service requests
- monitor and process Product Service requests
- monitor and process Application Management requests

The Global Server process needs the normal transaction management to retrieve and store user, product, firm, trading session, and exchange data.

4.10 TIPS Adapter Server

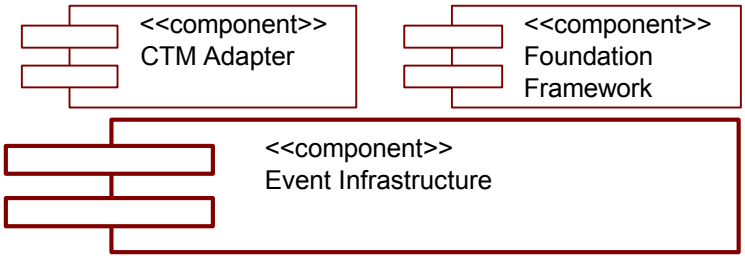


The TIPS Adapter Server process starts when a system administrator manually brings up the server or it is started by a monitoring process. When started by a monitoring process, the process relies on the Foundation Framework to set up the threading policy, to establish connection with the InfrastructureServices, and to set up itself for system management. The process then, initializes the TIPS Adapter component and prepares for accessing data from TIPS. The process goes to the normal server operation mode to

- retrieve underlying data and news alerts from TIPS and disseminate the information using the Event Infrastructure
- monitor and process Application Management requests

The TIPS Adapter Server process keeps no stateful information about the underlying data and news alerts as they pass through the process.

4.11 CTM Adapter Server

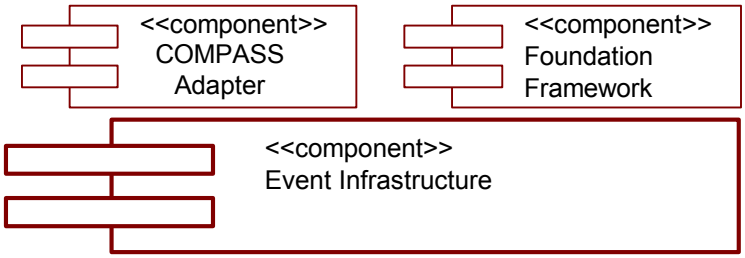


The CTM Adapter Server process starts when a system administrator manually brings up the server or it is started by a monitoring process. When started by a monitoring process, the process relies on the Foundation Framework to set up the threading policy, to establish connection with the InfrastructureServices, and to set up itself for system management. The process then, initializes the CTM Adapter component and prepares to send data to CTM. The process goes to the normal server operation mode to

- monitor and process trade report events from the Event Infrastructure
- monitor and process Application Management requests

The CTM Adapter Server process needs normal transaction management to retrieve and store trade report data.

4.12 COMPASS Adapter Server

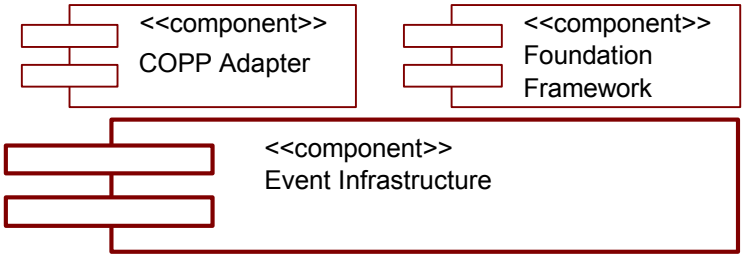


The COMPASS Adapter Server process starts when a system administrator manually brings up the server or it is started by a monitoring process. When started by a monitoring process, the process relies on the Foundation Framework to set up the threading policy, to establish connection with the Infrastructure Services, and to set up itself for system management. The process then establishes the connection between the COMPASS and the COMPASS Adapter. The process goes to the normal server operation mode to

- monitor and process incoming wire order requests from COMPASS
- monitor and process wire order status events from the Event Infrastructure

Order requests are processed as they come to the process. There is no transaction and recovery needed at this process.

4.13 COPP Adapter Server



The COPP Adapter Server process starts when a system administrator manually brings up the server or it is started by a monitoring process. When started by a monitoring process, the process relies on the Foundation Framework to set up the threading policy, to establish connection with the InfrastructureServices, and to set up itself for system management. The process then, initializes the COPP Adapter component and prepares to send data to COPP. The process goes to the normal server operation mode to

- monitor and process current market, last sale, and market summary events from the Event Infrastructure

- monitor and process Application Management requests

The COPP Adapter Server process keeps no stateful information about the data as they pass through the process.

5. Deployment View

The following is not a definitive layout, but rather a first high level cut of a possible deployment of SBT. This view is necessary in order to identify security, reliability, availability, performance, and capacity needs. It will be updated throughout the lifecycle of the application, as each phase dictates.

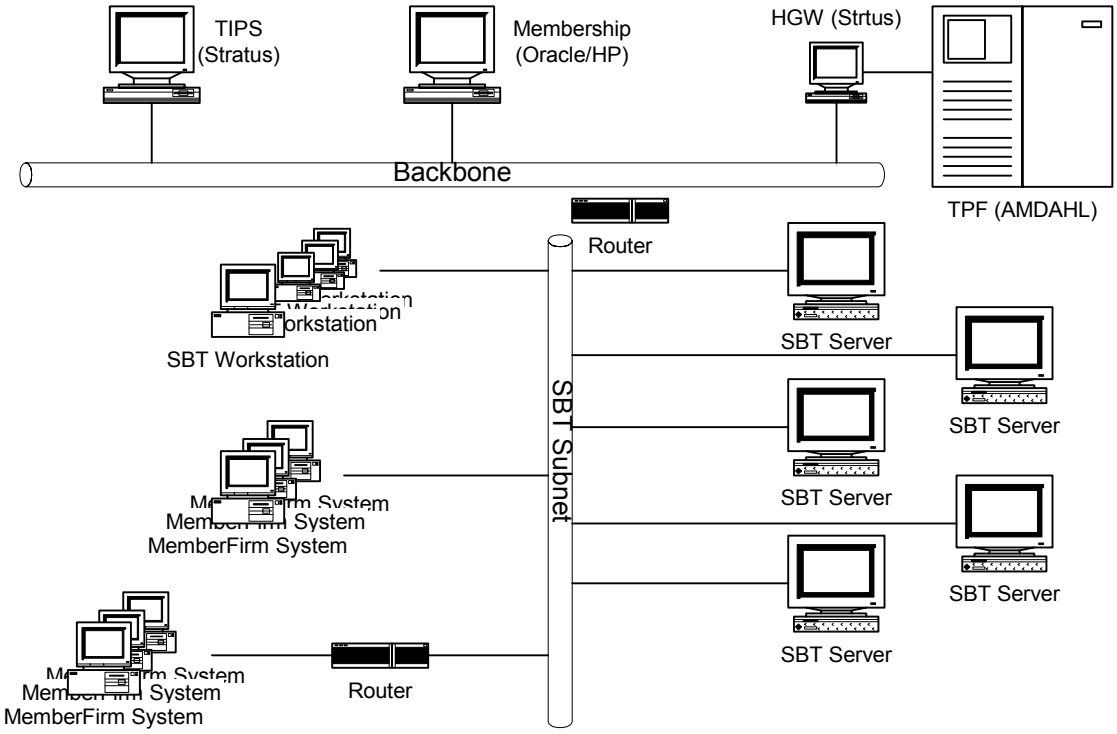


Figure 18 Deployment Overview

5.1 Process Deployment

[Describe the proposed physical location of hardware, and the processes that run on each machine.]

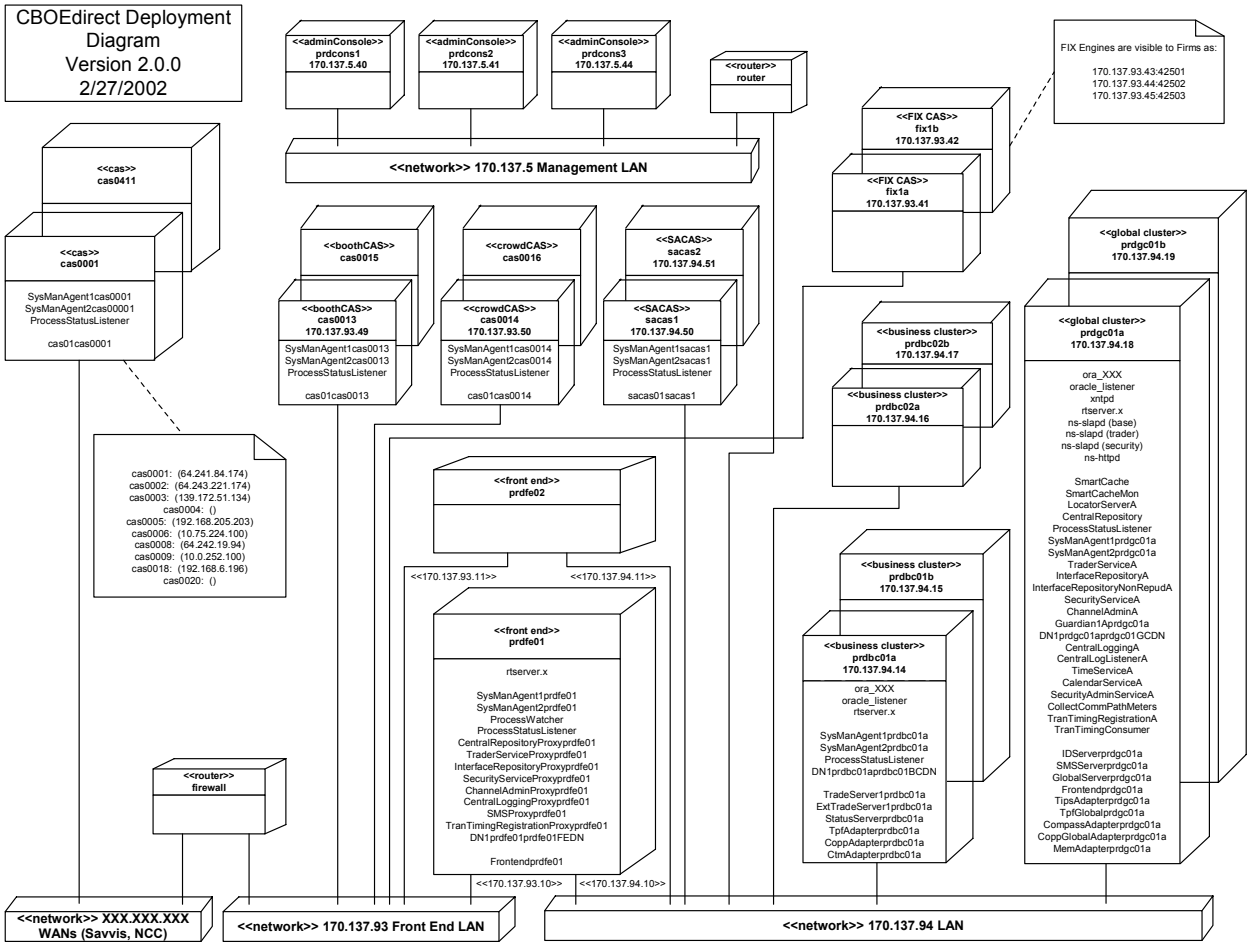


Figure 19 Process Deployment Overview

5.2 Processing Needs (Application Layer)

5.2.1 Hardware

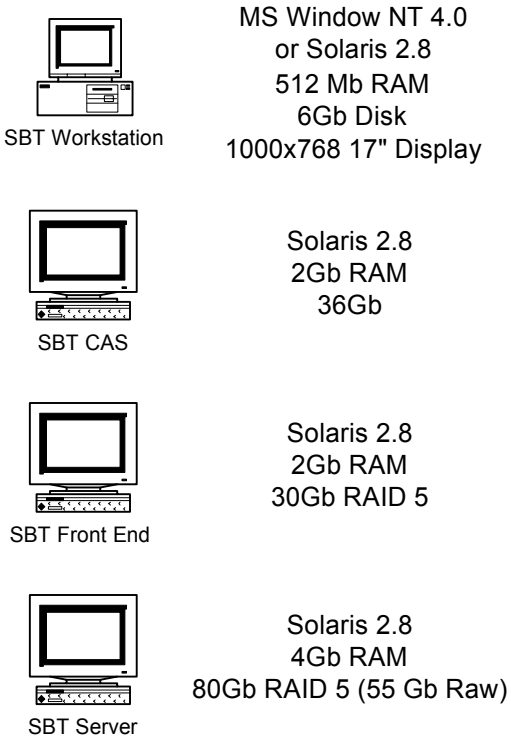


Figure 20 Hardware Overview

5.2.2 Performance

SBT services are deployed in servers using three different styles of services: front end server, global service server and service partition server. With a *front end server*, requests are received and routed to the appropriate server for processing by the front end. The processing at the front end is minimal. Multiple instances of the front end are running to provide the performance needed, and to provide a level of fault-tolerance. A client can connect to any of the front ends and receive the same service. A client will typically be assigned a primary and secondary *front end server*.

With *global service servers*, a number of servers with identical services are deployed. Each of the global service servers is self-contained and has a complete set of replicated data to function independently. The characteristics of deployment for this style of service are:

- Global usage of service, where the client does not care which instance of the server processes the request.
- Frequency of service is less frequent than the other style. For example, the service is only needed on the initialization or when validating data (using some type of caching after getting the initial data).
- There is more querying than updating to the data source that the global service servers contain. This is to minimize the replication effort between them.

The performance on this style of server is scaled increased by deploying additional instances of the server.

With the *service partition server*, the processing of services is partitioned among instances of servers. Each instance of a server is responsible for a mutually exclusive group of trading sessions/products (classes). All the instances of services use the same processing logic, but the processed data are all different between server instances. In SBT processing of products can be logically divided by products (e.g. IBM Jan 110 CALL, IBM Mar 110 PUT). The characteristics of deployment for this style of service are:

- Frequency of service at an aggregation can be significantly far more than any instance a server provides.
- Does not distinguish about the usage pattern of data (read vs. write) beside the fact that each of the server instances owns its data.

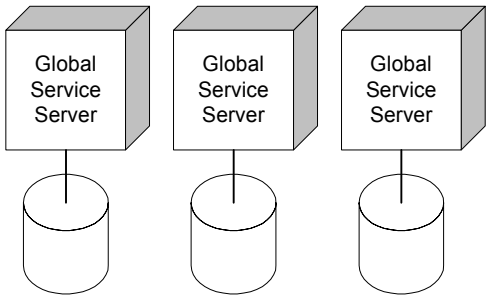
Balancing the products to evenly distribute the processing load, and deploying more server instances achieve the performance for this style of server.

5.2.3 Availability/Reliability

The many benefits of distributed computing systems come with a price: an increase in potential faults. Each of the distinct components in a distributed system is a potential failure point. Dealing with availability and reliability are delegated as much as possible down to the component level (Event Messaging Infrastructure has guaranteed message delivery capability; each service component has self logic error checking, etc.). However, potential failure points exist where components interact with other components.

When deploying the front end server, multiple instances of the server should be available. A client can connect to an additional secondary front end for high availability. All the requests are sent using the primary until the primary goes down. The client can switch to the secondary as the new primary and, optionally, another secondary can be selected and reconnected.

When deploying the global service server, a minimum of two instances of the server is required on two separate machines. Retrying the call to the service on the next available server eliminates a single point of failure for services running on this style of server. The interruption of service is handled transparently by the smart proxy pattern of communication. Note: When multiple instances are deployed they are sharing the load of the whole system because each of them is self-contained.

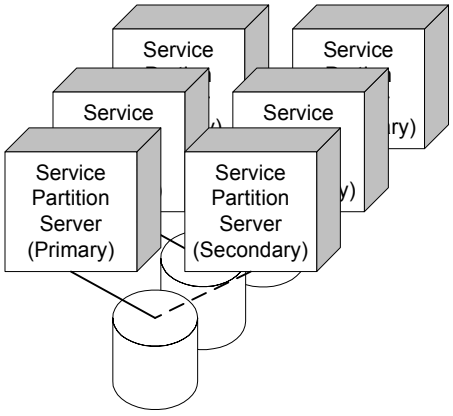


Each Global Service Server instance is self contained. Each of them is fully functional by itself. Deploy more than one instance to avoid single point of failure.

Figure 21 Global Service Server

Another approach for the deployment of a global service server is having a single set as primary and a backup as secondary. The secondary takes over the primary job when it detects the failure. The interruption of service for this approach is the time for the secondary to take over the primary. During this time interval, no service is available for servicing requests.

The service partition servers are deployed with primary and secondary backup. The secondary can be a hot/warm backup when the setup is in a high availability mode, where the secondary can assume the working environment as soon as the primary goes down. The secondary can also be set up in a pool of secondary machines where the disk array on the primary data can be manually switched to a secondary machine when the primary goes down. The interruption of service is localized to the set of partitions for the service. In SBT, only the products that are supported by the primary server are not available until the secondary is up and running.



Each Service Partition Server is paired with a secondary. When the primary fails, the secondary takes over the opertion. A partition of service will be unavailable during the transition state.

Figure 22 Service Partition Server

5.2.4 Functionality

Below is a mapping of SBT functionality to the system resource. Using the UML notation, each of the boxes denotes a host machine that provides the service. Each of the processes (denoted as square box inside a host machine) is deployed as one or more processes. Database notations are attached to the host machine to signify the fact that a Database Management System is deployed as the data repository.

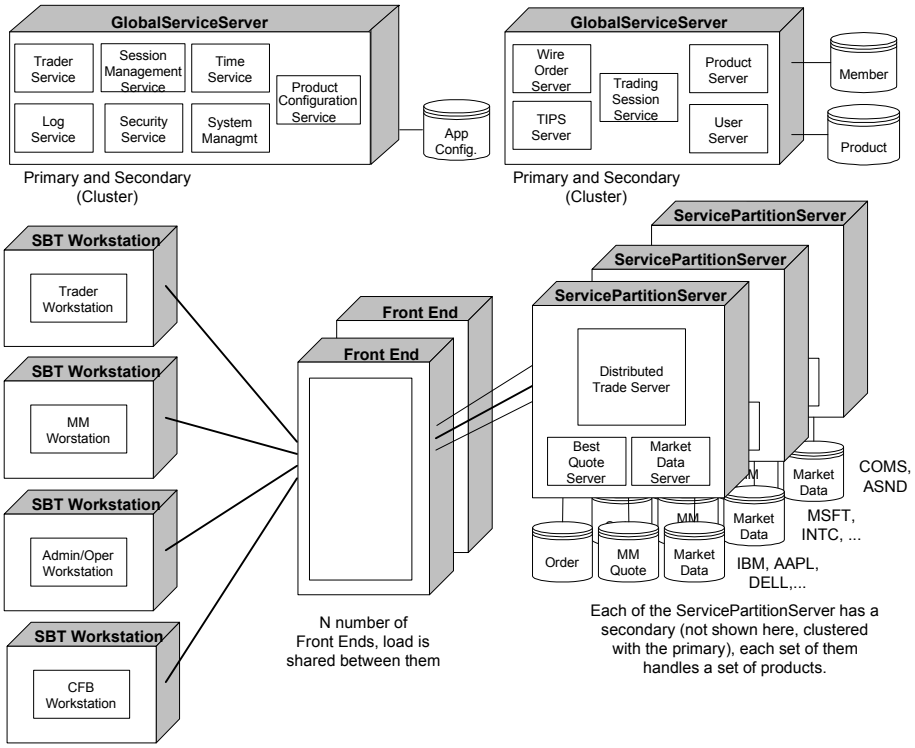


Figure 23 SBT Functional Deployment

5.2.5 Migration

All the interfaces between components are versioned to ensure the migration between the iteration and phases of development. SBT services are deployable with multiple versions. As far as deployment is concerned, each service and version comprises a unique deployable service.

5.3 Data Needs (Information Layer)

5.3.1 Physical Location

[Describe the proposed physical location of data. Identify the application's persistent and transient data items. Address issues with both centralized and distributed persistent data. Address issues and needs for any shared data. Identify the bottlenecks and performance issues associated with any centralized data.

Identify the application's distribution and synchronization issues associated with any distributed data.]

5.3.2 Capacity

[Discuss the application's expected memory and disk capacity needed to support the transient and persistent data. The analysis should include at least a five-year forecast for the data capacity.]

5.3.3 Integrity

[Discuss the integrity requirements of each data item. Address the application's reliability and auditing issues, as well as its backup and recovery needs.]

5.3.4 Access

[Discuss the access by systems and individuals to each data item. Address security issues and expected policies for handling security breaches and violations.]

5.3.5 Management

[Discuss the management of data for the application. Identify the target database management system or file system in which each data item will reside. Identify in as great a detail as possible the vendor, version, and operating system for any DBMS.]

Oracle 8.1.6 (Solaris 2.8) will be used for data management for the Distributed Trade Servers, and for Systems Management and Configuration. Oracle Parallel Server (OPS) will provide failover capabilities by running instances of the database server on each node in the cluster.

Netscape's LDAP 6 (Solaris 2.8) will be used for data management for the Directory and Security Services.

The file system and Oracle 8.1.6 (Solaris 2.8) will be used for Log Service messages, depending on quality of service.

5.3.6 Migration

[Discuss what data should migrate for legacy systems or from the previous version of the application. Identify a strategy for rolling the data forward with the capability of rolling the data backwards, when aborting an installation.]

No data will be required to be migrated to SBT. The system is designed to obtain product and membership data directly via the appropriate adapters.

5.4 External Interfaces (Communication Layer)

5.4.1 Physical/Logical Connectivity

[Discuss the proposed physical connections and protocols the application will use to implement each external interface. Determine if each interface is either

asynchronous or synchronous. Identify the data/message format from the integration architecture that each interface will use. Identify in as great a detail as possible the vendor, version, and operating system of any off-the-shelf product the application uses to implement each external interface, or identify the infrastructure services the application uses to implement the interface.]

5.4.2 Performance

[Perform some high level analysis of the size of data to be sent as well as the frequency and speed required for sending the data across each interface.]

5.4.3 Capacity

[Analyze the projected network, memory, and disk capacity needs to support each interface for at least the next five-years.]

5.4.4 Security

[Address all security constraints on each interface. Identify the network security requirements in the integration architecture that apply to this application.]

5.4.5 Migration

[Identify the strategy the communication layer will use to handle multiple versions of hardware, software, and data during any migration period.]

5.5 User Interfaces (Presentation Layer)

5.5.1 Physical

[Identify the target platform(s) for the user interface. Identify hardware needs, such as, mouse, color monitor, speech recognition, etc.]

5.5.2 Performance

[Document the forecasted number of concurrent user interfaces to the application over a five-year period. Identify any performance expectations on the application in support of the user interface.]

5.5.3 Security

[Address security needs such as user identification. From the choices in the technology architecture, select hardware and/or software that the application will use to implement user identification.]

5.5.4 Standards and Guidelines

[Identify any standards, like Windows programming standards or Motif programming standards that apply to the user interface paradigm selected from the technology architecture. Identify all presentation policies from the integration architecture that the application will implement.]

5.5.5 Migration

[Identify the strategy the presentation layer will use to handle multiple versions of data and messaging during any migration period.]

5.6 Hardware Installation

[Describe the expected order in which to install the hardware. Describe dependencies in both installation and in system startup/shutdown order. Define the setup of each system for user access, disk configuration, backup facilities, and monitor/control capabilities.]

5.7 Third-Party Products

This section identifies the third-party products required by SBT by name, version, and patches.

- JavaSoft JDK 1.2.2_12 (Java 2)
- Talarian SmartSockets 5.5 r3
- Oracle 8.1.6
- Oracle JDBC 8.0406 Thin Driver
- JavaIDL ORB (CMi to client)
- Netscape LDAP client development toolkit
- Netscape Certificate Server
- IAIK cryptographic libraries

5.8 Software Installation

[Describe the expected method of deploying the application into the operational environment. Define the media and format for deploying the system. For example, the application could be a Java JAR on a server that each client downloads and installs. Describe the state of the system before and after the software installation. Define any dependencies on hardware, third party software, or other applications.]

5.9 Maintenance

[Describe the approach for routine maintenance, such as software patches. Describe emergency maintenance procedures, such as backing out an installed version of the application to the previous version, due to some catastrophic errors in the software.]

6. Appendix

[Use for any elaboration of concepts introduced in the previous sections of the document.]

7. Glossary

7.1 Acronyms

Acronym	Definition
AON	See 7.2 terms - All or None Order.
BART	Booth Automated Routing Terminal is a CBOE client/server system used by booth staff to receive and re-route Orders.
BQS	The Best Quote Service is responsible for calculating the market best (best quotes) for each product. It also calculates and disseminates the NBBO.
BBBO	Best Book Bid Offer
BS	The Broker Service is responsible for executing various types of orders (i.e. Limit, Market, All or None, Fill or Kill, Immediate or Cancel, Stop, Stop Limit, Spread). It notifies the Trade Service of all the orders matched in the trade and it also notifies the OHS and MMQS of the fills.
CAS	Client Application Server
CBOE	Chicago Board Options Exchange
CFB	See 7.2 terms - Clearing Firm Broker.
CMi	CBOE Market Interface
COMPASS	A CBOE system that connects member firm branches with the trading engine. It sends Orders to the trading engine and receives fill reports from the trading engine.
COPP	CBOE OPRA Participant Processor is a CBOE system that forwards price reports, quote reports, and administrative messages to OPRA for dissemination to vendors and the general public.
COSS	Commercial off-the-Shelf Software
CQS	Consolidated Quote System is a SIAC system that receives stock quotes from stock exchanges and disseminates them to data feed vendors and the general public.
CTM	See 7.2 terms - Trade Match System.
CTS	Consolidated Tape System is a SIAC system that receives stock trade reports from stock exchanges and disseminates them to data feed vendors and the general public.
DART	Direct Access Routing Terminal is a CBOE system (currently known as BERS) used by the following:

Acronym	Definition
	i. Booth staff to enter Orders into the trading engine ii. Fill reporters to enter fill reports into the trading engine iii. Floor workers to enter inquiries into the trading engine iv. The display subsystem to display information to floor workers
DTE	Distributed Trading Engine
EB Backup	Electronic Book Backup is a CBOE system that provides a view of the book if the trading engine fails.
EPW	See 7.2 terms - Exchange Prescribed Width.
FIX	Financial Information Exchange Protocol
FOK	See 7.2 terms - Fill or Kill Order.
GTC	Good Till Cancelled.
ICS	Integrated Class Series is a CBOE system used for entry and storage of information concerning valid products. Will become the IPS.
IOC	See 7.2 terms - Immediate or Cancel Order.
IPS	Integrated Product System which will replace ICS.
MDS	The Market Data Service maintains a snapshot of market data, in addiiton to publishing market summary data. It also provides an interface to clients to query historical market data.
MIN	See 7.2 terms - Minimum Order.
MMHH	Market-Maker Hand-Held is the PSP application responsible for accepting trade data that a Market-Maker enters into his/her Market-Maker Terminal. MMHH routes the trade to CTM and forwards price reports to the trading engine. In addition, it receives and displays trade acknowledgments from RAES.
MMQS	The Market-Maker Quote Service is responsible for receiving and submitting RFQs, sending fill reports, and cancelling or updating quotes. It also provides the history of the quotes submitted by a Market-Maker.
MMT	Market-Maker Terminal is a hand-held terminal used by Market-Makers to enter trade reports. It also provides tools to help the Market-Maker trade more efficiently, such as position tracking, risk management, equivalent stock position, delta analysis, pricing models, and graphic position monitor. The MMT is also the hardware component of

Acronym	Definition
	the MMHH application that interacts with a Market-Maker.
NASDAQ	National Association of Securities Dealers Automated Quotation System
NBBO	National Best Bid Offer.
NMMPT	See 7.2 terms - Non-MarketMaker Professional Trader.
OBO	Order Book Official is a CBOE employee who maintains the book and trades booked Orders.
OBS	The Order Book Service is responsible for publishing the new BBBO and last sale upon changes to the top of the book. It cancels and cancel/replaces resting orders and it acknowledges that an order was accepted. The OBS also assists in calculating the opening price during a products pre-opening period.
OCC	Options Clearing Corporation is an organization owned equally by the US options exchanges. It is responsible for clearing options transactions and makes possible the secondary market for options by severing the contractual link between the buyer and the seller.
OHS	The Order Handling Service application maintains the current state of all orders. It is responsible for creating, cancelling and updating orders. Notification of changes in order status is supported by the Order Handling Service as well.
OPG	See 7.2 terms - Opening Only Order.
OPRA	Options Price Reporting Authority.
ORB	Object Request Broker
ORS	The Order Routing System is a TPF application responsible for routing electronic Orders to RAES, EBOOK, PAR, or booth terminals.
OSS	The Order Status Service provides subscription and notification services related to orders (i.e. fill reports, cancel reports, order accepted by book, etc.).
P&S (Purchase-and-sale) statement	A statement sent by a brokerage firm to a customer when a futures or options position is offset or extinguished by delivery. A P&S statement typically shows the number of contracts involved, the prices at which and dates on which the contracts were bought and sold, the gross profit or loss, the commission charges, the net profit or loss on the transactions and the account balance.
PAR	Public Automated Routing is a CBOE system used by Floor Brokers to process public Orders.
PCS	The Product Configuration Service is responsible for providing the location of where a product is processed/traded.

Acronym	Definition
	product is processed/traded.
PDS	Post Data Server is the part of the display subsystem that is the central repository of class and series information for a specific post. The trading engine sends display information (e.g., CBOE best quotes and NBBO quotes) to the PDS. The Display Subsystem is a CBOE system used to display information on AutoOrder, Datawall, MMT, PAR, RCN, SideBySide, DART, and WPT.
PS	The Product Service maintains all product-related information. It downloads the product information from TIPS and TPF.
PSS	The Product State Service is responsible for coordinating product state changes for all products, e.g. pre-opening, opening, trading, halting, closing and post-closing.
QSS	The Quote Status Service provides subscription and notification services related to quotes (i.e. fill reports, deletion reports, etc.).
RAES	Retail Automated Execution System is used for executing small market Orders automatically.
RAT	RAES Trade Tape is a tape created by the trading engine at the end of the day that contains information about all the trades for the day.
RFQ	Request for Quote.
ROS	Rapid Open System is a CBOE system used by OMM's to determine the opening price for options.
RTH	Regular Trading Hours.
SBT	Screen-Based Trading.
STP	See 7.2 terms - Stop Order.
STP LIMIT	See 7.2 terms - Stop Limit Order.
TAT	Trade Acknowledgment Ticket is a notification to a Market-Maker that he / she is the counter-party for a trade.
TIPS	Ticker Processing System is the CBOE system that forwards market information, e.g., stock prices, stock quotes, NBBO quotes, etc., from external exchanges to the trading engine.
TS	The Trade Service is responsible for receiving trade notification from the Broker Service, formatting and storing trade reports, and forwarding trade reports to CTM via TPF.

Acronym	Definition
TSS	The Trading Session Service maintains all business day and trading session-related information and manages the different states of a trading session, e.g. open, closed, halted.
US	The User Service maintains all user-related information, both specific to SBT and contained in CBOE's Membership System. If provides a unified interface to SBT components accessing user information, hiding the actual location of the maintained data, thus simplifying client logic and facilitating replacement of the Membership System in the future.

7.2 Terms

All or None Order	This type of Order can be executed at anytime and has to be filled completely, not partially, or not at all.
American Option	An option that can be exercised on any business day before it expires.
Ask	Also called “offer. ” Indicates a willingness to sell an options contract at a given price. (See Bid).
At-The-Money	An option whose strike price equals the current price of the underlying commodity, security, currency, index or futures contract.
Bear	One who expects a decline in prices. The opposite of a "bull.”
Bear Market	A market in which prices are declining.
Beta (or Beta coefficient)	A statistical measure of the relationship between the price volatility of an individual stock or stock portfolio and the price volatility of the overall market. Beta is often used in computing hedge ratios for stock index futures positions.
Better	In the context of a purchase, price A is Better than price B if A < B. In the context of a sale, price A is Better than price B if A > B.
Bid	An Order to buy at a given price.
Book	Contains pre-open Orders, limit Orders, and intra-day limit Orders at least one tick away from the same-side market quote.
Book Staff Clerk	Someone who maintains book Orders using the trading engine.
Broker	A person who is paid a fee or commission for executing Orders. In futures and options trading, the term may refer to: (1) a floor broker, i.e., an exchange member who

	executes Orders on the trading floor of an exchange; (2) an account executive or associated person who deals with customers for a futures or options commission merchant or introducing broker; and (3) a futures or options commission merchant.
Bull	One who expects a rise in prices. The opposite of a "Bear."
Bull Market	A market in which prices are rising.
Call Option	An option that gives the buyer (holder) the right, but not the obligation, to purchase a specific asset or obtain a long futures position at a fixed price within a specified period of time.
Class	The Underlying Asset of an Option, e.g., IBM stock, S&P 100 index.
Clearing	The procedure through which the clearinghouse becomes the buyer to each seller of a futures or options contract and the seller to each buyer and assumes responsibility for the financial integrity of each open contract.
Clearing Firm Broker	A broker employed by a Member Clearing Firm.
Clearing Member	A member of a clearinghouse through whom all trades must be settled.
Clearing Member Firm	A firm, member of an exchange, which is also a clearing member of a clearinghouse. In the case of CBOE, a Clearing Member Firm is a member of OCC.
Clearinghouse	An adjunct to a futures or options exchange through which transactions executed on the floor of the exchange are matched, settled and guaranteed. Charged with assuring the adequate financial protection of trading through collection and payment of margin and the proper conduct of the exchange's delivery procedures.
Contingency Order	One of the following types of Orders: i. OPG – Opening Only Order ii. AON – All or None Order iii. FOK – Fill or Kill Order iv. IOC – Immediate or Cancel Order v. MIN – Minimum Order vi. STP – Stop Oder vii. STP LIMIT – Stop Limit Order viii. MOC – Market On Close

Covered Call Writing	To grant, or write, a call option while holding or having a long position in the underlying security, commodity, currency, index or futures contract.
Cross-Hedge	Hedging a cash market risk in a futures or options contract for a different but price-related commodity, security, currency, or index.
Day-Trading	Establishing a futures or options position and offsetting it the same day.
Default	Failure to perform on a futures or short options contract as required by exchange rules.
Delivery	The tender and receipt of an actual commodity or financial instrument, or cash in settlement of a futures or options contract.
Delta	The amount of change of an option's price or theoretical value for a unit change in the price of the underlying security, commodity, currency, index or futures contract
Exchange Prescribed Width	The maximum difference allowed between the market's Bid andAsk Prices for the quote to be considered valid.
Exercise or Strike Price	The price at which the holder (buyer) may purchase or sell the underlying asset upon the exercise of an option.
Expiration Date	The last day that an options contract may be exercised.
Fill or Kill Order	This type of Order has to be filled completely within a predefined period of time or otherwise killed (cancelled).
Hedging	Taking a position in a futures and/or options market to minimize the risk of an adverse price change in a position held in the cash market.
Holder	One who purchases an option.
Immediate or Cancel Order	This type of Order can be partially or completely filled. However, if it is partially filled, the remaining quantity is cancelled.
Initial Margin	Customers' funds put up as security to guarantee contract fulfillment at the time a futures or options position is established.
In-The-Money	A call option with a strike price lower, or a put option with a strike price higher, than the current market price of the underlying asset or futures contract.
Intrinsic Value	For a call option, the excess of the current market price of the asset or futures contract underlying the option over the strike price of the option; for a put option, the excess of the strike price over the current market price of the asset or futures contract underlying the option.

Limit Move	A price that has advanced or declined the permissible amount during one trading session, as fixed by the rules of an exchange.
Limit Order	An Order that can only be matched to another, opposite-side Order at the Limit Order Price or Better price, but not at a Worse price.
Limit Order	This type of Order can only be filled at the specified Limit Price or better. See “” for further details about execution of this type of Order.
Liquidation	(1) Offsetting or closing out a futures or options position; (2) a market in which open interest is declining.
Long	(1) One who has bought a futures or options contract to establish a market position; (2) a market position that obligates the holder to take delivery; (3) one who owns an inventory of commodities or securities.
Margin Call	(1) A request from a brokerage firm to a customer to bring margin deposits back to initial levels, normally because of losses resulting from an adverse price move; (2) a request by a clearinghouse to a clearing member to make payments to or increase deposits at the clearinghouse.
Market Ask Price	The lowest price of all Outstanding sell Orders.
Market Bid Price	The highest price of all Outstanding buy Orders.
Market Order	This type of Order is filled at the standing Bid or Offer quote. If the available quantity is insufficient, certain rules apply to execute the remaining volume.
Market-Maker	A Market-Maker is a broker member of an exchange that trades exclusively for himself. Market-Makers provide liquidity to the market by acting as contrabrokers in trades of pre-specified products.
Market-Maker Quote	Two Orders – a buy Order and a sell Order – for the same product and quantity and with prices within the EPW, entered simultaneously by a Market-Maker.
Mark-To-Market	The daily adjustment of margin accounts to reflect profits and losses.
Matching an Order	The action of trading an Outstanding Limit Order against another, opposite-side, Outstanding Limit Order for the same product at a certain price. If the quantities specified on the Orders are not equal, the Order with the greater quantity is filled partially and its quantity reduced by the quantity of the other Order. Thus, each Order has an Original Quantity, and optionally, a Fill Quantity and a Remaining Quantity.
Membership	The CBOE system that maintains member, Trader, and firm information.
Minimum	In this type of Order, the filled quantity should at least equal the minimum volume

Order	specified.
Minimum Price Fluctuation	Smallest price change possible in a futures or options contract. Also called the tick value.
Nearby	The nearest active trading month of a futures or options contract. Also referred to as "lead month."
News Wire	
Non-MarketMaker Professional Trader	A broker trading for himself off the floor. Also known as broker/dealer.
Offer	An indication of willingness to sell at a given price; opposite of Bid. Also known as "Ask."
Offset	(1) Liquidating a purchase of futures or options through the sale of an equal number of contracts of the same delivery month, or liquidating a sale of futures or options through the purchase of an equal number of contracts of the same delivery month; (2) matching total long with total short contracts for the purpose of determining a net long or net short position.
Open Interest	All futures or options contracts that have been entered into and not yet liquidated by an offsetting transaction or by delivery.
Open Order	An Order to a broker that is good until it is canceled or executed.
Open Outcry	A system of trading in which Traders stand in pits announcing the prices at which they want to buy or sell and consummating trades.
Opening Only Order	Opening Only Order. This type of Order is executable only in the opening trade; it expires after the opening trade or after the opening quote is disseminated.
Opening Price (Or Range)	The range of prices at which the first bids and offers were made or first transactions were completed.
Opening, The	The period at the beginning of the trading session during which all transactions are considered made or first transactions were completed.
Option	A contract that gives the buyer the right but not the obligation to buy or sell a specified quantity of an asset at a specific price within a specified period of time, regardless of the current market price of the Underlying Asset.
Original Execution Price	The price at which an Order is initially filled.

Out-of-the-money	A call option with a strike price higher or a put option with a strike price lower than the current market price of the underlying commodity, security, currency, index or futures contract.
Outstanding Order	A simple Limit Order pending matching.
Out-Trades	A situation that results when there is some confusion or error on a trade. A difference in pricing, with both Traders thinking they were buying, for example, is a reason why an out-trade may occur.
Position Trader	A futures or options Trader who buys or sells contracts and holds them for an extended period of time--as distinguished from a day Trader, who normally initiates and offsets positions within a single trading session and ends the day "flat."
Post	A specially constructed arena on the trading floor where futures and options trading is conducted. Also called Pit.
Premium	The amount agreed upon between the purchaser and seller for the purchase or sale of an option – purchasers pay the premium and sellers (writers) receive the premium.
Product	A Series traded at CBOE.
Put	An option to sell a specified amount of a commodity, security, currency, index or futures contract at an agreed-upon price within a specified period of time.
Regular Trading Hours	The period of time during which open outcry trading takes place.
Round Turn	A completed transaction involving both a purchase and a sale.
SBT Server	The SBT Server is the server software component of the SBT application.
SBT Workstation	A SBT Workstation is a computer running the client sotfware component of the SBT application. The SBT client application is used by an SBT user to interact with the SBT application.
Scalp	To trade for small gains. Scalping normally involves establishing and liquidating a position quickly, usually within the same day, hour or even just a few minutes.
Scalper	A speculator on the trading floor of an exchange who buys and sells frequently, holding positions for only a short period of time during a trading session. In liquid markets, scalpers stand ready to buy at the minimum price change (tick) below the last transaction price and to sell at a tick above.
Series	A specific option defined by: i. Class or the underlying asset, e.g., IBM

	ii. Type of option: Call or Put iii. Strike Price iv. Expiration Date
Settlement Price	The price at which the clearinghouse each day settles all accounts between clearing members for each open position in each contract month of each futures or options contract. Settlement prices are used to determine both margin calls and invoice prices for deliveries.
Short	(1) The selling side of an open futures contract; (2) a Trader whose net position in the futures market shows an excess of open sales over open purchases; (3) selling (granting) an options contract.
Short-Covering	Buying to offset an existing short position.
Spot	Market for immediate delivery and payment of the product.
Spread	The purchase of one futures or options contract against the sale of another futures or options contract of the same or related commodity, security, currency or index.
Squeeze	Situation in which those who are short cannot repurchase their contracts, except at an artificially inflated price,
Stop Limit Order	This type of Order becomes a Limit Order when a quote or trade takes place at or through the stop price
Stop Order	This type of Order becomes a Market Order when a Order or Trade takes place at or through the Stop Price.
Stop Order (or Stop)	An Order to buy or sell at the market when and if a specified price is reached.
System Operator	A CBOE employee who maintains the operation (e.g., startup, shutdown, control functions) of CBOE systems.
Tick	Refers to a change in price, either up or down. Minimum price fluctuation of a futures or options contract.
Time Value	The excess, if any, of an option's premium over its intrinsic value.
Trade Match System	The CBOE system that matches trades based on trade report information sent from various CBOE systems and from member firm submissions.
Trend	The general direction of the market.
Underlying Asset	The asset that the writer of an option is obligated to buy from or sell to the buyer of the option. Such asset may be a commodity, security, currency, index, or futures contract on one of them.
Valid Market Quote	The market for a Product is said to have a valid quote if: There exist a Bid Price and an Ask Price, and

	The difference between the two is lower than or equal to the Exchange Prescribed Width.
Variation margin	Settlement via the clearinghouse of daily or intraday gains and losses between clearing member firms.
Vendor Machine	The machines maintained by vendors and used to send vendor quotes to the trading engine.
Volatility	A measure of variability, usually of prices, and a major factor influencing the price of an option. The standard deviation of a price series is commonly used to measure price volatility.
Volume	The number of contracts (either the long or the short side of the market) traded during a specified period of time.
Worse	In the context of a purchase, price A is Worse than price B if A > B. In the context of a sale, price A is Worse than price B if A < B.
Write	The act of selling or granting an options contract.
Writer	The seller or grantor of an option.

8. Index

[Optionally, provide an index for referencing important topics.]