Module
INTRO

Introduction to BEA Tuxedo

At the end of this module you will be able to

➢ describe the main features of Tuxedo
➢ understand the client-server communication mechanisms used in Tuxedo
➢ identify benefits of the Tuxedo architecture
➢ describe some of the extended Tuxedo features

This Course covers BEA Tuxedo product versions 6.5, 7.1, and 8.0. Almost all the topics covered in the course are applicable to all these product versions. Features that are not available in earlier Tuxedo versions are identified as such when they are presented.

This module is intended to introduce the BEA Tuxedo product and architecture and discuss features at a high level so that you are aware of them and their applicability.
The Application Server Architecture (n-Tier Client Server)

Application Server Architecture (or 3/n-Tier Client/Server Model)
The application server Client/Server model implements a “three-tier application architecture”, also referred to as “n-tier” as the tiers can be extended with multiple application and database layers.

The Client implements presentation services
- Displays input screens to the user, collects user input, and displays results

The Application Server implements business logic
- Messages (Service Calls) are passed from client to application server
- Business logic is confined to the application server

The Database server hosts the Data Base Management System (DBMS)
- Focuses on what it does best - managing data

Benefits
- Improved performance
- Modular application design
- Location independence of business function
- Extensible (scalable) environment
- Reduced network traffic
- Business function request not dependent on implementation details of the function
Transaction Processing (TP) Applications

On line transaction processing (OLTP) applications typically involve many end-users at terminals or workstations entering transactions that are critical to the operation of the business. In the Internet environment, most transactions for a particular application can originate from a web browser.

Transaction Processing applications have the following general characteristics:

- numerous (ten’s) to very large numbers (ten’s of thousands) of similar users
- predictable input (filling in data on a form)
- many short user interactions in a short period of time
- transactions are used to coordinate instantaneous access and updates to large shared databases

Transaction Processing (TP) Monitors were designed to support very large numbers of terminals, client workstations. Since not every user is actively submitting a transaction all the time, the actual transactions taking place at any one time are very much less and so fewer application server resources and database connections can be used at the back-end.
The Tuxedo Approach

The Tuxedo system provides application services to multiple clients by multiplexing client service requests to a controlled number of application servers supporting the particular service. The application servers, in turn, provide access to databases or other services.
What Is BEA Tuxedo?

- Middleware (between clients & servers)
- Transaction Processing (TP) Monitor
- Infrastructure for distributed applications
  - multi-tier, client-server environment that is centrally managed
- The name Tuxedo comes from it’s origins
  - Transactions for UNIX (TUX) + distributed client-server communications (Extended for Distributed Operation)

What is BEA Tuxedo?

BEA Tuxedo is a “middleware” product that provides an infrastructure for distributing applications across multiple platforms, databases and operating systems by utilizing message-based communications and if desired, distributed transaction processing.

Middleware systems divide application programs into client and server components. The function of middleware is to effectively distribute processing among multiple servers, manage distributed transactions, and ease the integration of multiple database platforms. Middleware systems are commonly used to build on-line transaction processing (OLTP) systems; however, not all forms of middleware are designed or able to support OLTP.

BEA Tuxedo is a mature product based on years of development from a diverse group of technology companies, including AT&T, Unix System Laboratories (USL), Novell Inc., and finally BEA Systems, Inc. It is both a development platform and an execution platform, and can be considered as an extension to an operating system.

The BEA Tuxedo system provides:

- A standard for creation and central administration of distributed on-line transactional applications in a heterogeneous client/server environment.
- The fundamental underpinnings for creating, managing and maintaining reliable, high-performance, easily-managed distributed systems.
- Ease-of-use for application developers, since they do not need to know the details about server locations, routing, or kinds of platforms, all of which are transparent to the developer.
Core Features of Tuxedo

Clients and servers are the application processing components of a BEA Tuxedo system. Server processes provide one or more named services. Client processes request services without having to know where they are located. The Name Service feature provides a “directory” of services that results in the request being routed to one of the servers providing the service.

Clients and servers communicate by sending messages. When the clients and servers are distributed over different machines, Tuxedo makes the networking infrastructure connecting client and server machines transparent to the client-server request/response model. Programmers therefore do not have to worry about where the service is located or what the underlying network protocols are. The applications code remains the same whether the clients and servers are running on a single machine or distributed over multiple machines.

The Tuxedo product provides utilities to configure and operate the distributed Tuxedo application (or domain). The same set of administration programs are used whether the Tuxedo application is deployed on a single machine or multiple machines.

Tuxedo also supports the function of a Transaction Monitor or Manager and acts as the transaction coordinator for distributed transactions that utilize Resource Managers. A distributed transaction is one that involves multiple Databases (generically termed Resource Managers).
Benefits of the Tuxedo Architecture

- Enables sophisticated messaging features such as synchronous, asynchronous, and parallel or pipeline operation
- Makes the application service location transparent to the programmer
- Improves database efficiency by sharing database connections at the back-end
- Centralizes configuration management and system administration tasks
- Facilitates dynamic system administration

Benefits of the Tuxedo Architecture

All the benefits of the Application Server model (enhanced performance, modularity, extensibility, reduced network traffic, service independence) are realized in the Tuxedo architecture. It enables high performing applications and makes very efficient use of system and network resources.
**Components of a Tuxedo Application**

- Bulletin Board Liaison (Tuxedo administration process)
- Bulletin Board, Message Queues (Tuxedo uses operating system resources)
- Client, Server (Tuxedo application processes)

**Basic Components of a Tuxedo Application**

The basic components of a basic Tuxedo application running on a single machine are:

- **Message Queues**: Communication between clients and servers is performed via operating system memory-based message queues provided by the UNIX Inter-process Communication (IPC) mechanism.

- **Bulletin Board (BB)**: The Bulletin Board is a shared memory segment available to all the processes of the Tuxedo system. The Bulletin Board holds configuration and dynamic information for the system. The Bulletin Board is where all the application configuration and dynamic processing information is held at run-time. The Bulletin Board translates a service name to a specific server. When a service is called, the Bulletin Board is used to look up what servers offer the requested service. Based on this information, the caller places the request data on the request queue of the correct server.

- **Bulletin Board Liaison (BBL) process**: A Tuxedo administrative process that monitors the other components.

- **Tuxedo Client processes**: Executable programs which call services through the Tuxedo system (and are usually customer-developed programs).

- **Tuxedo Server processes**: Executable programs which offer named services through the Tuxedo system. These are normally customer-developed programs.

The UNIX IPC subsystem makes the following resources available to applications: shared memory, message queues, and semaphores. The Tuxedo software uses these IPC resources directly on UNIX systems. On operating systems such as Microsoft Windows NT, that do not implement the UNIX IPC mechanism, Tuxedo provides the software to simulate these IPC features using the native operating system’s functions.
The Bulletin Board and Bulletin Board Liaison

The Bulletin Board (BB) is where all the application configuration and dynamic processing information is held at run-time.

- The Bulletin Board translates a service name to a specific server. When a service is called, the Bulletin Board is used to look up what servers offer the requested service. Based on this information, together with any specialized routing criteria, the caller places the request data on the request queue of the correct server.

- Server Location Transparency provided by the BB allows the application to be developed independently of deployment. Therefore, development and deployment costs are minimized.

- The Bulletin Board also holds dynamic information about the state of the application, such as how many requests are on a given server’s queue and how many requests have been processed.

The Bulletin Board Liaison (BBL) process is a BEA Tuxedo supplied administrative server which performs periodic health checks and coordination functions of all the components of the system.
**Tuxedo Clients**

- **Native Clients**
  - clients running in a configured machine (one that hosts a Bulletin Board and BBL); can also access services in another configured machine

- **Remote Clients (/WS clients, Jolt clients)**
  - support for clients running in a machine that does not host a Bulletin Board (BB) and BBL; supported through proxy clients
    - supported by the /WS subsystem (/WS client)
    - supported by the BEA Jolt product (Java client)

- **BEA WebLogic Server applications can also access Tuxedo services**

---

**Clients**

Clients are not configured in a Tuxedo configuration. They start up as independent entities but must connect to a Tuxedo application in order to obtain services.

These client application processes run on a machine that is part of the Tuxedo machine/s configuration for a domain – these machines have a BB/BBL. Native clients can make ATMI calls to a service that may be offered by a server on the same or a different machine defined in the configuration.

**Workstation and Jolt Clients**

Workstation and Jolt clients are Tuxedo client applications that run on a remote machine that is not part of the Tuxedo machine/s configuration. These clients are supported through proxy clients that run on a configured machine.

**WebLogic-Tuxedo Connector**

BEA WebLogic-Tuxedo Connector software enables BEA WebLogic Server 6.x applications to be integrated with BEA Tuxedo 8.0 applications.
The BEA Tuxedo Implementation of the C/S Model

The diagram shows the a logical view of the Tuxedo application programming interface, called ATMI for Application-to-Transaction Manager-Interface. The ATMI functions are implemented as Tuxedo libraries linked with the application code.

The execution flow is:

1. The client application program uses an ATMI function to allocate a data buffer and puts some data in it.
2. The client application program uses an ATMI function to call a named service and passing it the data buffer.
3. Transparent to the programmer, the ATMI code in the client then consults the Bulletin Board to find the (queue address of a) server which offers the requested service.
4. Transparent to the programmer, the call information then is placed on the server’s request queue by the client ATMI code from the client application program.
5. Transparent to the programmer, the ATMI code in the server process then dequeues the call packet, and calls the named application service routine to process the request.
6. The application service routine is invoked and after processing the request, the application service routine returns control to the ATMI code in the server, which places the service’s reply information on the client’s reply queue.
7. The ATMI code in the client process then dequeues the reply information and returns the processing results to the client application program.
**Messaging & Communications**

- User data is passed between clients and servers in Tuxedo message buffers
  - Typed Buffers identify the type of data in the buffer (for example, a C data structure)
- Buffers are passed between clients and servers using IPC or the network (TCP/IP)
  - for inter-machine communication, data is converted to a neutral format for transmission
  - data is converted and transferred transparently between different platforms
  - the programmer does not have to convert data formats (for example, a floating point number on an Intel machine sent to a Sparc machine)

All communication within Tuxedo is accomplished by transferring messages, in buffers, between the message queues of the clients and servers.

All communication within a machine is performed using operating-system supported Inter-Process Communication (IPC) queues.

Communication between Tuxedo components on different machines is performed using a network Bridge Process on each machine which link the machines together using protocols such as TCP/IP.
Data Management (Buffer Types)

Tuxedo defines several buffer types for efficient and transparent data transfer. The slide shows how the different Tuxedo buffers types can be used by applications. The buffers in the applications are converted into messages for transferring the buffer data between Tuxedo processes.
Message Paradigms

- Clients and/or servers may use any of the following communication models
  - request/response
    - synchronous
    - asynchronous
  - conversation
  - unsolicited notification
  - publish and subscribe
  - store and forward

Described in more detail on following slides

Message Models

Tuxedo offers a set of Application-to-Transaction Monitor Interface (ATMI) functions that support a number of communication models. These models along with a brief definition are:

Synchronous
Clients may issue a call and then block waiting for the reply.

Asynchronous
Clients may issue a call but not wait for the reply. At a later time, they may retrieve the reply.

Conversational
Clients may establish a “dialogue” with one or more conversational servers. The server and client then exchange data until the conversation is terminated. The server is tied to the client for the duration of the conversation.

Unsolicited Notification
Clients and Servers may send unsolicited messages to (other) clients at any time.

Publish and Subscribe
Clients or Servers may subscribe to events and specify an action to be taken by Tuxedo when that event occurs. Events can be defined by users or be pre-determined Tuxedo system events.

Store and Forward
Clients may store a request in a queue for deferred processing and retrieve a reply at a later time. These calls are available with the BEA Tuxedo /Q feature.
Request/Reply Model

To implement the request/reply architecture, Tuxedo uses a queued model. Queues are the key to connectionless communication. Each server is assigned an Inter-Process Communication (IPC) message queue called a request queue and each client is assigned a reply queue. Therefore, rather than, say, establishing and maintaining a connection with a server, a client application need only check for messages on a queue.

The example on the slide shows how requests and replies flow through the system in this model.

There are two types of service requests that use this model:

- **Synchronous requests** - Clients may issue a call and then block (suspend execution at that point) waiting for the reply
- **Asynchronous requests** - Clients may issue a call but not wait (continue execution) for the reply. At a later time, they may retrieve the reply
Conversation Model

Tuxedo implements the data request architecture as a conversation between a client and a server. In this form of communication, a virtual connection is maintained between the client and server. Just as in a conversation between two people, a number of different messages pass back and forth between the two entities until a conclusion is reached. Over the course of the communication, both sides “remember” the point (or state) of the conversation so that relatively long operations, like ad hoc queries, reports, or file transfers, can be supported.

Tuxedo provides a conversational Application Programming Interface (API) that applications use to connect to the server, send and receive messages, and end the conversation. The API specifies the formats of messages and the allowable points at which applications may send messages. In general, it is a protocol violation for both sides participating in a conversation to be waiting for a message at the same time. This is called half-duplex in data communication terminology.
Unsolicited Messaging Paradigm

Tuxedo supports some communications paradigms which do not require a process to be listening to receive another’s request. You can design your system so that the occurrence of certain conditions is noted and communicated to other parts of the system spontaneously, even when they are not requesting data.

Ordinarily, clients are senders, not receivers of data. When clients receive data without first asking for it, this event is called *unsolicited messaging* or *unsolicited notification*.

Unsolicited notification can be used to send a message with or without guaranteed delivery to an individual client (*point-to-point notification*) or to broadcast information to a group of clients (*broadcast notification*). For example, a server may alert a single client that the account about which the client is inquiring has been closed. Similarly, a server may broadcast a message to all the clients on a machine to remind the users that this machine will be shutting down for maintenance at a specific time.

Since managing the varying number of entities that can generate and/or receive multiple unsolicited notifications can become quite complex over time, Tuxedo also provides a way of handling this scenario as we will see next.
Publish and Subscribe Model

The BEA Tuxedo Event Broker supports the Publish and Subscribe paradigm, in which an arbitrary number of suppliers can post messages for an arbitrary number of subscribers. This allows client and server processes to communicate with one another based upon a set of “triggers” that are known to both

Event generators (publishers) inform the broker of events as they occur, and the broker passes the notification on to the event targets that have registered their interest in such events with the broker. In a sense, the broker is like a newspaper delivery person who receives newspapers from the printer and delivers them only to those customers who have paid for a subscription. This paradigm is sometimes referred to as publish and subscribe.

In the Tuxedo environment:

1. A process registers a subscription with the Event Broker, indicating interest in some event.
2. Some time later, the event broker is notified by another process in the system that an event has occurred. This is known as posting an event.
3. Then, the Event Broker will publish the occurrence of this event to any entity defined by the subscription.

Event notification can be accomplished via several mechanisms, including the following:

- Asynchronous call to a service
- Unsolicited message to a client
Store and Forward Message Queues: The /Q Queuing Subsystem

Store and Forward

There are two important aspects of the store and forward paradigm:

- The target of the message (the receiver) does not have to be running at the time the message is "sent" as it is stored in a queue for later recovery by the target
- The target of the message is free to recover the request at some (later) time most appropriate for processing the message

The store and forward paradigm is a variation of the request/reply paradigm, where the processing by the server is to be deferred for a period of time. Such deferral may be due to a deliberate aspect of the application’s design, or it may be a consequence of a service being unavailable. BEA Tuxedo supports the store and forward paradigm through its stable-storage and shared memory queuing facility (/Q).

The slide illustrates a typical example of a store and forward data flow. An example of use of the Store and Forward paradigm is to capture an input request when the database is not “up” at the time but the transaction can be processed later and the result can be communicated back to the originating end-user by some means, for example an e-mail message or sent via postal mail.

With the Tuxedo 6.5 and earlier releases, persistent storage (disk) is the only mechanism for the store and forward messages to guard against loss of messages. However, the feature has been extended in Tuxedo 7.1+ to allow for the message store to be memory-based instead of disk. While this has the obvious disadvantage of loss of messages in case of system failure, it also provides a very fast store/retrieve facility for some type of applications that can tolerate some message loss in the event of a system hardware or software failure.
Work Distribution

- **Load Balancing**
  - Request is queued to least busy server
- **Multiple Server Single Queue (MSSQ) set**
  - Similar to having a single customer line in a bank versus multiple lines
- **Data Dependent Routing (DDR)**
  - route client request based on the value of a data field in the request buffer
- **Service Priority**
  - assign relative priorities to service requests; higher priority requests get processed sooner

**Work Distribution**

Tuxedo distributes work throughout the system based on various criteria and methods:

- **Load Balancing** distributes data to the queue for the program best able to process the data based on the statistical load in the system. These statistics are accumulated based on weighting factors assigned to the services which are being called.

- **Data Dependent Routing (DDR)** is a feature of Tuxedo to enable a client to send requests for a service to an appropriate instance of the service routine, depending on the content of the request’s data. The bulletin board includes routing tables that specify what subset of data each service is capable of handling. The routing tables are set up as part of the BEA Tuxedo configuration specification.

- Data Dependent Routing is often used to partition a database horizontally, or to create several different versions of the same named service to handle requests differently, depending on the data value (for example, withdrawal requests for amounts over $5000.00 may be routed to a version of the service that imposes additional security).

- **Priority Queueing** is also used. This allows the system architect to better balance response times to improve performance. Services can dequeue messages from their request queue in priority order. There is a “starvation prevention” mechanism in place to prevent low-priority messages from stalling on the queue; every tenth message is dequeued in FIFO order regardless of priority.
Basic Security

- Clients
  - Authentication *(are you a valid user?)*
    - Passwords, Tuxedo-defined users/groups
  - Authorization *(what are you allowed to do?)*
    - Simple Access Control Lists (ACLs) can be set up to control access to resources such as Services

- Servers
  - are started by the Tuxedo BBL; assume the operating environment of the Tuxedo Administrator and are assumed to be “trusted”

- A simple Authorization Server is provided with the Tuxedo product

Tuxedo provides a security plug in that enables users to plug in a customized, user-written Security service. A simple Authorization server is also provided with the Tuxedo product and can be used to perform simple authentication and authorization checks.
**Link Level Encryption**

Link Level Encryption (LLE) software is distributed with Tuxedo product; encrypts data sent over network links. In Tuxedo 6.5 and 7.1, use of the LLE is a separately licensed software option; LLE usage is included in the Tuxedo 8.0 license.

**Extended Security** (not available in Tuxedo 6.5)

Security was extended with the Tuxedo 7.1 release to include new capabilities referred to as the Security Provider Interface (SPI). It is a framework that allows security products from companies that specialize in security for distributed applications, to be used with Tuxedo applications.

Many businesses today have standardized on a particular vendor’s security package for their corporate security architecture (such as Entrust). With Tuxedo’s new security framework, BEA customers can leverage the expertise and capability of these security vendors for their Tuxedo applications, while adhering to the corporate security standard for their business.

**Tuxedo Security API and SPI**

A Tuxedo security API can be used with the third-party security product that is integrated into Tuxedo using the Tuxedo-provided Security Provider Interface. The API supports the use of:

- Digital Envelopes: Encryption of data at the application level
- Digital Signatures: Data integrity and proof of message originator’s identity
A major feature of the Tuxedo product is that it implements transaction management for distributed transactions. This allows application programmers to define the scope of a transaction through the application programming interface or through configuration, and Tuxedo will ensure the all-or-nothing character of the work within the scope of that transaction. To do this, Tuxedo uses the industry standard XA protocol.
Configuration and Administration

- Centralized Configuration
  - One configuration file describes the entire Tuxedo application/domain; for a multiple machine configuration, all the machines are also defined in this configuration file
  - One Master machine controls the domain

- Centralized Administration
  - The Tuxedo application (domain) is started from the Master machine; the BBL and other configured servers on the other machines are also started up
  - Control commands only be performed on the Master machine

Configuration and Administration

The Tuxedo system allows you to configure your application to run on a number of different machine and network environments to match the business needs (reliability, scalability, distribution, location) of the application.

Among the strengths of Tuxedo are the operations, administration and maintenance tools that are available when a distributed application is active. In a Tuxedo application, administration functions are performed on a single node, designated as the Master machine. This is the machine on which you configure the application, initiate start-up and shutdown, and perform administrative tasks during runtime. Working from the Master machine, the system administrator has control over configuration, fault management, security management, monitoring, and performance management.

Control commands can only be performed on the Master machine but some monitoring can also be done on the non-Master machines.

A domain is a runtime instance of a Tuxedo application that is defined by a Tuxedo configuration file. A domain may contain a single system or multiple systems defined in the configuration.
Single Domain -
Multiple Machine (MP) Architecture

Single Domain Architecture

The following are the configurable machine dimensions of a Tuxedo domain:

- Single machine per domain
- Multiple machines per domain

A distributed Tuxedo configuration is a centrally administered system that operate on multiple machines. To accomplish this, these two additional components are required:

- **Bridge**: Bridges are Tuxedo supplied servers which send and receive messages between machines, and route them to local components (literally, local server queues).

- **DBBL**: *(Distinguished Bulletin Board Liaison process)* is responsible for ensuring that the BBL servers on each machine are alive and functioning correctly. This server runs on the ‘Master’ machine of an application and communicates directly with any administration facility.

If the Tuxedo application is configured for a single machine, no DBBL or bridges are present.
Multi-Domain Architecture

Multiple Tuxedo domains can be inter-connected, allowing clients in one domain transparent access to services physically located in remote domains. This is accomplished using a concept similar to a Bridge, called a Gateway.

The DOMAIN feature provides the following functionality:

- Scalability and Modular Growth - Application programmers can structure their application for modularity, isolation of failures, and independent growth. Interoperation with other TP applications can be easily achieved by adding the description of the services that a remote application uses to the DOMAIN configuration.

- Transparency and independence - Applications are totally unaware of service distribution. Client application programmers do need to know the implementation changes made to a service, the location of a service, or network addresses.

A client can transparently access a service on either:

- The same machine node
- Another machine in the same domain
- Another machine in a remote domain
Application Administration

- **Command Line Utilities**
  - to perform administrative tasks such as
    - boot the application configuration
    - startup or shut down some or all application servers
    - monitor and view statistics of a running application
    - shutdown the entire application

- **GUI Administration Console**
  - a Java applet that runs in a web browser such as Netscape Navigator, Internet Explorer
  - same set of facilities as the command line utilities but with a graphical user interface
  - easy to use; must be configured initially

Administration Utilities

The command-line utilities are used to activate, configure and manage, and deactivate, the application.

The Tuxedo Administration Console is part of the standard Tuxedo product.
The Tuxedo Administration Console provides a graphical user interface (GUI) to the Tuxedo administration utilities. The Administration Console is implemented as a Java applet that is downloaded and run in a web browser - Netscape Navigator and Microsoft Internet Explorer.

The slide shows a sample display of the Administration Console.
Module Summary

- In this module we discussed the BEA Tuxedo product, a high level view of the architectural and administration features, and a review of some extended features.