Business and Application Architecture Engineering

**Business Architecture Modeling**

Business architecture modeling helps extract the overall business model of an organization, which includes the set of underlying processes necessary to operate the various functions of the organization. The business model includes the organization, location and process model as described below.

**Organization model**

The organization model provides role definitions and a process/role matrix. Role Definitions are groups of functional capabilities, which must be present in a single person in order for that person to carry out one or more specific business processes. A single person may perform more than one role. The process/role matrix indicates the business roles responsible for executing specific business processes. It is usually built at the Elementary Business Process (EBP) level but may also be completed at higher levels if useful.

As an example, the roles to consider in a typical financial instruments trading business model would be as follows.

1. **Clients:** Existing institutional customers presently managed by a salesperson. Clients have full trading privileges only in the instruments for which they are authorized.

2. **Salespeople and Sales/Traders:** Salespeople report to the sales desk manager. Salespeople are responsible for managing client’s needs with regard to a particular product / products or asset class. Salespeople and sales trading desks have trading privileges on behalf of a subset of clients, e.g. the clients covered by a particular sales person or that sales person’s group. They interact with the various trading desks on behalf of clients in order to:
   - Obtain quote requests
   - Enter orders
   - Discuss market commentaries
   - Help resolve issues (trade detail inquiries)
   - Change status of client orders
Salespeople also help customers understand research and suggest trade ideas.

3. **Execution Desk (Traders):** Traders report to a trading desk manager. Traders are responsible for making markets in a particular product / products, within an asset class. They also manage the risk associated with a particular trading book/s. Traders are a special type of client (internal clients). Like clients, they have full trading privileges only in the instruments for which they are authorized. Some trading responsibilities include:
   - Provide prices being published to clients
   - Provide pricing for quote requests
   - Approve / reject / counter-offer orders sent to the trading desk by clients
   - Determine axis to be offered to clients
   - Offer a market commentary

4. **Support – Back-Office Operations:** Often referred to as “back-office personnel, support people ensure the timely processing, settlement and delivery of transactions among other things. Support has access to all windows accessible to clients. They have no trading privileges but they can modify the status of an order on behalf of a client. Support people staff the following areas:
   - Trades processing
   - Settlements
   - Margin
   - Stock loan
   - Sales Support
   - Others

Assumption: There may need to be a distinction made between back-office and front-office personnel’s functionality. Support – Front-Office Operations: Typically work along-side Traders and Salespeople. They facilitate telephone calls, process trades and resolve trading and sales/customer issues. Front–Office personnel would consist of:
   - Assistant traders
   - Assistant salespeople

5. **Client Mid-Office:** These would be the client’s support staff.

6. **Administrators:** Administrators do not have trading privileges. Administrators alone have access to administrator windows. Administrators would provide the following support:
   - Account set-up
   - Account maintenance
   - Modify client access
   - Modify client trading privileges

7. **Client Administrators:** Administrators do not have trading privileges. Administrators alone have access to administrator windows. Client administrators can only to modify account holder privileges, within the limits of that client’s trading and access privilege.

8. **Marketing Desk:** Provide marketing support to salespeople.
Interact with traders on behalf of regional salesforce, performing similar functions as a salesperson would. However, they may have limited client contact.
- Assist in educating the salesforce with regard to financial products.
- Apprise the salesforce of new issues and other market related information received by the trading desks and other sources

**Assumption: There exist such a desk performing the above listed duties.**

9. **Trading Desk Managers**: Supervise traders on individual trading desks. Responsible for the overall profitability of their trading desk.

10. **Sales Desk Managers**: Supervise salespeople on individual salesdesks. Responsible for the overall profitability of their sales desk.

11. **Upper Management**: Key managers that oversee several business units, operations departments and other areas. These would include:
   - Division Managers
   - Group Managers
   - Risk managers

**Assumption: CIO, CFO and CEO level of management may want certain functionality.**

12. **Systems Management**: Key technical people responsible for the daily operation of the trading system, including:
   - Site Manager
   - Systems Maintenance Manager
   - Content Manager
   - Help Desk
   - Others

### Location Model

The Conceptual Location Model shows how business processes will be distributed geographically. In the location model, location type definitions identify locations in terms of location type and general geographical area. A process/location matrix is also provided to indicate location types at which each process is performed. This matrix is used at the level of elementary business processes (EBPs). In our sample trading business model the location type definitions could be:

- Boston branch sales office
- Midwest district service center
- London regional distribution center
- New York Headquarters
- Singapore Headquarters
- London Headquarters
Process Model
The business process model can be summarized in business process hierarchies. A business process hierarchy is a schematic representation of the results of process decomposition. Decomposition is taken down to the level of the elementary business process (EBP). An EBP is the smallest unit of business activity that:

- is worth examining;
- is performed by or in support of a single user;
- may include both manual and automated activities;
- has user involvement occur at one time;
- preserves important business distinctions between processes;
- leaves the Conceptual Entity Model in a consistent state.

Business process flow can be captured within a process map. Business process maps show the sequence of selected processes from the Process Hierarchies as they relate to the role that performs them.

A high-level business process hierarchy for our sample trading business model is given below:
EBPs can be further decomposed in sub-hierarchies as illustrated below in the case of the Enter Order process of our sample trading business model:
EBP definitions contain a brief description of what occurs in each process. The definitions, combined with the relevant process flows, provide the basis to move forward into detailed design and support continued development of functional specifications. They also identify any assumptions made during the creation of the architecture, any rules for manual decisions that need to be made, and cross reference information, where applicable, to match the functional requirements of the application being designed.

The numbering of the EBPs corresponds to the layout on the hierarchies, and is used to cross-reference all of the other work products in the Business, Organization, and Location domains.

In the case of our sample trading business model, EBP definitions are as follows:

1. Subscribe Prices
   1.1 Receive Prices. Receive the updated prices from all pricing sources. Show the relevant information required to identify an instrument (e.g. ticker symbol, coupon, maturity, price, yield for bonds). This needs to be as robust and frequent as the Reuters update. Manage instruments (add, change, delete) through backend systems.
   1.2 Create Market Windows
      1.2.1 Configure Market Windows. Set up screens and layout of windows and workspace.
      1.2.2 Enter Search Criteria. Locate instrument/groups/portfolios for windows. Will need a request to publish if user is the first one to request this instrument’s price feed. Check Price Subscription Entitlements.
      1.2.3 Select Instrument for Market Windows. Select and delete instrument/group/portfolios to be represented on the Market Watch window.
      1.2.4 Populate Market Windows. Populate market watch view to show prices to users for their selected instruments. Create market depth view of an instrument. Accessed after selecting an instrument (description) from the market watch window. Allows users to view price sources for a given instrument from multiple sources.

2.0 Process Order
2.1 Quote Request. Look up instrument via search functionality starting at asset class, checking view permissions. Select instrument to move into order/quote request process. Client uses to access price not generally available or not on their Market Watch (in which case, user could just search, add instrument to their Market Watch.
   2.1.1 Select Quote Request Function. Select a Request for Quote (RFQ) button, menu item, or Market Watch window.
   2.1.2 Enter Search Criteria. Fill in as much info as possible for the search window to narrow the possible options: instrument, group, and portfolio. Selected results may be added to Market Watch. Check entitlements for viewing.
2.1.3 Select Instrument to Populate Ticket. Check trading entitlements. Opportunity to go back to search criteria and get another instrument/group/portfolio to add to ticket.

2.1.4 Review Ticket (RFQ, Order). View pre-populated RFQ or Order Ticket. Default fields are filled in based on Ticket type and asset class.

2.1.5 Enter Additional Information on Ticket (RFQ, Order). Enter quantity, and provide capability to override default settlement and other fields. Enter order types - Fill or Kill, GTC, Day Order. Enter account number. User has option to enter an order status change such as suspend order. User has option here to cancel or return to select another instrument (e.g. clicked on wrong market price or instrument or wants to add more.)

2.1.6 Submit Ticket (RFQ, Order). Submit RFQ or Order to Trading Desk. If Execution Desk does not respond to RFQ in 30 seconds.

- Validate ticket and/or entitlement Details – Make sure all relevant information has been entered on the ticket. User can edit and cancel. If info is missing or incorrect, send error message.
- Send ticket. Send acknowledgement to audit trail or user. Order state = ‘waiting’
- Route ticket to specific back-end system – The order will be routed to the appropriate back-end system for the given instrument based on order routing criteria.
- Send acknowledgement – An acknowledgement will be sent once the order has been received by the back-end systems.

2.1.7. Fill RFQ. Trader provides price, and any other required information, such as quantity, settlement for Quote Request and sends back to User.

2.2 Enter Order

2.2.1 Enter Order From Menu. Presumes price is missing and user must populate this in the Order Ticket.

2.2.1.1. Select Enter Order Function. Access menu item.
See EBPs 2.1.2. through 2.1.6.

2.2.2 Enter Order from Market Windows

2.2.2.1 Select Instrument from Market Watch. Choose the instrument for which an order is to be entered. This should open the Market Depth window and display multiple prices, and price sources.

2.2.2.2 Select Price of Instrument. Click on the desired price. If the offer price is selected, a buy order ticket will be displayed. Should the bid price be selected, a sell order ticket will appear.
See EBPs 2.1.4. through 2.1.6

2.2.3 Respond to filled RFQ
2.2.3.1 Select RFQ. Select from Order queue or from asynchronous pop-up window? Client can retrieve RFQ’s sent to the Execution Desk in order to cancel or modify the order. User has 5 seconds to respond once RFQ is received back with price. If time limit ends up being more than 5 seconds, may want to have Order queue functionality.

See EBP 2.1.4

2.2.3.2 Transform RFQ into Order. Continue order with information received back from Trading Desk within the specified time limit. If RFQ times out, user can still resubmit as order, but cannot expect it to be filled at that price.

See EBP 2.1.6

2.3 Execute Trade

2.3.1 Review Client Order Ticket. Trader reviews information on order ticket received from Client. Trader will decide if and how he will fill this order ticket. Order status changes to ‘confirm’ to note that Execution Desk has received it, and order queue reflects this.

2.3.2 Fill Order. Trading Desk person accepts or denies order. This part is manual. To provide process speed, parts of this should be automated based on some criteria (e.g. price limits, quantity limits). If order is approved, update order status (approved) and send return message that order is approved. If order is denied, updates order status (denied) and return message with denial, alternate price and/or quantities (can’t fill). Provide Alternate Price and/or Quantities.

2.3.2.1. Completely Fill Order. The user is notified by the status update of their order on their order queue that their order has been filled.

2.3.2.2 Partially Fill Order. System will send updates to order until completely filled or balance rejected/cancelled. User may change or cancel the unfilled portion of the order.

2.3.3 Confirm Fills. After receiving acknowledgement that order was filled, user will confirm fill.

2.4 Change/Cancel RFQ, Order. Change, Cancel, Change status (suspend). Depending on instrument type, user may need to cancel and add and order instead of change. Based on SEC regulations. Change and cancel orders may only be done if current Order State is ‘pending’.

2.4.1 Select Order from Order Blotter. Status must be pending or partially filled

See EBP 2.2.3.1 and 2.1.4

2.4.2. Cancel

2.4.2.1 Cancel Order. Cancel directly in order blotter or optionally review ticket then cancel from ticket itself.

2.4.2.2 Cancel RFQ

2.4.3 Modify Information on Ticket (RFQ, Order). Change information on previously submitted ticket
See EBP 2.1.6

2.5 Process Unfilled Orders

2.5.1 Reject Order. A message will be sent to the user that their order cannot be filled. User may change order.

2.5.2 Terminate Order, End of Day. At the end of the business day, users will be advised that all day orders that have not been filled are considered dead. Order will have an ‘expired’ status. If the user wishes to re-enter their order, they may do so, on the next business day.

3.0 Perform Management Functions

3.1 Set up Entitlements & Security

3.1.1 Provide Client Security Access

3.1.2 Determine Entitlements. Based on view and business functions allowed to do, role, asset class, account.

3.1.3 Administer Entitlements. Set up user on system, which drives their views and allows them to perform functions.

3.2 Provide Management Reporting. Periodic reporting, reconcile trades at end of day.

3.3 Perform Queries. Ad hoc or static on line queries. For trading/sales managers to get an overview of the business being done.

3.4 Manage Content. User news and other ‘push’ information, personalization.

3.5 Manage Site

3.6 Configure User Preferences. Provide user set up functionality to configure content and layout of views such as market watch and depth, Order queue and trade history. Access to data is based on user entitlements.

3.7 Maintain Accounts, Customers and Instruments. Add, change, and delete.

4.0 Maintain Systems

4.1 Perform Backup and Recovery

4.2 Monitor Systems. Monitor routing and workflow, audit trail.

4.3 Do Performance Tuning

4.4 Perform System Administration. Archiving to support deletion of data and maintenance of databases.

4.5 Provide Help Desk. Customer Support such as hot line, problems, issues, etc.

Process Map Diagrams show the sequence of selected processes from the Process Hierarchies as they relate to the role that performs them. Only significant processes of
special business interest appear in the following Process Maps. While any process appearing on a Process Map should also appear on the Business Process Hierarchy, the reverse is not true. Process Flow Diagrams may show:

- Business events that initiate a process
- Business results from completing a process
- Processes
- Process flow, also called process sequence
- Process flow breaks
- Iteration, optionality, and exclusivity
- Supporting comments.

In the case of our sample trading business model, a process map diagram for the Enter Order is as follows:
User wants to enter order from Market Depth

User wants to enter order from Market Watch

Select Instrument from Market Watch

Select Instrument

Select Price of Instrument

Select Enter Order Function

Enter Search Criteria

Enter Additional Info (RFQ, Order)

Order ticket received

Order ticket not entitled

Order stopped (error msg)

User wants to enter order from menu

Review Ticket (RFQ, Order)

Submit Ticket

Enter Search Criteria

Select Instrument

Select Price of Instrument

Select Enter Order Function

Submit Ticket

Order ticket received

Order ticket not entitled

Order stopped (error msg)
Process Design and Process Decomposition

While process design and decomposition was illustrated in the previous section, this sub-topic focuses on the interrelationship of the Elementary Business Processes to the Derived Logical Processes (DLPs) that link the business processes to a new software application. DLPs are not considered a further decomposition of the Business Process Hierarchy.

An Elementary Business Process is a unit of business activity, executable by one person at one time and in one place. They are depicted as elementary steps/tasks in the Process Hierarchies. The EBP’s that support the user requirements, combined with EBP’s that support the general and system management processes, comprise a majority of the Functional Specifications.

Derived logical processes (DLPs) describe what the computer system will do to automate a business process. DLPs provide shorthand for enumerating required system capabilities. In the Development Phase, specifications will be written to describe the DLP in enough detail for it to be developed.

An EBP/DLP matrix is usually developed concurrently with the Logical Process Hierarchy to connect the systems to the business by mapping the derived logical processes to the elementary business processes they support or automate. One EBP may use several DLPs, and several EBPs may reuse one DLP. This supports validation linking from Architecture Phase user requirements to Development Phase functional specifications and design specifications. A sample EBP/DLP matrix for our trading business model is shown below:

<table>
<thead>
<tr>
<th>EBP no.</th>
<th>EBP name</th>
<th>DLP name</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Subscribe Prices</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>Receive Prices</td>
<td>Add price</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Change price</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>Create Market Windows</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2.1</td>
<td>Configure Market Windows</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add Workspace</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Change Workspace</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delete Workspace</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Save Workspace</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add Window</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Change Window</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delete Window</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EBP no.</td>
<td>EBP name</td>
<td>DLP name</td>
<td>Implementation</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>----------</td>
<td>----------------</td>
</tr>
<tr>
<td>2.1.3</td>
<td>Enter Search Criteria</td>
<td>Find instrument&lt;br&gt;Check entitlements</td>
<td></td>
</tr>
<tr>
<td>1.2.2</td>
<td>Select Instruments for Market Windows</td>
<td>Add price subscription&lt;br&gt;Delete price subscription&lt;br&gt;Find price&lt;br&gt;Find User Preferences</td>
<td></td>
</tr>
</tbody>
</table>
### Business Model Execution Engines

The ideal software development environment would be one where the business model could be provided as an input to a “business model execution engine” that would take care of converting the business model into an executable software application. In that process, the execution engine would select the best combination of frameworks and reusable components based on the requirements set forth by the business model. The environment could also provide business model engineering tools that would semi-automate the process of gathering business requirements via a graphical user interface. The business requirements would then be compiled into a business model that could be provided as an input to our hypothetical “business model execution engine”. Although the current state of software environments is not as advanced, today’s technology is clearly moving in that direction. Today’s data modeling CASE tools automate the generation of logical database design from a conceptual database design. The proposed environment would be an extension of the capabilities provided in these case tools, where the tool would be able to encompass other models than the data model such as the process, event, and persistence models. Technology has clearly made a step in that direction with the advent of Enterprise Java Beans (EJB) application servers, which can automate the management of messaging, transaction, persistence and so on. Some tools (such as the mapping tools provided by Persistence Software Inc.) are dedicated to the generation of object-relational mappings. Clearly, the level of automation provided in today’s technology is still limited.

### Frameworks and Reusable Components

Most of today’s application architectures are being modeled using Object Oriented Analysis and Design techniques. The architecture object model is comprised of sub-models focused on the application, application infrastructure, and technology infrastructure. The application architecture object model combines information from the business model, data model, and content model. The business model includes the organization, location, and process models. The organization model captures the functional roles of the different types of business users. The location model identifies how business processes will be distributed geographically. The process model sets forth a set of business rules tailored to the application’s functional requirements. The data model formalizes the application’s underlying business entities and their inter-relationships. The content model focuses on the interface of the application, and encompasses navigation, presentation, and the underlying publishing mechanisms. The content model closely follows the look and feel and navigation guidelines imposed by the corporation implementing the application. The architecture object model for the application infrastructure is subsumed in a layered framework (i.e., an assembly of design patterns) that drives the grouping of application components into logical packages that provide added operational functionality. These logical packages enable the mapping of the application components onto the application infrastructure. The design of the application infrastructure framework is influenced by the business constraints imposed on the overall system design. Finally, the technology infrastructure model is subsumed in one or more instances of physical architectures that map the aforementioned logical packages onto the underlying hardware and operating system platform. The technology infrastructure model takes into account the company’s
operating standards and constraints. The physical architectures implement the technology infrastructure model and satisfy the operational requirements of the application, and application infrastructure implementations. Altogether, the application, application infrastructure, and technology infrastructure models capture the preliminary analysis and design information required to move on to detailed design, and implementation of the first release of the application.

As mentioned in the above, the resulting system is architected in three layers as depicted in Figure 1. The top layer hosts the application suite, which implements the various business functions. The middle layer hosts the application infrastructure, which provides operational support for the application suite. Finally, the bottom layer hosts the technology infrastructure, which includes hardware and operating system software. The architecture offers a global vision of the application. It describes the strategic capability choices that determine the application’s overall software quality from a business and technical standpoint. The application’s software “quality” results from meeting specific business and technical constraints. Business constraints include the support for business specific requirements (e.g., guaranteed delivery of order and order status messages in our sample trading business model). Technical constraints include accuracy, correctness, performance/latency, scalability, extensibility / adaptability, continuity / availability / reliability / robustness, security, testability, and usability. These capability choices, which provide a measure of the application's overall software quality, directly result from an attempt at fulfilling business and technical requirements. Such requirements are usually collected by various workstreams during the vision and strategy phase. At that stage, typical workstreams focus on customer facing interviews, competitive analysis, and a detailed analysis of current state technology and business processes. It is important to note that the assessment of an application’s required architectural capabilities is currently based on a limited set of requirements. Other design factors that will drive the implementation of a good application architecture are simplicity, elegance, intelligibility, well-defined levels of abstraction, and a clear separation between the interface and the implementation of each level.

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Infrastructure</td>
</tr>
<tr>
<td>Technology Infrastructure</td>
</tr>
</tbody>
</table>
Within the context of object-oriented analysis and design, modeling is often used as a synonym of analysis and it decomposes systems into collaborating objects. The Unified Modeling Language (UML) defines several models for representing systems: the class, state, use case, interaction, implementation, and deployment models. The class model captures the static structure. The state model expresses the dynamic behavior of objects. The use case model describes the requirements of the user. The interaction model represents the scenarios and messages flows. The implementation model shows the work units. The deployment model provides details that pertain to process allocation. Models are browsed and manipulated by users by means of graphical representations, which are projections of the elements contained in one or more models. Many different perspectives can be constructed for a base model - each can show all or part of the model, and each has one or more corresponding diagram. UML defines nine different types of diagrams: class, sequence, collaboration, object, statechart, activity, use case, component, and deployment diagrams. Class diagrams represent the static structure in terms of classes and relationships as model elements. Sequence diagrams are a temporal representation of objects and their interactions; they involve actors, objects, and messages. Collaboration diagrams are a spatial representation of actors, objects, links, and messages. Object diagrams represent objects and their relationships, and correspond to simplified collaboration diagrams that do not represent message broadcasts; they use classes, objects, and links. Statechart diagrams represent the behavior of a class in terms of states and transitions. Activity diagrams represent the behavior of an operation as a set of activities; they involve transitions, and actors. Use case diagrams represent the functions of a system from the user's point of view; they rely on actors, and use cases. Component diagrams represent the physical components of an application, and deployment diagrams use nodes and links to represent the deployment of components on particular pieces of hardware.

From an architecture standpoint, putting a well-adapted architecture in place is the key to successful development. Software architecture are often characterized using the following formula:

$$\text{Software Architecture} = \text{Elements} + \text{Patterns} + \text{Motivations}$$

In this equation, elements correspond to UML model elements such as classes, and objects. Patterns refer to recurring combinations of objects and classes. Patterns can be grouped in larger representations called frameworks. Frameworks are truly reusable infrastructures dedicated to the implementation of targeted applications. In the case of our sample trading application, we could envision an application infrastructure web framework to support the various application layers. Finally, the "Motivations" variable set forth in the above equation correspond to the driving forces behind the architecture implementation. In the case of our sample trading application, we would identify a set of business, and technical constraints on the application as well as a set of guidelines for designing a "good" application architecture.

As an architecture is being designed, there is more than one way to represent a system. Multiple perspectives are required. Moreover, in order to satisfy many parties, each UML
diagram type only gives a partial image of a system. As a result, the architectural model, which describes the architectural vision, must be well adapted to the representation of various constraints that the architect must take into account. A possible architectural model is the "4 + 1 view model", which relies on a combination of logical, implementation, process, deployment, and use case views as illustrated in Figure 2. The logical view describes the static and dynamic aspects of the system in terms of classes and objects. It involves a combination of class, object, sequence, collaboration, statechart, and activity diagrams. The implementation view relies on component diagrams, and is concerned with the organization of modules within the development environment. The process view represents the decomposition, in terms of execution flows (processes, threads, tasks, etc.), the synchronization between flows, and the allocation of objects and classes within the various flows. It relies on sequence, collaboration, statechart, activity, and component diagrams. The deployment view describes the various hardware resources and deployment of software within these resources. The deployment view uses component and deployment diagrams. Finally, the use case view constitutes the glue that unifies the four previous views. Use cases motivate and justify the architectural choices. They facilitate the identification of critical interfaces, they force designers to focus on the concrete problems, and they demonstrate and validate the other architectural views. The use case view involves use case, object, sequence, collaboration, statechart, and activity diagrams.

Figure 2: The "4 + 1" Architectural View Model

Sub-topic 1.4 fully illustrates our sample trading application architecture. The documentation is limited to the "core" functionality, which excludes the details of the presentation layer. The proposed application architecture documentation includes:

- A textual description of the features of the architecture (i.e., the views) and the key system requirements.
- The compromises reached and the alternatives explored.
- A high-level representation of the logical view.
- Scenarios specific to the architecture.
- Description of key mechanisms.
As the actual sample trading application is too complex to be understood, designed or implemented in one shot, it is better to implement it iteratively. Each iteration can focus on functionality increments. The object-oriented approach provides a comfortable framework to develop in an evolutionary fashion using such an iterative approach. We therefore suggest an iterative and incremental project lifecycle for our sample trading application, and we will use UML to model the system as it provides a comprehensive notation for the full lifecycle of object-oriented development.
2. Process/Entity Matrix

Meshing business processes and data entities can be done in various ways. When using the object model approach (i.e., UML, or EER data model), business processes can be represented as the methods of classes, which are used to represent the typical entities. If the data model of the language used to implement the conceptual database design is not as expressive as an object model (e.g., E-R data model), it may be necessary to use process/entity matrices to associate relevant business processes to the entities they act upon. A simple alternative to a process/entity matrix is that of a Logical Process Hierarchy. A Logical Process Hierarchy is a structure for organizing DLPs using object-based principles. It starts with the Logical Entity Model and identifies business objects to correspond to the major data entities. Each DLP/object that is used by an EBP, is noted here. The Logical Process Hierarchy is used as a reference tool to identify development components and to promote reusability.

A Logical Process Hierarchy for our sample trading application is illustrated below:
3. Conceptual Database Design

This section illustrates the database design steps for our sample trading application. It first shows how to derive a logical data model from a traditional E-R data model. It then relates the mapping of the resulting logical data model to the object model of the overall trading application.

Logical Data Model for the Sample Trading Application

This logical data model has been constructed based on data requirements derived from sample functional requirements. The mapping of entities to the functional requirements that they support is listed as part of the data dictionary. Where possible, the model has been validated by business people from the appropriate subject areas, but as requirements are still being defined, some areas will need revision as these requirements are completed.

It is also important to note that the primary intent of this model is to present the logical entities and their relationships. The attributes listed as belonging to these entities are just samples of what may be required and will have to be modified and expanded upon as business requirements are completed.

The logical data model is broken up into five subject areas based on the functions the entities contained in each support. These subject areas are: Orders and Quote Requests, Price Subscriptions, User Preferences, Privileges, and Instruments. A description of the purpose of each subject area follows, and a definition of each entity can be found in the data dictionary. Entities that appear in gray are not required for phase one.

The Entity-Relationship (ER) diagrams for the five subject areas, and the associated data dictionary are illustrated below:

Orders and Quote Requests

The model below supports both quote requests and orders. Quote requests will contain most of the same information as an order, so logically they can be stored within the same entity. Since an order or quote request can consist of orders/requests for multiple instruments, two entities are used to represent this. The order entity links all of the individual lines together and describes their dependencies, while the order line consists of all of the details needed for an order or quote request for a single instrument. The execution entity contains the results of a trade whether that is a fill or partial fill, while the quote entity contains responses to price requests. Each change in the status of an order or quote request, including rejections and cancellations, is maintained in the order status entity.
Price Subscription

A user can define price subscriptions in two ways. An individual instrument can be assigned to a market window, or a group of instruments may be assigned. These groups or lists can be user defined and do not imply that the instruments have anything in common; rather, it allows the instruments to be more easily manipulated as a group. The assignment of instruments to windows defines a subscription. Included in gray are some entities that will support parametric prices in a later phase. These entities allow for the fact that there can be many prices for an instrument if different conditions or parameters are applied, such as conditions for delivery or settlement. Entitlements to view prices are controlled through their assignment to price groups and price tiers.
User Preferences

This model defines four sets of user preferences. The first, through the user entity, identifies the user and specifies a collection of default settings that apply throughout the trading application. The next is the set of default settings that depends on the asset class that the user is working with. The default order parameters entity allows the user to specify default values that will be used unless overridden when placing an order or making a quote request. Finally, there are the entities that represent the configuration of the user’s work area. A user can create multiple workspaces, each containing a set of windows (order queue market watch, market depth). Each of these windows can in turn be configured to display a selected set of columns and their positions, sort orders, and filters.
Privileges

The proposed data model would need revision as business requirements in this area become better defined. There are three main groups of privileges: price entitlements, trading permissions, and administrative privileges. In this model, all three groups are represented through the privileges entity. This entity associates an action, such as ‘place order’, with a target object, such as ‘product X’. Access will be controlled to objects such as price tiers, price groups, order books, accounts, and products. These privileges can be assigned directly to a user, or can be assigned first to roles for easier administration. Users can then be assigned to these roles. In addition there is a set of roles and privileges that are associated with a client (firm). Client users must then be assigned a subset of the privileges and roles available to the client. It is also possible that where a client is a subsidiary of another client, the subsidiary can inherit the privileges of its parent. Alternately, it can be given its own set of privileges.
**Instruments**

The instrument model is needed to support searching for instruments as well as for identifying an instrument to a back end system in an order. This model will need to evolve as new requirements are being added. A couple of different instrument groupings are represented here. The first is through product and asset class. These are used in determining defaults and for controlling access within the application. They also support classifying and searching for instruments. The other grouping method allows a hierarchy of groups that can be defined either by the company developing the trading application or by the users themselves. These custom groups are not used to classify instruments or imply any similarities between instruments in the group; rather, they are used to allow users to easy find and manipulate commonly used sets of instruments. A structure also exists for representing multiple identifiers by which an instrument may be known.

**Data Principles for the Sample Trading Application**

- All data that needs to be persisted will be stored within a database.
- The logical model needs to conform to naming and abbreviation conventions.
- The application needs to maintain audit logs for auditing, reporting, and recovery purposes.
- All persistent data must be backed up regularly.
- Lost sessions and failed messages can be reconstructed from persistent data.
• Access to data within the application will be controlled according to the company’s security standards for data stores.

Data Assumptions for the Sample Trading Application

• Database replication:
  ✓ For locations that handle same client base, databases may need to be shared and or synchronized.
  ✓ Databases for servers that are in different countries will not replicate each other’s data.
• The back end system is responsible for mapping a trading account to an account within the back end system.
• The trading application will receive instrument data from back end systems and maintain a version locally.
• Backends will maintain client credit data.
• The following estimations are given:

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Update Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>10K</td>
<td>100/day</td>
</tr>
<tr>
<td>Accounts</td>
<td>5K</td>
<td>10/day</td>
</tr>
<tr>
<td>Firms</td>
<td>5K</td>
<td>10/day</td>
</tr>
<tr>
<td>Entitlements</td>
<td>500K</td>
<td>5K/day</td>
</tr>
<tr>
<td>Price requests</td>
<td>50K/day</td>
<td>2/second</td>
</tr>
<tr>
<td>Orders</td>
<td>20K/day</td>
<td>1/second</td>
</tr>
<tr>
<td>Peak Hour</td>
<td>15% of daily global volume</td>
<td></td>
</tr>
<tr>
<td>Avg. price subscriptions/user</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Price update rate</td>
<td></td>
<td>1/second/product</td>
</tr>
<tr>
<td>Total price update rate</td>
<td></td>
<td>300K/sec</td>
</tr>
<tr>
<td>Price update size</td>
<td>50 bytes = ~500 bits (no compression)</td>
<td></td>
</tr>
<tr>
<td>Total price update data volume</td>
<td>150 Mbits/second</td>
<td></td>
</tr>
</tbody>
</table>

Client distribution: North America 40%, Europe 40%, Asia 20%

Constraints for the Sample Trading Application

• Due to the high volume up price updates, price data will need to be stored in an in-memory database.
• Similarly, the high number of references to entitlements means that a copy of these entitlements needs to be stored in an in-memory database, while a permanent copy is persisted in a relational database.
• If Tibco products are used, we may need to use a Sybase database as DB2 is not fully supported.
• The system must be fault tolerant.
• Currently, some business requirements are still being developed. As a result, data requirement in the following areas are incomplete:
Object Model for the Sample Trading Application

As mentioned earlier, the architectural model chosen for the sample trading application is the "4 + 1 view model", which relies on a combination of logical, implementation, process, deployment, and use case views. The logical, use case and process views are handled as part of the application modeling phase. The implementation view is addressed as part of the application infrastructure modeling phase, and the deployment view is addressed as part of the technology infrastructure modeling phase.

Use Case View

The use case view constitutes the glue that unifies the logical, process, implementation, and deployment views. Use cases motivate and justify the architectural choices. The process model and associated process maps described earlier give a conceptual representation of the use case view of the sample trading application object model. In the UML notation, the use case view involves use case, object, sequence, collaboration, statechart, and activity diagrams. These diagrams should be developed incrementally as needed as part of the detailed design.

Logical View

The logical view describes the static and dynamic aspects of the system in terms of classes and objects. The logical view overlaps the class and state models, as the class model is focused on the static structure, and the state model focuses on the dynamic behavior of objects. In the UML notation, the logical view involves a combination of class, object, sequence, collaboration, statechart, and activity diagrams.

The UML notation supports model elements and visual elements (aka., diagram). The visual elements provide textual or graphical projections that facilitate the manipulation and representation of model elements. Classes and objects are examples of common model elements used to represent abstractions of a system being modeled. Packages are common model elements that provide a general mechanism for partitioning models and grouping model elements. The sample trading application architecture can be expressed as a hierarchy of inter-dependent packages. The package hierarchy is follows:

Enterprise Applications Suite
  Channels
    Web Portal
      Web Interface
        Web View
The following class diagram illustrates some of the classes would populate the above packages. More diagrams would need to be developed incrementally as part of the detailed design.

*Business Object Model*
**Component Manager Package**

The process view details execution flows (processes, threads, tasks, etc.), synchronization between flows, and allocation of objects and classes within the various flows. The process view overlaps the state model and the interaction model, as the state model focuses on the dynamic behavior of objects, and the interaction model focuses on scenarios and message flows. The process model and associated process maps described earlier give a very high-level conceptual representation of process view of the sample trading application object model. In the UML notation, the process view relies on sequence, collaboration, statechart, activity, and component diagrams. These diagrams should be developed incrementally as needed as part of the detailed design, which would take place early on in the detailed design phase. The detailed design phase would also focus on the specification of events, exits, errors, and detailed sequence diagrams including preconditions, triggers, and responses. At this stage of the analysis and design phase, we would focused on the interaction model and would develop a high-level message specification, and a preliminary message hierarchy. We would also put together a conceptual representation of message flows within the system. Additional work products would need to be developed incrementally as part of the detailed design phase.
**Application Model Implementation View**

The implementation view is concerned with the organization of modules within the development environment. In the UML notation, the implementation view relies on component diagrams. These diagrams will be developed as part of the detailed design. At this stage of the Analysis and Design Phase, a Logical Architecture Diagram would be provided. It would act as a conceptual illustration of the implementation view that would serve as a starting point for the detailed design of the architecture implementation view. The diagram would illustrates the proposed logical components of the sample trading application, and their mapping to an application framework that would meet the business and technical constraints imposed on the overall application design.

**Deployment View**

The deployment view describes the various hardware resources and deployment of software within these resources. In the UML notation, the deployment view uses component and deployment diagrams. These diagrams would be developed as part of the detailed design. A Physical Architecture Diagram would provide a conceptual illustration of the deployment view that would act a starting point for the detailed design of the architecture deployment view.