BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to Middleware

Madhu Venkat

Guest Faculty
BITS, WILP

In this segment

Introduction to Middleware

- What is a Distributed Transaction ?
- What is Middleware ?
- Middleware in Distributed computing
- Different forms of Middleware

What is a Transaction ?

Transaction

- “A transaction symbolizes a unit of work performed (comprising of multiple operations if needed) within a system (distributed or monolith) and treated in a coherent and reliable way independent of other operations.”
- Properties of a Transaction:
  - Atomic
  - Consistent
  - Isolated
  - Durable
- All or Nothing mode of operation
- Ex: Funds transfer (debit remittance account and credit beneficiary account)

What is Middleware ?

What is it ?

- “Middleware is the software that connects software components or enterprise applications in a distributed system”.
- Examples: Enterprise Application Integration software, telecommunications software, transaction monitors, and messaging-and-queueing software.

Middleware in Distributed computing

What does it do ?

- Lies between the operating system and the applications on each side of a distributed computer network
- Hides the intricacies of distributed applications
- Hides the heterogeneity of hardware, operating systems, and protocols
- Provides uniform and high-level interfaces used to make interoperable, reusable and portable applications
- Provides a set of common services that minimizes duplication of efforts and enhances collaboration between applications

Middleware – common forms

Commonly used Architectures of Middleware

- Sockets
- Remote Procedure Calls
- Distributed Object Oriented Components (Ex: ORB)
- Message Oriented Middleware (Message Queues/ Enterprise Message Bus etc.)
- Service Oriented Architectures
- Web services (Arbitrary / RESTful)
- SQL-oriented data access
- Embedded middleware
- Cloud Computing

Thank You!

In our next session:

Socket Data structures

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to Enterprise Bus (EB)

Madhu Venkat

Guest Faculty
BITS, WILP

In this segment

Introduction to ESB

- ESB – Overview
- Capabilities of an ESB
- Why ESB is important?
- ESB challenges

Defining the ESB

- "A style of integration architecture that allows communication via a common communication bus that consists of a variety of point-to-point connections between providers and users of services."
- "An infrastructure that a company uses for integrating services in the application landscape."

What is an ESB?

- A bus based architecture to enable integration of multiple heterogenous systems in an efficient manner.
- A sophistication over traditional P2P integrations
- ESB acts as an intelligent middleware to enable communication among different systems/application.

ESB Architecture

Capabilities of ESB

- **Routing** – Logically route data to different integrated servers/applications.
- **Orchestration** – Combining functionalities of discrete services to provide aggregated services.
- **Transformation/Translation** – Data formats transformations and translations as per connected application needs. Example (SOAP to Rest , JSON to XML conversions)
- **Mediation/Message Enrichment** – Customize, enrich, re-format messages as a middleware/mediator.
- **Transportation/Protocol conversions** – Transportation of data between connected systems as per supported transport protocols.
- **Non-Functional Consistency** – Enforcement of security policies, monitoring, management , HA/FT

ESB Blueprint

When Should an ESB be used?

- The use of an ESB is worth considering when three or more applications or services need to be integrated.
- A simple point-to-point integration is significantly easier and much more cost-effective when connecting two applications.
- An ESB can also be worthwhile if services are going to be incorporated from external service providers over which the company has no control.
- The ESB can then be used to monitor the service level agreements that the external provider guarantees.
- If many different protocols, such as HTTP, SOAP, and FTP, are to be used and standardized to one protocol like SOAP, the ESB can perform the necessary protocol transformation.
- If services are to be consistently incorporated into an architecture to be able to receive, process, and produce messages, then the use of ESB is also suitable.

Why ESB is important?

- Efficient decoupling of multiple systems/applications.
- Better use of integration capabilities.
- Improved time to market and conditions.
- Better synchronization for upgrades, monitoring and management of integration layer

Challenges with ESB

- ESB itself can become a bottleneck and a single point of failure.
- ESBs have associated regression of risks with dozens of heterogenous systems/applications connected.
  - Changes/upgrades to a single fat middleware may have ripple effect on many functionalities.
- ESB products have a learning curve involved and lack of professionals in the market.

ESB tools

- IBM Websphere
- Microsoft BiZtalk
- Oracle ESB
- Mule ESB
- RedHat JBOSS Fuse

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to Enterprise Application Integration (EAI)

Madhu Venkat

Guest Faculty
BITS, WILP

In this segment

Introduction to Enterprise Application Integration

- EAI – Overview
- Message Channels
  - Point to Point messaging
  - Publish-subscribe model

Introduction to EAI

- Enterprise application integration allows diverse applications in an enterprise to communicate with each other to achieve a business objective in a seamless reliable fashion irrespective of platform and location of these applications.
- EAI comprises message acceptance, transformation, translation, routing, message delivery and business
- To achieve this,either Hub/Spoke or Bus Architecture can be used.

Hub/Spoke Architecture

- Hub has the centralized integration engine, which makes system with this architecture easy to manage and maintain but loses on scalability.

Bus Architecture

- In the bus architecture, adapters have integration engine and run on the same platform on which source and target applications run; and therefore it scales better but it is difficult to maintain.

Enterprise Application Integration (EAI)

Introduction

- An integration framework composed of a collection of technologies and services which form a middleware or "middleware framework" to enable integration of systems and applications across an enterprise
- *"unrestricted sharing of data and business processes among any connected application or data sources in the enterprise" - Gartner*
- Application of EAI
  - Data integration
  - Vendor/3rd party independence
  - Common façade
- Patterns
  - Mediation
  - Federation

Enterprise Application Integration (EAI)

Message Channels - Overview

- Message Channel works as a logical addressing system in Messaging
- Messaging Channel acts as a medium of communication between two specific applications in a complex enterprise, where in the sender application selects which particular ”channel” of the messaging system to use, that designates the target application/ group of applications.
- Design challenges
  - One to one or one to many
  - Data type channel
  - Invalid and dead message handling
  - Crash proof design / Guaranteed delivery
  - Non-messaging client access - Channel adapter
  - Communications backbone – Message bus

Enterprise Application Integration (EAI)

Message Channels – Example (JMS)

- Configuring Message channels (Queues and Topics)

```
j2eeadmin -addJmsDestination jms/mytopic topic
j2eeadmin -addJmsDestination jms/myqueue queue
```

- Accessing Message channels

```
Context jndiContext = new InitialContext();
Queue myQueue = (Queue) jndiContext.lookup("jms/myqueue");
Topic myTopic = (Topic) jndiContext.lookup("jms/mytopic");
```

Enterprise Application Integration (EAI)

Point to point channel

- Point-to-Point Channel ensures that only one receiver consumes any given message, irrespective of number of receivers in that channel
- Increases scalability in multi consumer applications
- JMS Implementation uses 'Queue' interface to implement Point to point channel
  - QueueSender – send messages
  - QueueReceiver – receive messages

Enterprise Application Integration (EAI)

Point to point channel – Example (JMS)

- Sending message

```
Queue queue = // obtain the queue via JNDI
QueueConnectionFactory factory = // obtain the connection factory via JNDI
QueueConnection connection = factory.createQueueConnection();
QueueSession session = connection.createQueueSession(true, Session.AUTO_ACKNOWLEDGE);
QueueSender sender = session.createSender(queue);

Message message = session.createTextMessage("The contents of the message.");

sender.send(message);
```

Enterprise Application Integration (EAI)

Point to point channel – Example (JMS)

- Receiving message

```
Queue queue = // obtain the queue via JNDI
QueueConnectionFactory factory = // obtain the connection factory via JNDI
QueueConnection connection = factory.createQueueConnection();
QueueSession session = connection.createQueueSession(true, Session.AUTO_ACKNOWLEDGE);
QueueReceiver receiver = session.createReceiver(queue);

TextMessage message = (TextMessage) receiver.receive();
String contents = message.getText();
```

Enterprise Application Integration (EAI)

Publish - subscribe channel

- Used for broadcasting messages to each recipient in the channel
- Ensures that each recipient gets only one copy of the message
- Helps in debugging channel without disturbing traffic
- JMS Implementation uses 'Topic' interface to implement Point to point channel
  - TopicPublisher – send messages
  - TopicSubscriber – receive messages

Enterprise Application Integration (EAI)

Publish - subscribe channel – Example (JMS)

- Sending message

```
Topic topic = // obtain the topic via JNDI
TopicConnectionFactory factory = // obtain the connection factory via JNDI
TopicConnection connection = factory.createTopicConnection();
TopicSession session = connection.createTopicSession(true, Session.AUTO_ACKNOWLEDGE);
TopicPublisher publisher = session.createPublisher(topic);

Message message = session.createTextMessage("The contents of the message.");
publisher.publish(message);
```

Enterprise Application Integration (EAI)

Publish - subscribe channel – Example (JMS)

- Receiving message

```
Topic topic = // obtain the topic via JNDI
TopicConnectionFactory factory = // obtain the connection factory via JNDI
TopicConnection connection = factory.createTopicConnection();
TopicSession session = connection.createTopicSession(true, Session.AUTO_ACKNOWLEDGE);
TopicSubscriber subscriber = session.createSubscriber(topic);

TextMessage message = (TextMessage) subscriber.receive();
String contents = message.getText();
```

Thank You!

In our next session:

Middleware Security

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Integration Styles

Srikanth Gunturu

Guest Faculty
BITS, WILP

In this segment

Integration Styles

- Introduction - Application Integration Criteria
- Application Integration Options
  - File Transfer
  - Shared Database
  - Remote Procedure Invocation
  - Messaging

Integration Styles

Introduction – Application Integration Criteria

- Enterprise integration is the task of making (possibly heterogenous) applications work together to produce a unified set of functionality.
- Integration Criteria
  - Application integration –need to reuse, interact and integrate other applications
  - Application coupling – Minimum dependency /
  - Integration simplicity – Minimize customization needed to integrate applications
  - Integration technology – Reduce the need for specialized hardware/software for integration
  - Data format – Agreement on data format / translation
  - Data timeliness – Reduce staleness of data due to integration nuances
  - Data or functionality – Ability to share both data and functionality
  - Asynchronicity – Ability to proceed without being blocked for complex batch transactions

Integration Styles

Application Integration Options

- No one fit for all methodology – option chosen depends on context
- Major integration options:
  - File Transfer – applications communicate via files
  - Shared Database – applications share common data using shared database
  - Remote Procedure Invocation – Applications expose procedures to be invoked by integrating applications
  - Messaging – applications connect/communicate via common messaging system

Integration Styles

File Transfer

- Each application produces files with data that it needs to share with others
- Integrators (middleware) handle transforming files into various formats and access aspects
  - Legacy file formats (Mainframe)
  - Text format (Unix variants)
  - CSV format (Windows based systems)
  - XML/JSON formats (Web based systems)
- Files are produced/consumed at a frequency demanded by business
  - Hourly, Nightly, Weekly etc.
  - Ex: Batch processing applications like medical insurance claims (nightly)
- Less work on integration side, more work for application developers
- Staleness of data, in the generation of real time processing

Integration Styles

Shared Database

- Integrate applications by having them store their data in a single Shared Database
- Consistency in data format and ease of access
- Challenges
  - Deciding on a unified schema that fits all applications' needs
  - External packages that do not conform to enterprise schema
  - Performance bottlenecks on database
    - Distributed database locking / deadlock issues

Integration Styles

Remote Procedure Invocation

- Applies the principle of encapsulation to integrating applications
- Applications provide interface for the methods/data they want to expose to other applications to integrate with
- Each application maintains its own encapsulation/integrity of data
- Promotes certain degree of (tight) coupling between applications
- Examples: CORBA, COM, .NET Remoting etc.

Integration Styles

Messaging

- Transfers packets of data frequently, immediately, reliably, and asynchronously, using customizable formats
- Supports asynchronous communication seamlessly
- Allows loose coupling of applications
- Supports behavioral collaboration of applications with event based communication

Thank You!

In our next session:

Messaging Systems

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Message Channels

Madhu Venkat

Guest Faculty
BITS, WILP

In this segment

Message Channels

- Message Channel
- Point to Point Channel
- Publish-Subscr Channel
- Data Type Channel
- Invalid Message Channel
- Dead Letter Channel
- Guaranteed Delivery
- Channel Adapter
- Messaging Bridge
- Message Bus

Message Channel

- A Message channel is a basic architectural pattern of a messaging system and is used fundamentally for exchanging data between applications. An application can use a channel to make a specific type of data available to any receiver applications that need to consume that type of data.

Message Channel

Point to Point Channel

- The Point-to-Point Channel EIP ensures that only a single receiver consumes a message (when there are multiple receivers waiting to consume the message).

Message Channel

Point to Point Channel – Example

Message Channel

Point to Point Channel

Point-to-Point EIP (Figure 1)	Point-to-Point Channel Example Scenario (Figure 2)
Sender	Stock Quote Client
Receiver	Three instances of the Stock Quote service
Point to Point Channel	Load-balance Endpoint
Order	Stock Quote Requests

Message Channel

Publish-Subscribe Channel

- The Publish-Subscribe Channel EIP receives messages from the input channel, and then splits and transmits them among its subscribers through the output channel. Each subscriber has only one output channel.

Message Channel

Publish-Subscribe Channel - Example

- In this example, both *Subscriber Y* and *Subscriber Z* each receive a copy of *Message 3* as they are subscribing to the same *Topic C*, but through separate subscriptions.

Message Channel

Datatype Channel

- This EIP creates a separate channel for each type of data so that all the messages on a given channel will contain the same data type. The sender, who knows the data type, should select the appropriate channel on which to send the message. The receiver identifies the type of data a message contains, based on the channel in which it is received.

This example scenario depicts a Stock Quote service deployed in Axis2 server. It offers several service operations to the user. The ESB uses the filter mediator to identify each actio

Message Channel

Datatype Channel - Example

- This EIP creates a separate channel for each type of data so that all the messages on a given channel will contain the same data type. The sender, who knows the data type, should select the appropriate channel on which to send the message. The receiver identifies the type of data a message contains, based on the channel in which it is received.

Message Channel

Invalid Message Channel

- The Invalid Message Channel EIP pattern allows administrators to define an error indication that appears when an endpoint fails to process a request.

Message Channel

Invalid Message Channel - Example

- The example scenario creates a deliberate error situation to demonstrate how the ESB handles errors on message failures.

Message Channel

Dead Letter Channel

- The Dead Letter Channel (DLC) EIP outlines how messaging systems can handle messages that cannot be delivered to the recipient. Due to system/network failures or failures at the recipient's end, messages sometimes do not get delivered to the target. In such cases, the messaging system can deliver the message to a DLC.

Message Channel

Dead Letter Channel - Example

- This example takes a proxy service called StockQuoteProxy, which fronts a service by the name SimpleStockQuoteService. As long as the SimpleStockQuoteService is running, the clients calling StockQuoteProxy service got response. But if the SimpleStockQuoteService is down or a failure occurs while trying to send message to the SimpleStockQuoteService, the faultsequence will get involved and the message will be delivered to the dead letter channel.

Message Channel

Guaranteed Delivery Channel

- The Guaranteed Delivery EIP ensures safe delivery of a message by storing it locally and transmitting it to the receiver's data store. Even when the receiver is offline, the EIP ensures that the message goes through when the receiver comes online.

Message Channel

Guaranteed Delivery Channel - Example

- This example is a stock quote service where a stock quote request is sent to a specific endpoint when the receiver is offline. An Axis2 server acts as the receiver. The ESB stores the request message in a JMS message store provided by the ESB. In this scenario, a Message Broker acts as the JMS message store. Here, the Message Broker can either be the WSO2 Message Broker or ActiveMQ.

Message Channel

Channel Adapter Channel

- Channel Adapter accesses an application's API or data and publishes messages on a channel based on this data. Also, it receives messages and invokes functionality inside the application.

Message Channel

Channel Adapter Channel - Example

- This example demonstrates WSO2 ESB's Salesforce connector transferring a message coming from a stock quote client to Salesforce API and then sends the queried response that comes from Salesforce back to the client.

Message Channel

Messaging Bridge Channel

- The Messaging Bridge EIP facilitates the connection between messaging systems and replicates messages between them by transforming message formats from one system to the other.

Message Channel

Messaging Bridge Channel - Example

- This example illustrates the ESB as a message bridge, which accepts a REST message from the client, transforms the REST message to SOAP format, and sends the SOAP message to the back-end Axis2 server.

Message Channel

Messaging Bus Channel

- The Message Bus EIP enables separate applications to work together in a decoupled manner so that applications can be easily added or removed without affecting each other. This approach makes maintenance and testing smoother, since editing or removing an application will not affect the functionality of any other application.

Thank You!

In our next session:

Message Routing

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Message Construction

Srikanth Gunturu

Guest Faculty
BITS, WILP

In this segment

Message Construction

- Command Message
- Document Message
- Event Message
- Request-Reply
- Return Address
- Correlation Identifier
- Message Sequence
- Message Expiration
- Format Indicator

Message Construction

Command Message

- Command Message is used to invoke a procedure in another application, reliably.
- Asynchronous invocation of procedures, as opposed to RPC

- Message contains command for the other application, there's no other specific indication on message
- Usually sent on point-to-Point channel
- Ex: SOAP messages

Message Construction

Document Message

- Document Message is used to reliably transfer a data structure between applications

- Can contain any kind of message in the system, except that it represents a data object
- Usually sent on a point-to-point channel
- Ex: XML documents sent across the network

Message Construction

Event Message

- Document Message is used to reliably send event notifications between application
- Sender will create an event object and sends it in a message (no special indicator used)
- Push model – sends event and the related state change info
- Pull model – sends only the event details and optionally additional info to obtain state details
  - Update
  - State Request
  - State Reply

Message Construction

Request-Reply

- Facilitates two way communication between sender/receiver using two-way channel

- Requestor – sends request message and waits for reply
- Replier – receives request message and sends the reply
- Request channel could be point-to-point or publish-subscribe
- Reply channel is almost always point-to-point
- Reply can be:
  - Synchronous (blocking)
  - Asynchronous (callback)

Message Construction

Return Address

- Asynchronous request-reply model has callback mechanism, which needs the receiver to know where to send the response, especially with multiple senders in question
- Return Address is supplied along with the Request Message, that will be used by receiver to send the reply back (Ex: JMS – JMSReplyTo field on request object)

Message Construction

Correlation Identifier

- Each reply in request-reply model should contain a Correlation Identifier that uniquely indicates a which request this reply is for.
- Correlation Identifier has 6 parties to it:
  - Requestor – Sender App
  - Replier – Replier App
  - Request – Contains Request ID
  - Reply – Contains Correlation ID
  - **Request ID – A unique token from the request**
  - **Correlation ID – Unique token in response copied from Request ID**
- Correlation ID (and even Request ID) are specified in respective message headers
- Ex: JMS - JMSCorrelationID and JMSMessageID

Message Construction

Message Sequence

- Message Sequence is used whenever a large set of data may need to be broken into message-size chunks
- Each message in Message Sequence is marked with sequence identification fields
- Message Sequence identification fields
  - Sequence identifier
  - Position identifier
  - Size or End indicator
- Example Use cases:
  - Large document transfer
  - Multi item query
  - Distributed query

Message Construction

Message Expiration

- Message Expiration sets the time limit after which the message becomes stale
- One the time limit expires, the message is not considered by consumers for processing, moved to Invalid Message Channel
- It could be configured to be rerouted to Dead Letter Channel for further handing
- Message Expiration is a timestamp
  - Relative (longevity), using sent time property
  - Absolute
- JMS implementation - MessageProducer.setTimeToLive(long)

Message Construction

Format Indicator

- The format indicator enables the sender to tell the receiver the format of the message
- This handles versioning and future changes
- Format indicator can be implemented as:
  - Version Number
  - Foreign Key
  - Format Document

```
<?xml version="1.0"?>
```

```
<!DOCTYPE greeting SYSTEM "hello.dtd">
```

```
<!DOCTYPE greeting [
  <!ELEMENT greeting (#PCDATA)>
]>
```

Thank You!

In our next session:

Message Routing

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Message Routing

Madhu Venkat

Guest Faculty
BITS, WILP

Message Routing

- Content-Based Router
- Message Filter
- Dynamic Router
- Recipient List
- Splitter
- Aggregator
- Resequencer
- Composed Msg-Processor
- Scatter-Gather
- Routing Slip

Message Routing

Content-Based Router

- The *Content-Based Router* examines the message content and routes the message onto a different channel based on data contained in the message. The routing can be based on a number of criteria such as existence of fields, specific field values etc.

Message Routing

Message Filter

- The *Message Filter* has only a single output channel. If the message content matches the criteria specified by the *Message Filter*, the message is routed to the output channel. If the message content does not match the criteria, the message is discarded.

Message Routing

Dynamic Router

- Use a *Dynamic Router*, a Router that can self-configure based on special configuration messages from participating destinations.

Message Routing

Recipient List

- Define a channel for each recipient. Then use a *Recipient List* to inspect an incoming message, determine the list of desired recipients, and forward the message to all channels associated with the recipients in the list.

Message Routing

Splitter

- **Use a *Splitter* to break out the composite message into a series of individual messages, each containing data related to one item.**

Message Routing

Aggregator

- **Use a stateful filter, an *Aggregator*, to collect and store individual messages until a complete set of related messages has been received. Then, the *Aggregator* publishes a single message distilled from the individual messages.**

Message Routing

Resequencer

- **Use a stateful filter, a *Resequencer*, to collect and re-order messages so that they can be published to the output channel in a specified order.**

Message Routing

Composed Message Processor

- **Use *Composed Message Processor* to process a composite message. The *Composed Message Processor* splits the message up, routes the sub-messages to the appropriate destinations and re-aggregates the responses back into a single message.**

Message Routing

Scatter-Gather

- **Use a *Scatter-Gather* that broadcasts a message to multiple recipients and re-aggregates the responses back into a single message.**

Message Routing

Routing Slip

- **Attach a *Routing Slip* to each message, specifying the sequence of processing steps. Wrap each component with a special message router that reads the *Routing Slip* and routes the message to the next component in the list.**

Thank You!

In our next session:

Message Transformation