BITS Pilani

Pilani | Dubai | Goa | Hyderabad | Mumbai

WORK INTEGRATED LEARNING PROGRAMMES

Middleware Technologies

PPT Booklet

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Course Name : Middleware Technologies
Code: CSI ZG524
Prof.(Dr.)Prasanna Balaji Narasingapuram

BITS Pilani
Pilani Campus

innovate achieve lead

Start Recording

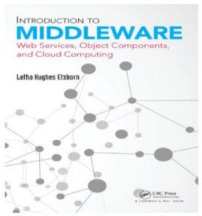
BITS Pilani, Pilani Campus

Important Information to WIMS 2021'ians

- It is important to know that just login to the session does not guarantee the attendance.
- Once you join the session, continue till the end to consider you as present in the class.
- IMPORTANTLY, you need to make the class more interactive by responding to Professors queries in the session.
- **Whenever Professor calls your number / name ,you need to respond, otherwise it will be considered as ABSENT**

BITS Pilani, Pilani Campus

Text Books



T1: Letha Hughes Etzkorn - Introduction to middleware _ web services, object components, and cloud computing-Chapman and Hall_CRC (2017).

T2: William Grosso - Java RMI (Designing & Building Distributed Applications)

R1: Gregor Hohpe, Bobby Woolf - Enterprise Integration Patterns_ Designing, Building, and Deploying Messaging Solutions -Addison-Wesley Professional (2003)

R2: MongoDB in Action

Note: In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations

BITS Pilani, Pilani Campus

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Modular Structure

No	Title of the Module
M1	Introduction and Evolution
M2	Enterprise Middleware
M3	Middleware Design and Patterns
M4	Middleware for Web-based Application and Cloud-based Applications
M5	Specialized Middleware

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

CS1: Introduction and Evolution

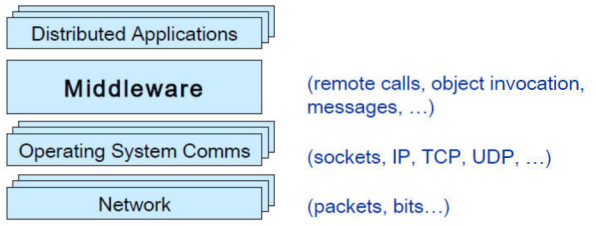
Source Courtesy: Some of the contents of this PPT are sourced from materials provided by Publishers of T1

Content

- **View of Middleware**
- **Forms of Middleware**

What is Middleware?

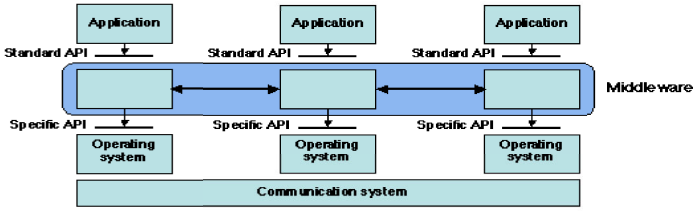
- Layer between OS and distributed applications
- Hides complexity and heterogeneity of distributed system
- Bridges gap between low-level OS communications and programming language abstractions
- Provides common programming abstraction and infrastructure for distributed applications
- Overview at: http://www.middleware.org



innovate achieve lead

"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 provides support for (some of):
  - Naming, Location, Service discovery, Replication
  - Protocol handling, Communication faults, QoS
  - Synchronization, Concurrency, Transactions, Storage
  - Access control, Authentication

Middleware dimensions:		
Request/Reply	vs.	Asynchronous Messaging
Language-specific	vs.	Language-independent
Proprietary	vs.	Standards-based
Small-scale	vs.	Large-scale
Tightly-coupled	vs.	Loosely-coupled components

Common Forms of MW

- 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

Socket

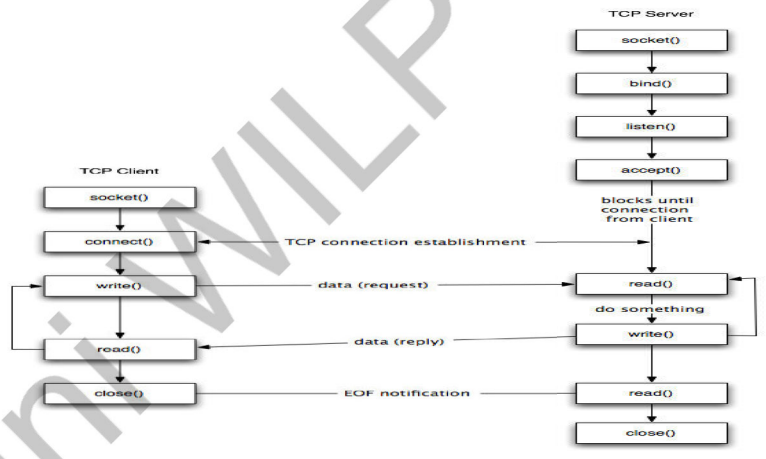
- Socket is an internal endpoint for sending/receiving data within a node on network
- Berkeley/POSIX sockets defined API for Inter Process Communication (IPC) within same host (BSD 4.2 – circa 1983)
- Early form of Middleware (limited to same host systems)
- Windows variant (WinSock) based on BSD Sockets.
- Treated similar to files in BSD/POSIX
- Maintained in File Descriptor table
- Supported protocols
  - TCP/IP – IPv4, IPv6
  - UDP

innovate achieve lead

Socket API

socket —creates a descriptor for use in network communications
connect —connect to a remote peer (client)
write —send outgoing data across a connection
read —acquire incoming data from a connection
close —terminate communication and deallocate a descriptor
bind —bind a local IP address and protocol port to a socket
listen —set the socket listening on the given address and port for connections from the client and set the number of incoming connections from a client (backlog) that will be allowed in the listen queue at any one time
accept —accept the next incoming connection (server)
recv —receive the next incoming datagram
recvmsg —receive the next incoming datagram (variation of recv)
recvfrom —receive the next incoming datagram and record its source endpoint address
send —send an outgoing datagram
sendmsg —send an outgoing datagram (variation of send)
sendto —send an outgoing datagram, usually to a prerecorded endpoint address
shutdown —terminate a TCP connection in one or both directions
getpeername —after a connection arrives, obtain the remote machine's endpoint address from a socket
getsockopt —obtain the current options for a socket
setsockopt —change the options for a socket

Socket – Life Cycle



TCP Client – Server example

- #Server

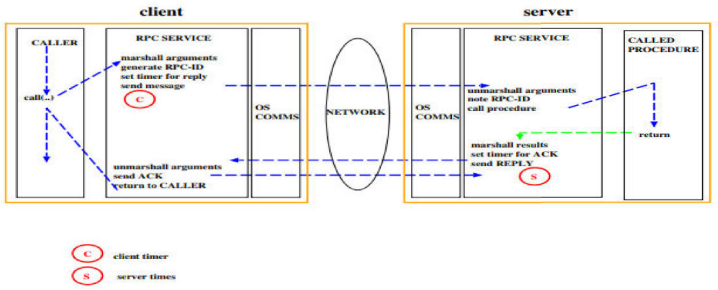
server.c

- #Client

client.c

Remote Procedure Call (RPC)

- Masks remote function calls as being local Client/server model
- Request/reply paradigm usually implemented with message passing in RPC service
- Marshalling of function parameters and return value



Properties of RPC

innovate achieve lead

Language-level pattern of **function call**
- easy to understand for programmer

Synchronous request/reply interaction
- natural from a programming language point-of-view matches replies to requests
- built in synchronisation of requests and replies

Distribution transparency (in the no-failure case)
- hides the complexity of a distributed system

Various **reliability** guarantees
- deals with some distributed systems aspects of failure

Failure Modes of RPC

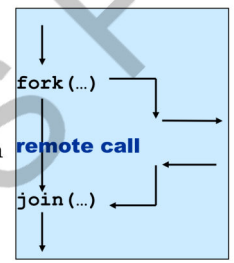
innovate achieve lead

- Invocation semantics supported by RPC in the light of: network and/or server congestion, client, network and/or server failure note DS independent failure modes
- RPC systems differ, many examples, local was Mayflower
- May be or at most once (RPC system tries once)
- Error return – programmer may retry
- Exactly once (RPC system retries a few times)
  - Hard error return – some failure most likely note that "exactly once" cannot be guaranteed

Disadvantages of RPC

innovate achieve lead

Synchronous request/reply interaction
- tight coupling between client and server
- client may block for a long time if server loaded leads to multi-threaded programming at client
- slow/failed clients may delay servers when replying multi-threading essential at servers



Distribution Transparency
- Not possible to mask all problems

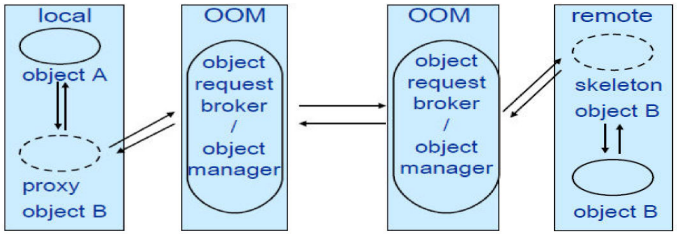
RPC paradigm is not object-oriented
- invoke functions on servers as opposed to methods on objects

Object-Oriented Middleware (OOM)

innovate achieve lead

- **Objects** can be *local* or *remote*
- **Object references** can be *local* or *remote*
- Remote objects have visible **remote interfaces**
- Masks remote objects as being local using **proxy objects**

Remote method invocation



Properties of OOM

Support for object-oriented programming model
- objects, methods, interfaces, encapsulation…
- exceptions (were also in some RPC systems e.g. Mayflower)

Synchronous request/reply interaction
–same as RPC

Location Transparency
–system (ORB) maps object references to locations

Services comprising multiple servers are easier to build with OOM
–RPC programming is in terms of ***server-interface (operation)***
–RPC system looks up server address in a location service

Java Remote Method Invocation (RMI)

```
public interface PrintService extends Remote {
int print(Vector printJob) throws RemoteException;
}
```

- Distributed objects in Java
- RMI compiler creates proxies and skeletons
- RMI registry used for interface lookup
- Entire system written in Java (single-language system)

CORBA

Common Object Request Broker Architecture
- Open standard by the OMG (Version 3.0)
- Language- and platform independent

Object Request Broker (ORB)
- General Inter-ORB Protocol (GIOP) for communication
- Interoperable Object References (IOR) contain object location
- CORBA Interface Definition Language (IDL)
  - Stubs (proxies) and skeletons created by IDL compiler
- Dynamic remote method invocation

Interface Repository
- Querying existing remote interfaces

Implementation Repository
- Activating remote objects on demand

CORBA IDL

- Definition of language-independent remote interfaces
  - Language mappings to C++, Java, Smalltalk, …
  - Translation by IDL compiler
- Type system
  - basic types: long (32 bit), long long (64 bit), short, float, char, boolean, octet, any, …
  - constructed types: struct, union, sequence, array, enum
  - objects (common super type Object)
- Parameter passing
  - in, out, inout
  - basic & constructed types passed by value
  - objects passed by reference

```
typedef sequence<string> Files;
interface PrintService : Server {
void print(in Files printJob);
};
```

CORBA Services (selection)

- Naming Service
  Names → remote object references
- Trading Service
  Attributes (properties) → remote object references
- Persistent Object Service
  Implementation of persistent CORBA objects
- Transaction Service
  Making object invocation part of transactions
- Event Service and Notification Service
  - In response to applications' need for asynchronous communication
  - built above synchronous communication with *push* or *pull* options
  - *not* an integrated programming model with general IDL messages

Disadvantages of OOM

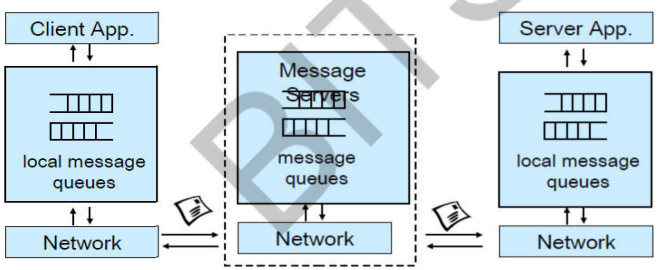
Synchronous request/reply interaction only
- So CORBA **oneway** semantics added and -
- Asynchronous Method Invocation (AMI)
- But *implementations* may not be loosely coupled

Distributed garbage collection
- Releasing memory for unused remote objects

OOM rather static and heavy-weight
- Bad for ubiquitous systems and embedded devices

Message-Oriented Middleware (MOM)

- Communication using **messages**
- Messages stored in **message queues**
- **message servers** decouple client and server
- Various assumptions about **message content**



Properties of MOM

- **Asynchronous** interaction
  - Client and server are only **loosely coupled**
  - Messages are queued
  - Good for application integration
- Support for **reliable** delivery service
  - Keep queues in persistent storage
- Processing of messages by intermediate message server(s)
  - May do filtering, transforming, logging, …
  - Networks of message servers
- Natural for database integration

innovate achieve lead

IBM MQSeries

One-to-one reliable message passing using queues
- Persistent and non-persistent messages
- Message priorities, message notification

Queue Managers
- Responsible for queues
- Transfer messages from input to output queues
- Keep routing tables

Message Channels
- Reliable connections between queue managers

Messaging API:

MQopen	Open a queue
MQclose	Close a queue
MQput	Put message into opened queue
MQget	Get message from local queue

Java Message Service (JMS)

- API **specification** to access MOM implementations
- Two modes of operation *specified*:
  - **Point-to-point**
    - one-to-one communication using queues
  - **Publish/Subscribe**
    - cf. Event-Based Middleware
- **JMS Server** implements JMS API
- JMS Clients connect to JMS servers
- Java objects can be serialised to JMS messages
- A JMS interface has been provided for MQ
- pub/sub (one-to-many) - just a specification?

Disadvantages of MOM

- Poor programming abstraction (but has evolved)
  - Rather low-level (cf. Packets)
  - Request/reply more difficult to achieve, but can be done
- Message formats originally unknown to middleware
  - No type checking (JMS addresses this – implementation?)
- Queue abstraction only gives one-to-one communication
  - Limits scalability (JMS pub/sub – implementation?)

Web Services

- Use well-known web standards for distributed computing
- **Communication**
  - Message content expressed in **XML**
  - **S**imple **O**bject **A**ccess **P**rotocol (**SOAP**)
    - Lightweight protocol for sync/async communication
- **Service Description**
  - **W**eb **S**ervices **D**escription **L**anguage (**WSDL**)
    - Interface description for web services
- **Service Discovery**
  - **U**niversal **D**escription **D**iscovery and **I**ntegration (**UDDI**)
    - Directory with web service description in WSDL

innovate achieve lead

Properties of Web Services

- Language-independent and open standard
- **SOAP** offers OOM and MOM-style communication:
  - Synchronous request/reply like OOM
  - Asynchronous messaging like MOM
  - Supports internet transports (http, smtp, ...)
  - Uses XML Schema for marshalling types to/from programming language types
- **WSDL** says how to use a web service
  http://api.google.com/Google Search.wsdl
- **UDDI** helps to find the right web service
- Exports SOAP API for access

Disadvantages of Web Services

Low-level abstraction
- leaves a lot to be implemented

Interaction patterns have to be built
- one-to-one and request-reply provided
- one-to-many?
- still synchronous service invocation, rather than notification
- No nested/grouped invocations, transactions, ...

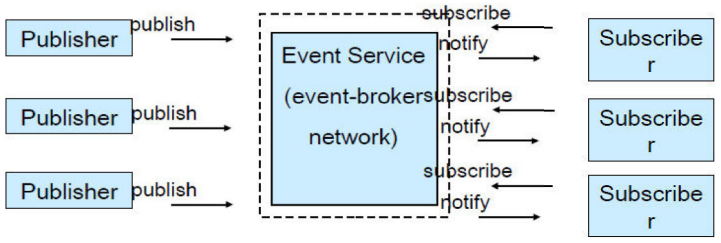
No location transparency

What we lack, so far

- General interaction patterns
  - we have one-to-one and request-reply
  - one-to-many? many to many?
  - notification?
  - dynamic joining and leaving?
- Location transparency
  - anonymity of communicating entities
- Support for pervasive computing
  - data values from sensors
  - lightweight software

Event-Based Middleware a.k.a. Publish/Subscribe

- **Publishers** (*advertise* and) *publish* **events** (messages)
- **Subscribers** express interest in events with *subscriptions*
- **Event Service** *notifies* interested subscribers of published events
- Events can have arbitrary content (typed) or name/value pairs



innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Topic-Based and Content-Based Pub/Sub

- Event Service matches events against subscriptions
  - What do subscriptions look like?
- **Topic-Based Publish/Subscribe**
  - Publishers publish events belonging to a **topic** or **subject**
  - Subscribers subscribe to a **topic**
    subscribe(PrintJobFinishedTopic, …)
- **(Topic and) Content-Based Publish/Subscribe**
  - Publishers publish events belonging to **topic**s and
  - Subscribers provide a **filter** based on ***content*** of events
    subscribe(type=printjobfinished, printer='aspen', …)

Properties of Publish/Subscribe

Asynchronous communication
- Publishers and subscribers are loosely coupled

Many-to-many interaction between pubs. and subs.
- Scalable scheme for large-scale systems
- Publishers do not need to know subscribers, and vice-versa
- Dynamic join and leave of pubs, subs,

(Topic and) Content-based pub/sub very expressive
- Filtered information delivered only to interested parties
- Efficient content-based routing through a broker network

Summary

- Middleware is an important abstraction for building distributed systems

1. Remote Procedure Call
2. Object-Oriented Middleware
3. Message-Oriented Middleware
4. Event-Based Middleware

- Synchronous vs. asynchronous communication
- Scalability, many-to-many communication
- Language integration
- Ubiquitous systems, mobile systems

Thank You

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

innovate achieve lead

innovate achieve lead

IMP Note to Students

- It is important to know that just login to the session does not guarantee the attendance.
- Once you join the session, continue till the end to consider you as present in the class.
- IMPORTANTLY, you need to make the class more interactive by responding to Professors queries in the session.
- **Whenever Professor calls your number / name ,you need to respond, otherwise it will be considered as**

innovate achieve lead

Text Books

T1: Letha Hughes Etzkorn - Introduction to middleware _ web services, object components, and cloud computing-Chapman and Hall_CRC (2017).

T2: William Grosso - Java RMI (Designing & Building Distributed Applications)

R1: Gregor Hohpe, Bobby Woolf - Enterprise Integration Patterns_ Designing, Building, and Deploying Messaging Solutions -Addison-Wesley Professional (2003)

R2: MongoDB in Action

Note: In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Modular Structure

No	Title of the Module
M1	Introduction and Evolution
M2	Enterprise Middleware
M3	Middleware Design and Patterns
M4	Middleware for Web-based Application and Cloud-based Applications
M5	Specialized Middleware

BITS Pilani
Pilani|Dubai|Goa|Hyderabad

CS2: CORBA - Common Object Request Broker Architecture

Source Courtesy: Some of the contents of this PPT are sourced from materials provided by Publishers of T1

CORBA

Common Object Request Broker Architecture
- Open standard by the OMG (Version 3.0)
- Language- and platform independent

Object Request Broker (ORB)
- General Inter-ORB Protocol (GIOP) for communication
- Interoperable Object References (IOR) contain object location
- CORBA Interface Definition Language (IDL)
  - Stubs (proxies) and skeletons created by IDL compiler
- Dynamic remote method invocation

Interface Repository
- Querying existing remote interfaces

Implementation Repository
- Activating remote objects on demand

CORBA IDL

- Definition of language-independent remote interfaces
  - Language mappings to C++, Java, Smalltalk, ...
  - Translation by IDL compiler
- Type system
  - basic types: long (32 bit), long long (64 bit), short, float, char, boolean, octet, any, ...
  - constructed types: struct, union, sequence, array, enum
  - objects (common super type Object)
- Parameter passing
  - in, out, inout
  - basic & constructed types passed by value
  - objects passed by reference

```
typedef sequence<string> Files;
interface PrintService : Server {
void print(in Files printJob);
};
```


innovate achieve lead

CORBA Services (selection)

- Naming Service
  Names → remote object references
- Trading Service
  Attributes (properties) → remote object references
- Persistent Object Service
  Implementation of persistent CORBA objects
- Transaction Service
  Making object invocation part of transactions
- Event Service and Notification Service
  - In response to applications' need for asynchronous communication
  - built above synchronous communication with *push* or *pull* options
  - *not* an integrated programming model with general IDL messages

CORBA

- The Common Object Request Broker Architecture (CORBA) is a standard architecture for a distributed objects system.
- CORBA is designed to allow distributed objects to interoperate in a heterogeneous environment, where objects can be implemented in different programming language and/or deployed on different platforms

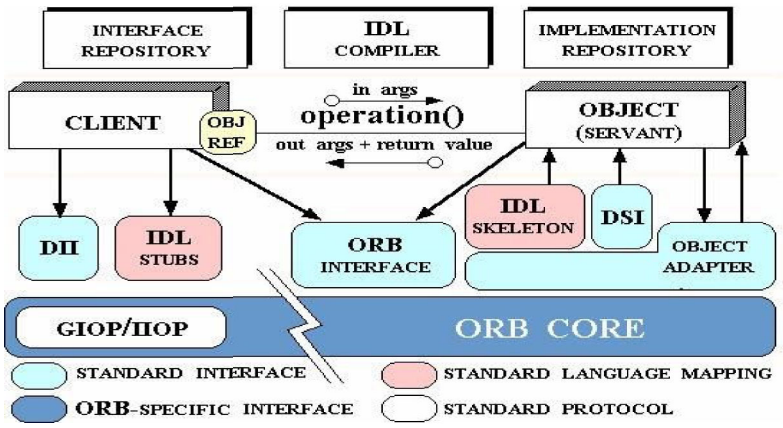
CORBA vs. Java RMI

- CORBA differs from the architecture of Java RMI in one significant aspect:
  - RMI is a proprietary facility developed by Sun MicroSystems, Inc., and supports objects written in the Java programming language only.
  - CORBA is an architecture that was developed by the Object Management Group (OMG), an industrial consortium.

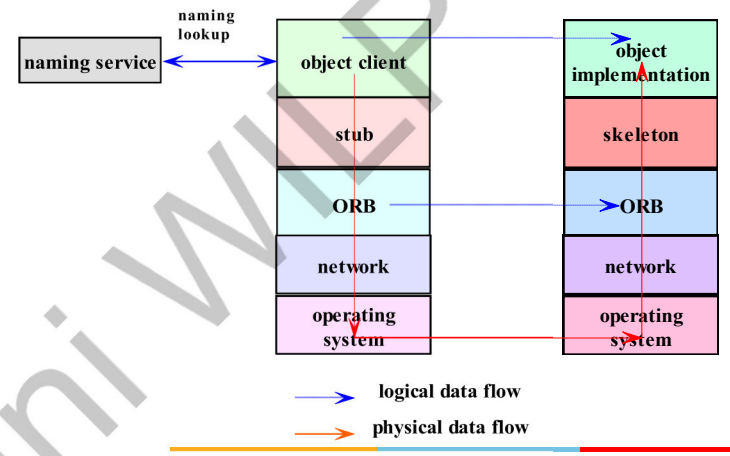
CORBA

- CORBA is not inself a distributed objects facility; instead, it is a set of protocols.
- A distributed object facility which adhere to these protocols is said to be CORBA-compliant, and the distributed objects that the facility support can interoperate with objects supported by other CORBA-compliant facilities.
- CORBA is a very rich set of protocols. We will instead focus on the key concepts of CORBA related to the distributed objects paradigm. We will also study a facility based on CORBA: the Java IDL.

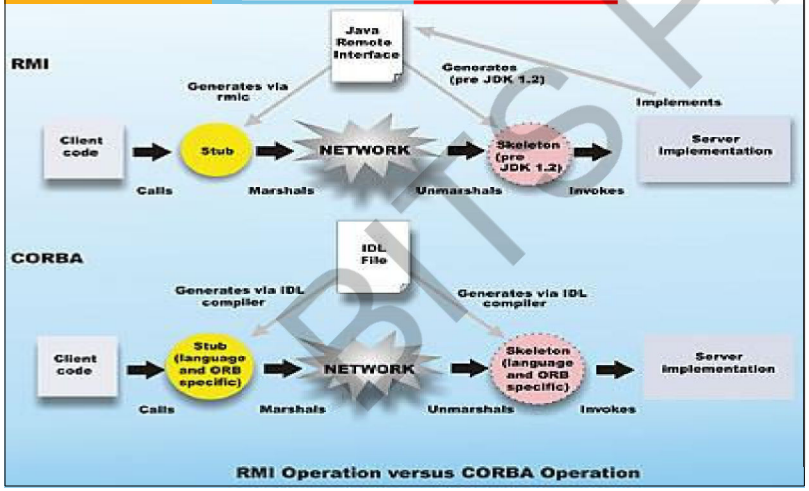
Architecture



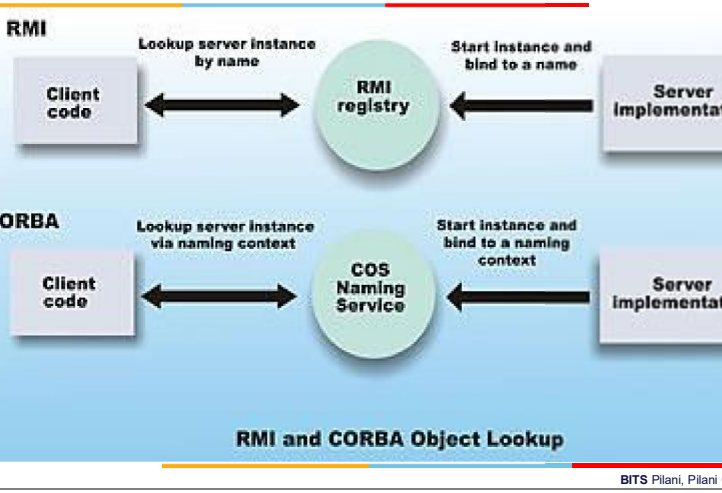
Basic Workflow



CORBA vs RMI



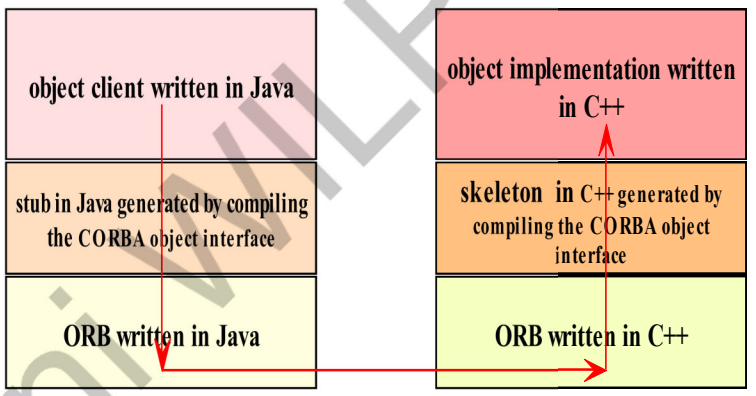
CORBA vs RMI



CORBA Object Interface

- A distributed object is defined using a software file similar to the remote interface file in Java RMI.
- Since CORBA is language independent, the interface is defined using a universal language with a distinct syntax, known as the ***CORBA Interface Definition Language*** (***IDL***).
- The syntax of CORBA IDL is similar to Java and C++. However, object defined in a CORBA IDL file can be implemented in a large number of diverse programming languages, including C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python, and IDLScript.
- For each of these languages, OMG has a standardized mapping from CORBA IDL to the programming language, so that a compiler can be used to process a CORBA interface to generate the proxy files needed to interface with an object implementation or an object client written in any of the CORBA-compatible languages.

Cross-language CORBA application



Inter-ORB Protocols

- To allow ORBs to be interoperable, the OMG specified a protocol known as the ***General Inter-ORB Protocol*** (***GIOP***), a specification which "provides a general framework for protocols to be built on top of specific transport layers."
- A special case of the protocol is the ***Inter-ORB Protocol*** (***IIOP***), which is the GIOP applied to the TCP/IP transport layer.

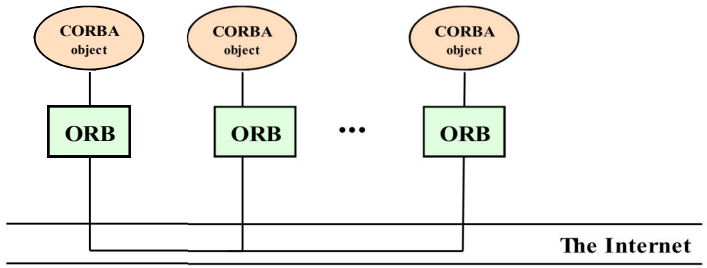
Inter-ORB Protocols

The IIOP specification includes the following elements:

1. **Transport management requirements**: specifies the connection and disconnection requirements, and the roles for the object client and object server in making and unmaking connections.
2. **Definition of common data representation**: a coding scheme for marshalling and unmarshalling data of each IDL data type.
3. **Message formats**: different types of message format are defined. The messages allow clients to send requests to object servers and receive replies. A client uses a Request message to invoke a method declared in a CORBA interface for an object and receives a reply message from the server.

Object Bus

An ORB which adheres to the specifications of the IIOP may interoperate with any other IIOP-compliant ORBs over the Internet. This gives rise to the term "***object bus***", where the Internet is seen as a bus that interconnects CORBA objects



ORB products

There are a large number of proprietary as well as experimental ORBs available:
(See CORBA Product Profiles, http://www.puder.org/corba/matrix/)

- Orbix IONA
- Borland Visibroker
- PrismTech's OpenFusion
- Web Logic Enterprise from BEA
- Ada Broker from ENST
- Free ORBs

Object Servers and Object Clients

- As in Java RMI, a CORBA distributed object is exported by an ***object server***, similar to the object server in RMI.
- An ***object client*** retrieves a reference to a distributed object from a naming or directory service, to be described, and invokes the methods of the distributed object.

CORBA Object References

- As in Java RMI, a CORBA distributed object is located using an ***object reference***. Since CORBA is language-independent, a CORBA object reference is an abstract entity mapped to a language-specific object reference by an ORB, in a representation chosen by the developer of the ORB.
- For interoperability, OMG specifies a protocol for the abstract CORBA object reference object, known as the ***Interoperable Object Reference*** (***IOR***) protocol.

innovate achieve lead

Interoperable Object Reference (IOR)

- For interoperability, OMG specifies a protocol for the abstract CORBA object reference object, known as the ***Interoperable Object Reference*** (***IOR***) protocol.
- An ORB compatible with the IOR protocol will allow an object reference to be registered with and retrieved from any IOR-compliant directory service. CORBA object references represented in this protocol are called ***Interoperable Object Reference***s (***IOR***s).

Interoperable Object Reference (IOR)

An IOR is a string that contains encoding for the following information:

- The type of the object.
- The host where the object can be found.
- The port number of the server for that object.
- An object key, a string of bytes identifying the object.

The object key is used by an object server to locate the object.

Interoperable Object Reference (IOR)

The following is an example of the string representation of an IOR [5]:

IOR:000000000000000d49444c3a677269643a312e3000000
00000000001000000000000004c000100000000015756c4
72612e6475626c696e2e696f6e612e6965000009630000002
83a5c756c7472612e6475626c696e2e696f6e612e69653a67
7269643a303a3a49523a6726964003a

The representation consists of the character prefix "IOR:" followed by a series of hexadecimal numeric characters, each character representing 4 bits of binary data in the IOR.

CORBA Naming Service

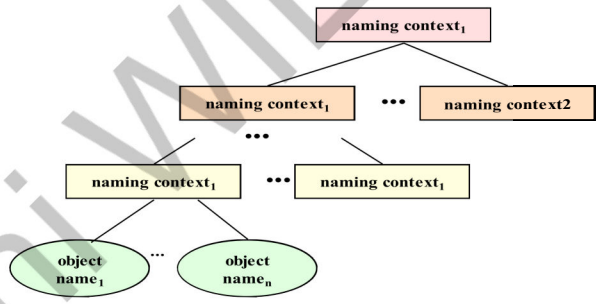
- CORBA specifies a generic directory service. The ***Naming Service*** serves as a directory for CORBA objects, and, as such, is platform independent and programming language independent.
- The Naming Service permits ORB-based clients to obtain references to objects they wish to use. It allows names to be associated with object references. Clients may query a naming service using a predetermined name to obtain the associated object reference.

CORBA Naming Service

- To export a distributed object, a CORBA object server contacts a Naming Service to ***bind*** a symbolic name to the object The Naming Service maintains a database of names and the objects associated with them.
- To obtain a reference to the object, an object client requests the Naming Service to look up the object associated with the name (This is known as ***resolving*** the object name.)
- The API for the Naming Service is specified in interfaces defined in IDL, and includes methods that allow servers to bind names to objects and clients to resolve those names.

CORBA Naming Service

To be as general as possible, the CORBA object naming scheme is necessary complex. Since the name space is universal, a standard naming hierarchy is defined in a manner similar to the naming hierarchy in a file directory



A Naming Context

- A naming context correspond to a folder or directory in a file hierarchy, while object names corresponds to a file.
- The full name of an object, including all the associated naming contexts, is known as a *compound name*. The first component of a compound name gives the name of a naming context, in which the second component is accessed. This process continues until the last component of the compound name has been reached.
- Naming contexts and name bindings are created using methods provided in the Naming Service interface.

A CORBA object name

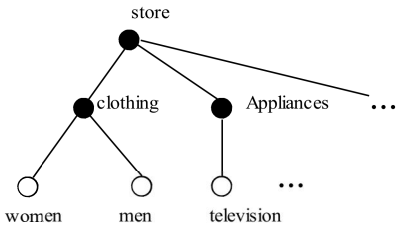
The syntax for an object name is as follows:

<naming context > ...<naming context><object name>

where the sequence of naming contexts leads to the object name.

Example of a naming hierarchy

As shown, an object representing the men's clothing department is named store.clothing.men, where store and clothing are naming contexts, and men is an object name.



Interoperable Naming Service

The ***Interoperable Naming Service*** (***INS***) is a URL-based naming system based on the CORBA Naming Service, it allows applications to share a common initial naming context and provide a URL to access a CORBA object.

CORBA Object Services

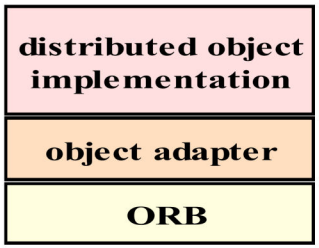
CORBA specify services commonly needed in distributed applications, some of which are:

- ***Naming Service***:
- ***Concurrency Service***:
- ***Event Service***: for event synchronization;
- ***Logging Service***: for event logging;
- ***Scheduling Service***: for event scheduling;
- ***Security Service***: for security management;
- ***Trading Service***: for locating a service by the type (instead of by name);
- ***Time Service***: a service for time-related events;
- ***Notification Service***: for events notification;
- ***Object Transaction Service***: for transactional processing.

Each service is defined in a standard IDL that can be implemented by a developer of the service object, and whose methods can be invoked by a CORBA client.

Object Adapters

In the basic architecture of CORBA, the implementation of a distributed object interfaces with the skeleton to interact with the stub on the object client side. As the architecture evolved, a software component in addition to the skeleton was needed on the server side: an **object adapter**.



Object Adapter

- An object adapter simplifies the responsibilities of an ORB by assisting an ORB in delivering a client request to an object implementation.
- When an ORB receives a client's request, it locates the object adapter associated with the object and forwards the request to the adapter.
- The adapter interacts with the object implementation's skeleton, which performs data marshalling and invoke the appropriate method in the object.

The ***Portable Object Adapter***

- There are different types of CORBA object adapters.
- The ***Portable Object Adapter***, or ***POA***, is a particular type of object adapter that is defined by the CORBA specification. An object adapter that is a POA allows an object implementation to function with different ORBs, hence the word portable.

Java IDL – Java's CORBA Facility

- IDL is part of the Java 2 Platform, Standard Edition (J2SE).
- The Java IDL facility includes a CORBA Object Request Broker (ORB), an IDL-to-Java compiler, and a subset of CORBA standard services.
- In addition to the Java IDL, Java provides a number of CORBA-compliant facilities, including ***RMI over IIOP***, which allows a CORBA application to be written using the RMI syntax and semantics.

Key Java IDL Packages

- package org.omg.CORBA – contains interfaces and classes which provides the mapping of the OMG CORBA APIs to the Java programming language
- package org.omg.CosNaming - contains interfaces and classes which provides the naming service for Java IDL
- org.omg.CORBA.ORB - contains interfaces and classes which provides APIs for the Object Request Broker.

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Java IDL Tools

Java IDL provides a set of tools needed for developing a CORBA application:

- idlj - the IDL-to-Java compiler (called idl2java in Java 1.2 and before)
- orbd - a server process which provides Naming Service and other services
- servertool – provides a command-line interface for application programmers to register/unregister an object, and startup/shutdown a server.
- tnameserv – an olderTransient Java IDL Naming Service whose use is now discouraged.

A Java IDL application example

The CORBA Interface file **Hello.idl**

```
01. module HelloApp
02. {
03.   interface Hello
04.   {
05.   string sayHello();
06.   oneway void shutdown();
07.   };
08. };
```


Compiling the IDL file (using Java)

The IDL file should be placed in a directory dedicated to the application. The file is compiled using the compiler ***idlj*** using a command as follows:

idlj -fall Hello.idl

The ***–fall*** command option is necessary for the compiler to generate all the files needed.

In general, the files can be found in a subdirectory named <some name>App when an interface file named <some name>.idl is compiled.

If the compilation is successful, the following files can be found in a ***HelloApp*** subdirectory:

HelloOperations.java	***Hello.java***
HelloHelper.java	***HelloHolder.java***
_HelloStub.java	***HelloPOA.java***

These files require no modifications.

innovate achieve lead

The *Operations.java file

- There is a file HelloOperations.java found in HelloApp/ after you compiled using idlj
- It is known as a ***Java* operations *interface*** in general
- It is a Java interface file that is equivalent to the CORBA IDL interface file (***Hello.idl***)
- You should look at this file to make sure that the method signatures correspond to what you expect.

HelloApp/HelloOperations.java

The file contains the methods specified in the original IDL file: in this case the methods *sayHello( )* and *shutdown*().

```
package HelloApp;
01. package HelloApp;
04. /**
05. * HelloApp/HelloOperations.java
06. * Generated by the IDL-to-Java compiler (portable),
07. *  version "3.1" from Hello.idl
08. */
09.
10. public interface HelloOperations
11. {
12.   String sayHello ();
13.   void shutdown ();
14. } // interface HelloOperations
```


HelloApp/Hello.java

The signature interface file combines the characteristics of the Java *operations* interface (*HelloOperations.java*) with the characteristics of the CORBA classes that it extends.

```
01. package HelloApp;
03. /**
04. * HelloApp/Hello.java
05. * Generated by the IDL-to-Java compiler (portable),
06. * version "3.1" from Hello.idl
07. */
09. public interface Hello extends HelloOperations,
10.   org.omg.CORBA.Object,
11.   org.omg.CORBA.portable.IDLEntity
12. { ...
13. } // interface Hello
```


HelloHelper.java, the Helper class

- The Java class HelloHelper provides auxiliary functionality needed to support a CORBA object in the context of the Java language.
- In particular, a method, ***narrow***, allows a CORBA object reference to be cast to its corresponding type in Java, so that a CORBA object may be operated on using syntax for Java object.

HelloHolder.java, the Holder class

- The Java class called HelloHolder holds (contains) a reference to an object that implements the Hello interface.
- The class is used to handle an out or an inout parameter in IDL in Java syntax
  (In IDL, a parameter may be declared to be ***out*** if it is an output argument, and ***inout*** if the parameter contains an input value as well as carries an output value.)

_HelloStub.java

- The Java class ***HelloStub*** is the stub file, the client-side proxy, which interfaces with the client object.
- It extends org.omg.CORBA.portable.ObjectImpl and implements the ***Hello.java*** interface.

HelloPOA.java, the server skeleton

- The Java class ***HelloImplPOA*** is the skeleton, the server-side proxy, combined with the portable object adapter.
- It extends ***org.omg.PortableServer.Servant***, and implements the ***InvokeHandler*** interface and the ***HelloOperations*** interface.

The application

Server-side Classes

- On the server side, two classes need to be provided: the servant and the server.
- The servant, ***HelloImpl***, is the implementation of the ***Hello*** IDL interface; each ***Hello*** object is an instantiation of this class.

The Servant - HelloApp/HelloImpl.java

```
// The servant -- object implementation -- for the Hello
// example.  Note that this is a subclass of HelloPOA,
// whose source file is generated from the
// compilation of Hello.idl using j2idl.
import HelloApp.*;
import org.omg.CosNaming.*;
import java.util.Properties; ...
class HelloImpl extends HelloPOA {
     private ORB orb;
     public void setORB(ORB orb_val) {
        orb = orb_val;
      }
// implement sayHello() method
            public String sayHello() {
            return "\nHello world !!\n";
            }

// implement shutdown() method
            public void shutdown() {
            orb.shutdown(false);
            }
} //end class
```


The server - HelloApp/HelloServer.java

```
public class HelloServer {
  public static void main(String args[]) {
   try{
   // create and initialize the ORB
   ORB orb = ORB.init(args, null);
   // get reference to rootpoa & activate the POAManager
   POA rootpoa =
     (POA)orb.resolve_initial_references("RootPOA");
   rootpoa.the_POAManager().activate();
   // create servant and register it with the ORB
   HelloImpl helloImpl = new HelloImpl();
   helloImpl.setORB(orb);
   // get object reference from the servant
   org.omg.CORBA.Object ref =
       rootpoa.servant_to_reference(helloImpl);
   // and cast the reference to a CORBA reference
   Hello href = HelloHelper.narrow(ref);
```


HelloApp/HelloServer.java - continued

```
// get the root naming context
// NameService invokes the transient name service
org.omg.CORBA.Object objRef =
  orb.resolve_initial_references("NameService");
// Use NamingContextExt, which is part of the
// Interoperable Naming Service (INS) specification.
NamingContextExt ncRef =
NamingContextExtHelper.narrow(objRef);
// bind the Object Reference in Naming
String name = "Hello";
NameComponent path[] = ncRef.to_name( name );
ncRef.rebind(path, href);
System.out.println
("HelloServer ready and waiting ...");
// wait for invocations from clients
  orb.run();
```


The object client application

- A client program can be a Java application, an applet, or a servlet.
- The client code is responsible for creating and initializing the ORB, looking up the object using the Interoperable Naming Service, invoking the narrow method of the ***Helper*** object to cast the object reference to a reference to a ***Hello*** object implementation, and invoking remote methods using the reference. The object's ***sayHello*** method is invoked to receive a string, and the object's shutdown method is invoked to deactivate the service.

innovate achieve lead

```
// A sample object client application.
import HelloApp.*;
import org.omg.CosNaming.*; …
public class HelloClient{
  static Hello helloImpl;
  public static void main(String args[]){
    try{
     ORB orb = ORB.init(args, null);
     org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");
   NamingContextExt ncRef =
     NamingContextExtHelper.narrow(objRef);
   helloImpl =
   HelloHelper.narrow(ncRef.resolve_str("Hello"));
   System.out.println(helloImpl.sayHello());
   helloImpl.shutdown();
```


Compiling and Running a Java IDL application

1. Create and compile the Hello.idl file on the server machine:

   idlj -fall Hello.idl
2. Copy the directory containing Hello.idl (including the subdirectory generated by ***idlj***) to the client machine.
3. In the ***HelloApp*** directory on the client machine: create ***HelloClient.java***. Compile the *.java files, including the stubs and skeletons (which are in the directory ***HelloApp***):

   javac *.java HelloApp/*.java

Compiling and Running a Java IDL application

4. In the ***HelloApp*** directory on the server machine:
   - Create ***HelloServer.java***. Compile the .java files:

     javac *.java HelloApp/*.java
   - On the server machine: Start the Java Object Request Broker Daemon, ***orbd***, which includes a Naming Service.

   To do this on Unix:

   orbd -ORBInitialPort 1050 -ORBInitialHost *servermachinename*&

   To do this on Windows:

   start orbd -ORBInitialPort 1050 -ORBInitialHost *servermachinename*

Compiling and Running a Java IDL application

5. On the server machine, start the Hello server, as follows:

   java HelloServer –ORBInitialHost <nameserver host name> -ORBInitialPort 1050
6. On the client machine, run the ***Hello*** application client. From a DOS prompt or shell, type:

   java HelloClient -ORBInitialHost *nameserverhost*

   -ORBInitialPort 1050

   all on one line.

   Note that *nameserverhost* is the host on which the IDL name server is running. In this case, it is the server machine.

Compiling and Running a Java IDL application

7. Kill or stop ***orbd*** when finished. The name server will continue to wait for invocations until it is explicitly stopped.
8. Stop the object server.

Summary-1

- You have been introduced to
  - the **Common Object Request Broker Architecture** (**CORBA**), and
  - a specific CORBA facility based on the architecture: **Java IDL**

Summary-2

- The key topics introduced with CORBA are:
  - The basic CORBA architecture and its emphasis on object interoperability and platform independence
  - Object Request Broker (ORB) and its functionalities
  - The Inter-ORB Protocol (IIOP) and its significance
  - CORBA object reference and the Interoperable Object Reference (IOR) protocol
  - **CORBA Naming Service** and the **Interoperable Naming Service** (**INS**)
  - Standard CORBA **object services** and how they are provided.
  - **Object adapters**, portable **object Adapters** (**POA**) and their significance.

Summary-3

- The key topics introduced with **Java IDL** are:
  - It is part of the Java ™ 2 Platform, Standard Edition (J2SE)
  - Java packages are provided which contain interfaces and classes for CORBA support
  - Tools provided for developing a CORBA application include ***idlj*** (the IDL compiler) and ***orbd*** (the ORB and name server)
  - An example application Hello
  - Steps for compiling and running an application.
  - Client callback is achievable.
- CORBA tookits and Java RMI are comparable and alternative technologies that provide distributed objects. An applicaton may be implemented using either technology. However, there are tradeoffs between the two.

Thank You

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

innovate achieve lead

Start Recording

Important Information to WIMS 2021'ians

- It is important to know that just login to the session does not guarantee the attendance.
- Once you join the session, continue till the end to consider you as present in the class.
- IMPORTANTLY, you need to make the class more interactive by responding to Professors queries in the session.
- **Whenever Professor calls your number / name ,you need to respond, otherwise it will be considered as**

Text Books

T1: Letha Hughes Etzkorn - Introduction to middleware _ web services, object components, and cloud computing-Chapman and Hall_CRC (2017).

T2: William Grosso - Java RMI (Designing & Building Distributed Applications)

R1: Gregor Hohpe, Bobby Woolf - Enterprise Integration Patterns_ Designing, Building, and Deploying Messaging Solutions -Addison-Wesley Professional (2003)

R2: MongoDB in Action

Note: In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Modular Structure

No	Title of the Module
M1	Introduction and Evolution
M2	Enterprise Middleware
M3	Middleware Design and Patterns
M4	Middleware for Web-based Application and Cloud-based Applications
M5	Specialized Middleware

BITS Pilani
Pilani|Dubai|Goa|Hyderabad

CS3: Enterprise Application Integration (EAI)

Source Courtesy: Some of the contents of this PPT are sourced from materials provided by Publishers of T1

Contents

- Enterprise Information Systems
- Enterprise Application Integration

Enterprise Information Systems

- Enterprise information systems:
  - Integrated ISs that support core BPs and functions.
  - e.g Marketing, Accounting, Finance, Info security, HR, Compliance, Production, Purchasing, and Logistics.
- Know them from terms e.g.
  - Supply Chain Management – For managing suppliers, inventory and shipping, etc.
  - Human Resource Management – For managing personnel, training and recruiting talents;
  - Employee Health Care – For managing medical records and insurance details of employees;
  - Customer Relationship Management – For managing current and potential customers;
  - Business Intelligence Applications – For finding the patterns from existing data from business operations.
  - ERP: Enterprise resource planning
  - CPFR: Collaborative planning, forecasting, and replenishment
  - KM: Knowledge management

innovate achieve lead

Enterprise Information Systems

- All these systems work as individual islands of automation. Most often these systems are standalone and do not communicate with each other due to incompatibility issues such as –
  - Operating systems they are residing on;
  - Database system used in the system;
  - Legacy systems not supported anymore.
- Main EIS need is for data integration (data sharing/ exchange):
  - e.g, ERP & SCM improve SC; KM & CRM for (un)profitable customers
  - All these are facilitated by communication!

EAI

- EAI is an integration framework, a middleware, made of a collection of technologies and services that allows smooth integration of all such systems and applications throughout the enterprise and enables data sharing and more automation of business processes.
- Characteristics of EAI
  - EAI is defined as "the unrestricted sharing of data and business processes among any connected applications and data sources in the enterprise."
  - EAI, when used effectively allows integration without any major changes to current infrastructure.
  - Extends middleware capabilities to cope with application integration.
  - Uses application logic layers of different middleware systems as building blocks.
  - Keeps track of information related to the operations of the enterprise e.g. Inventory, sales ledger and execute the core processes that create and manipulate this information.

EAI

- Need for EAI
  - Unrestricted sharing of data and business processes across an organization.
  - Linkage between customers, suppliers and regulators.
  - The linking of data, business processes and applications to automate business processes.
  - Ensure consistent qualities of service (security, reliability etc.).
  - Reduce the on-going cost of maintenance and reduce the cost of rolling out new systems.
- Challenges of EAI
  - Hub and spoke architecture concentrates all of the processing into a single server/cluster.
  - Often became hard to maintain and evolve efficiently.
  - Hard to extend to integrate 3rd parties on other technology platforms.
  - The canonical data model introduces an intermediary step.
  - Added complexity and additional processing effort.
  - EAI products typified.
  - Heavy customization required to implement the solution.
  - Lock-In – Often built using proprietary technology and required specialist skills.
  - Lack of flexibility – Hard to extend or to integrate with other EAI products!
  - Requires organization to be EAI ready.

EAI Types

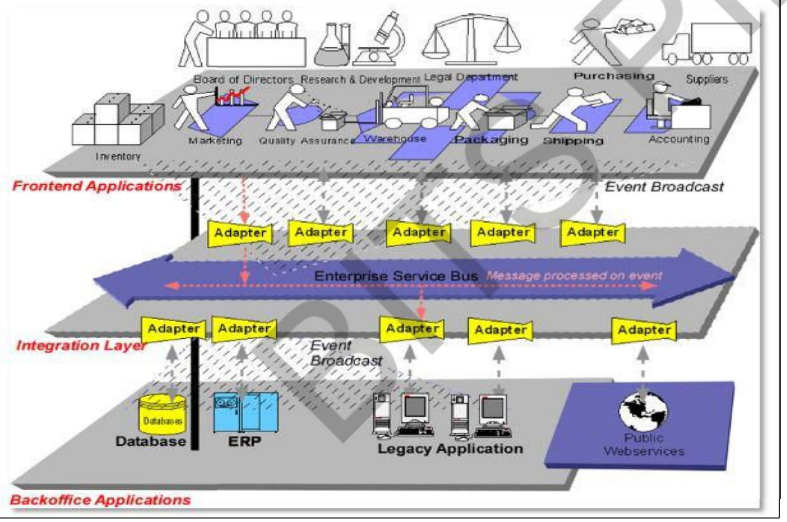
- There are three main types of EAI:
- Presentation Integration Model
- Data Integration Model
- Functional Integration Mode

- Data Level – Process, techniques and technology of moving data between data stores.
- Application Interface Level – Leveraging of interfaces exposed by custom or packaged applications.
- Method Level – Sharing of the business logic.
- User Interface Level – Packaging applications by using their user interface as a common point of integration.

ESB

- ESB -"enterprise service bus," is a software architecture that provides integration of enterprise applications and services for complex architectures, such as middleware infrastructure platforms.
- An ESB's primary function is to provide the connections between communicating applications
  - Acting much like a router to control the data. It is commonly used in enterprise application integration (EAI) or service-oriented architecture (SOA) principles.
  - The interaction and communication between components are across the bus, which has a similar function as a physical computer bus to handle data transfer or message exchange between services without writing any actual code.
- ESB as an infrastructure software service-oriented model works as a managed message system that provides routing, data transformation, translation upon a client's request and event-interpretation. It is often needed to transform messages into a format that the application can interpret. ESB is also used to change data content or execute services via a rule engine.

ESB

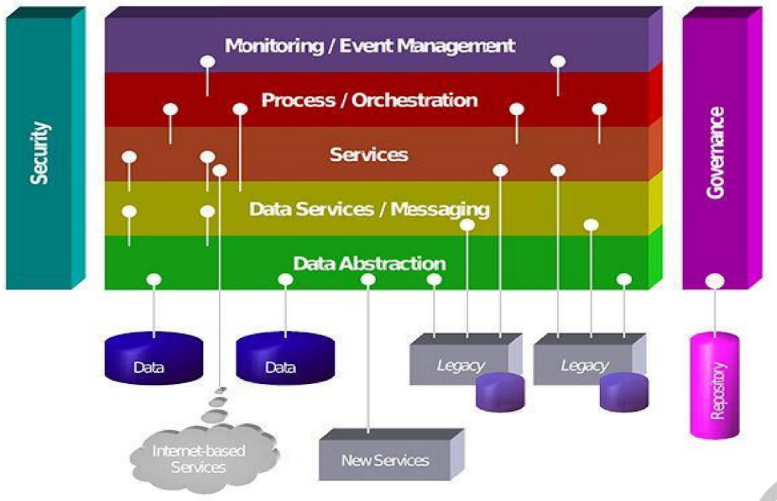


SOA

- Data & BP sharing between applications are EAI's primary purposes.
  - Links enterprise applications to talk to one another & do "batch" data transfers
  - But EAI also defines principles for linking multiple systems, such as message oriented middleware (MOM)
  - EAI is maybe old with SOA, but still EAI tools useful for large scale integrations
- SOA provides 'transactional' data transfer, needs no third-party s/w:
  - It differs from EAI in that it does not depend on a third-party solution.
  - Links interacting & contracted services via comms protocol (i.e. Web Services)
- Services are useful because they:
  - Are reusable in heterogeneous environments at multiple levels, including code, platform, so more flexible in the design of enterprise applications
  - Are implemented by 1/more code components in homogeneous environments
  - Aggregating 1/more components into a service, accessible through asynchronous messaging using open standards.

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

SOA



EAI, SOA or ESB: Which is Best?

	EAI Enterprise Application Integration	SOA Service Oriented Architecture	ESB Enterprise Service Bus
Pros	• Easy to quickly accomplish goals • Guaranteed delivery • Simplifies distributed asynchronous computing and access to data sources • Greater flexibility • Allows reuse of data across other applications	• Reusability of services in other applications • Easy to update and maintain • Greater reliability and fewer software errors • Location independent • Scalable and available • Platform independence	• Less customization required • Consistency and good practice in application integration • Boost operational efficiency and agility by consolidating apps • Wide selection of tools • Easier operational support due to transparency
Cons	• More complex architecture • Steeper learning curve • Performance bottlenecks possible • Difficulty accessing or maintaining business logic	• Increased overall costs due to higher response time and machine load • Complex service management • Significant upfront investment	• Risk of projects "bottlenecks" requires significant planning • Difficult to find developers with extensive ESB skills • Risk of regression • Single point of failure

Thank You

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Course Name : Middleware Technologies -CSIZG524

CS4: Integration Styles and Messaging Systems

BITS Pilani
Pilani Campus

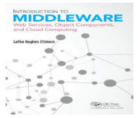
innovate achieve lead

Start Recording

IMP Note to Students

- It is important to know that just login to the session does not guarantee the attendance.
- Once you join the session, continue till the end to consider you as present in the class.
- IMPORTANTLY, you need to make the class more interactive by responding to Professors queries in the session.
- **Whenever Professor calls your number / name ,you need to respond, otherwise it will be considered as ABSENT**

Textbooks



T1: Letha Hughes Etzkorn - Introduction to middleware _ web services, object components, and cloud computing- Chapman and Hall_CRC (2017).

T2: William Grosso - Java RMI (Designing & Building Distributed Applications)

R1: Gregor Hohpe, Bobby Woolf - Enterprise Integration Patterns_ Designing, Building, and Deploying Messaging Solutions -Addison-Wesley Professional (2003)

R2: MongoDB in Action

Note: In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Modular Structure

No	Title of the Module
M1	Introduction and Evolution
M2	Enterprise Middleware
M3	Middleware Design and Patterns
M4	Middleware for Web-based Application and Cloud-based Applications
M5	Specialized Middleware

BITS Pilani
Pilani Campus

CS4: Integration Styles and Messaging Systems

Agenda

- Introduction
- Application Integration Criteria
- Application Integration Options

Application Integration Criteria

- If you can develop a single, stand-alone application that doesn't need to collaborate with any other applications
- We can avoid the whole integration issue entirely.
  - Application coupling
  - Integration simplicity
  - Integration technology
  - Data format
  - Data timeliness
  - Data or functionality
  - Asynchronicity

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Application Integration Criteria

- **Application Coupling:**
  - Integrated applications should minimize their dependencies on each other so that each can evolve without causing problems for the others.
  - Tightly coupled applications make numerous assumptions about how the other applications work
  - The interface for integrating applications should be specific enough to implement useful functionality
- **Integration Simplicity:**
  - Integrating an application into an enterprise, developers should strive to minimize changing the application and minimize the amount of integration code needed.
  - To provide good integration functionality and the approaches with the least impact on the application may not provide the best integration into the enterprise.

Application Integration Criteria

- **Integration Technology:**
  - Different integration techniques require varying amounts of specialized software and hardware.
  - These special tools can be expensive, can lead to vendor lock-in, and increase the burden on developers to understand how to use the tools to integrate applications.
- **Data Format:**
  - Integrated applications must agree on the format of the data they exchange, or must have an intermediate translator to unify applications that insist on different data formats.
  - A related issue is data format evolution and extensibility—how the format can change over time and how that will affect the applications.
- **Data Timeliness:**
  - Integration should minimize the length of time between when one application decides to share some data and other applications have that data.
  - Data should be exchanged frequently in small chunks, rather than waiting to exchange a large set of unrelated items.
  - Applications should be informed as soon as shared data is ready for consumption.

Application Integration Criteria

- **Data Functionality:**
  - Integrated applications may not want to simply share data, they may wish to share functionality such that each application can invoke the functionality in the others.
  - Invoking functionality remotely can be difficult to achieve, and even though it may seem the same as invoking local functionality, it works quite differently, with significant consequences for how well the integration works.
- **Asynchronocity:**
  - Where the remote application may not be running or the network may be unavailable—the source application may wish to simply make shared data available or log a request for a sub procedure call– Asynchronicity.

Application Integration Options

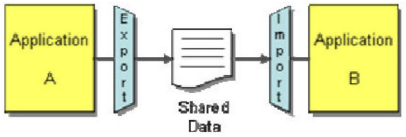
- File Transfer — Have each application produce files of shared data for others to consume, and consume files that others have produced.
- Shared Database — Have the applications store the data they wish to share in a common database.
- Remote Procedure Invocation — Have each application expose some of its procedures so that they can be invoked remotely, and have applications invoke those to run behaviour and exchange data.
- Messaging — Have each application connect to a common messaging system, and exchange data and invoke behavior using messages.

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

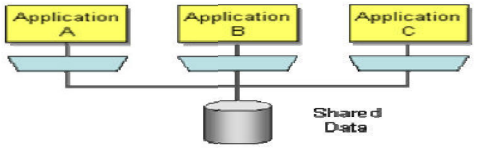
File Transfer

- Have each application produce files containing information that other applications need to consume.
- Integrators take the responsibility of transforming files into different formats.
- Produce the files at regular intervals according to the nature of the business.
- An important decision with files is what format to use. Very rarely will the output of one application be exactly what's needed for another, so you'll have to do a fair bit of processing of files along the way.
- File Transfer simple is that no extra tools or integration packages are needed, but that also means that developers have to do a lot of the work themselves.



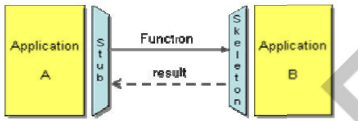
Shared Database

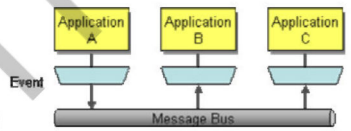
- File Transfer enables applications to share data, but can lack timeliness, yet timeliness of integration is often a critical issue. File Transfer also may not enforce data format sufficiently.
- Shared Database is made much easier by the widespread use of SQL-based relational databases. Biggest difficulties with Shared Database is coming up with a suitable design
- Multiple applications using a Shared Database to frequently read and modify the same data can cause performance bottlenecks and even deadlocks as each application locks others out of the data



Remote Procedure Invocation

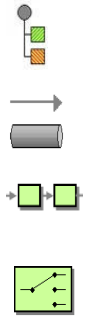
- Remote Procedure Invocation applies the principle of encapsulation to integrating applications.
- There are a number of Remote Procedure Call (RPC) approaches: CORBA, COM, .NET Remoting, Java RMI, etc.
- **Messaging:** Asynchronous messaging is fundamentally a pragmatic reaction to the problems of distributed systems.
- The high frequency of messages in Messaging reduces many of the inconsistency problems





Messaging Systems

- Message Structure is well defined by the messaging system used. It usually contains header and body sections.
- Channels are the mediums where messages are produced. They are the usual queue and topics
- Pipe and Filters pattern is useful when one needs to process messages before they are delivered to consumer applications.
- Message Router: When the sender application does not know on which channel the receiver is subscribed to. A router is used in between for delivering messages on the correct channel. It has routing rules that decides where the messages shall be delivered.

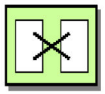


innovate achieve lead

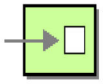
BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Messaging Systems

- Translators are used to change the format of message. The sender application might send CSV data while the receiver application understands XML, in that case we need to have a translator before the receiving application that does the job of CSV to XML conversion.



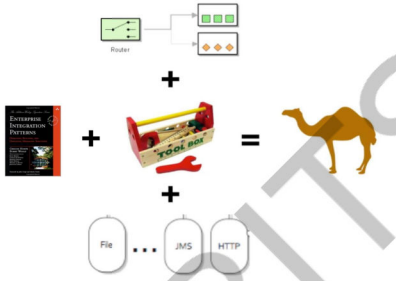
- Endpoint is a component that helps an application interact with messaging systems. They have the protocol built in for communication with the messaging systems. They are the message producers and the consumers.



Some Pattern examples

Publish-Subscribe Channel	Google Cloud Pub/sub
Dead Letter Channel	Amazon SQS
Return Address	GoLang
Content-based Router	Apache Camel
Message Filter	RabbitMQ
Event-driven Consumer	RabbitMQ
Competing Consumers	Apache Kafka
Channel Purger	Amazon SQS

Integration Framework – Apache Camel



Introduction to Apache Camel

- Open-source integration framework for routing and mediation of messages.
- Implements Enterprise Integration Patterns (EIP) using Java objects.
- Bridges communication protocols like HTTP, FTP, JMS, etc.
- Lightweight and embeddable in apps or cloud environments.
- Just like real plumbing pipes, Camel takes data from one point, and pipes it to another.
- Along the way, the data can be changed, transformed, or sent through other pipes.
- Included in your plumbing toolkit is a range of adaptors that fit all sorts of different apps and software systems.
- Camel was inspired by the book Enterprise Integration Patterns, which is an academic textbook about integrating software systems.

innovate achieve lead

Why Apache Camel?

- Simplifies Integration: Avoids custom protocol code.
- Declarative Routing: Easy DSL-based rule definitions.
  - Fluent Java
  - Spring XML
  - Blueprint XML
  - Scala
- Rich Ecosystem: 300+ components for various systems.
- Flexible Deployment: Standalone, embedded, or cloud.
- Supports Multiple Languages: Java, XML, Groovy, Kotlin, YAML.

Working of Apache Camel

- Define Routes: Set message paths from source to destination.
- Components: Built-in connectors for protocols/APIs.
- Processor: Validate, transform, or log messages.
- Router: Intelligent content-based routing.
- Example: HTTP to DB with optional email alerts.

Camel Architecture

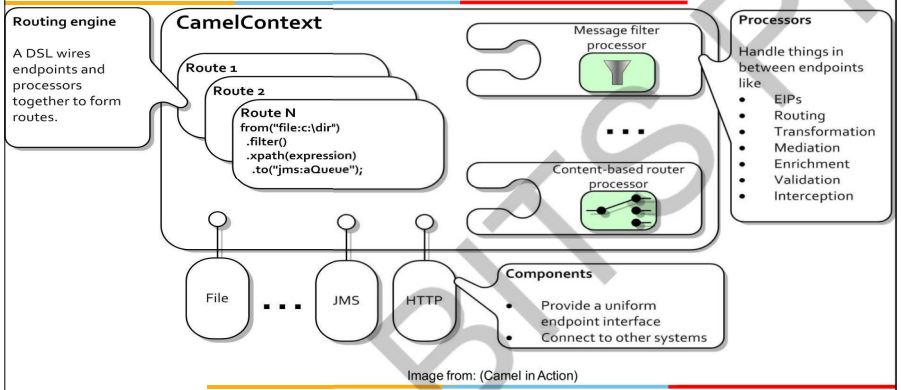


Image from: (Camel in Action)

Key Concepts in Camel

- Route: Message journey through processing steps.
- Endpoint: Interface to external systems (file, DB, HTTP).
- Processor: Java code for message manipulation.
- Component: Bridges Camel to external tech (FTP, JMS).
- CamelContext: Manages routes and components.