Distributed Systems – Technology

1. Modern Technologies for Distributed Applications

Two-tier architecture model

- Dominated the early years of distributed application architecture
- Client-server architecture
  - the first (upper) tier handles the presentation and business logic of the user application (client),
  - the second/lower tier handles the application organization and its data storage (server).
- Widely used in
  - enterprise resource planning,
  - billing,
  - inventory application systems (client business applications residing in multiple desktop systems interact with a central database server).

Common limitations of the client-server application model

- Complex business processing at the client side demands ***robust client*** systems.
- ***Security*** is more difficult to implement because the algorithms and logic reside on the client side making it more vulnerable to hacking.
- Increased network bandwidth is needed to accommodate many calls to the server, which can impose ***scalability*** restrictions.
- ***Maintenance and upgrades*** of client applications are extremely difficult because each client has to be maintained separately.
- Client-server architecture suits mostly database-oriented standalone applications and ***does not target*** robust reusable component-oriented applications.

CORBA

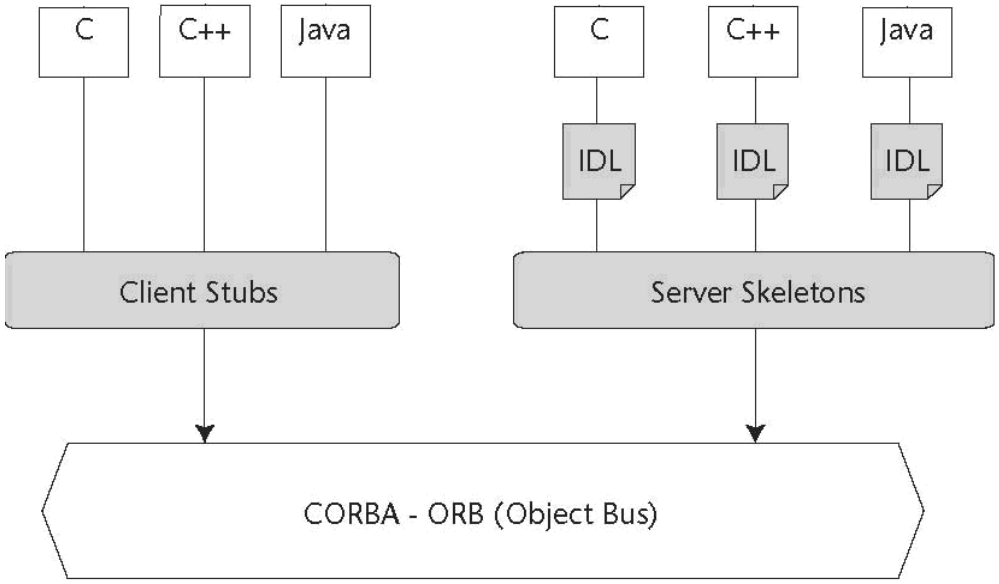
- Common Object Request Broker Architecture
- industry wide, open standard initiative, developed by the Object Management Group (OMG)
- differs from the traditional client/server model:
  - it provides an object-oriented solution that does not enforce any proprietary protocols or any particular programming language, operating system, or hardware platform
- Interface Definition Language (IDL) is a specific interface language designed to expose the services (methods/functions) of a CORBA remote object.
- defines a collection of system-level services for handling low-level application services like life-cycle, persistence, transaction, naming, security, and so forth.

ORB

- Object Request Broker
- is an object bus that provides a transparent mechanism for sending requests and receiving responses to and from objects, regardless of the environment and its location.
- intercepts the client's call and is responsible for finding its server object that implements the request, passes its parameters, invokes its method, and returns its results to the client.
- provides interfaces to the CORBA services, which allows it to build custom-distributed application environments.
- CORBA 2.0 added interoperability between different ORB vendors by implementing an Internet Inter-ORB Protocol (IIOP):
  - IIOP defines the ORB backbone, through which other ORBs can bridge and provide interoperation with its associated services.

CORBA architectural model

- IDL contracts to specify the application boundaries and to establish interfaces with its clients
- ORB acts as the object bus or the bridge, providing the communication infrastructure to send and receive request/responses from the client and server.



CORBA advantages

- OS and programming-language independence.
- Legacy and custom application integration.
- Rich distributed object infrastructure.
- Location transparency.
- Network transparency.
- Dynamic invocation interface.

CORBA disadvantages

- ***High initial investment.*** CORBA-based applications require huge investments in regard to new training and the deployment of architecture, even for small-scale applications.
- ***Availability of CORBA services.*** The Object services specified by the OMG are still lacking as implementation products.
- ***Scalability.*** Due to the tightly coupled nature of the connection-oriented CORBA architecture, very high scalability expected in enterprise applications may not be achieved.

Java RMI

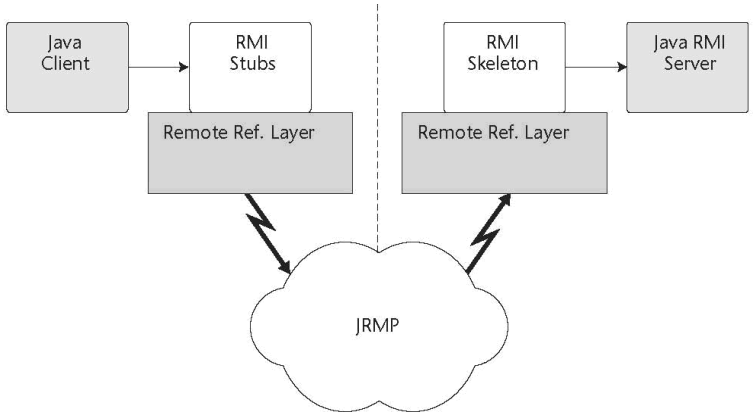
- enables object-oriented distributed computing
- developed by Sun Microsystems as the standard mechanism to enable distributed Java objects-based application development
- allows to call remote Java objects and passing them as arguments or return values.
- it uses Java object serialization—a lightweight object persistence technique that allows the conversion of objects into streams.

RMI architectural model (1)

- a registry (rmiregistry)-oriented mechanism provides a simple non-persistent naming lookup service that is used to store the remote object references and to enable lookups from client applications
- Java Remote Method Protocol (JRMP) is the inter-process communication protocol, enabling Java objects living in different Java VMs to transparently invoke one another's methods
- a reference-counting garbage collection mechanism keeps track of external live object references to remote objects (live connections) using the VM

RMI architectural model (2)

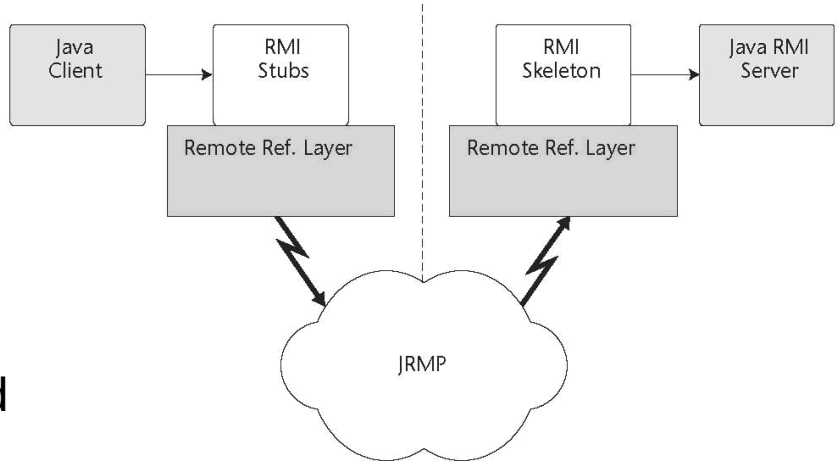
- RMI client: Java applet / stand-alone application,
  - performs the remote method invocations on a server object;
  - pass arguments that are primitive data types or serializable objects.
- RMI stub: the client proxy
  - generated by the RMI compiler (*rmic* in JDK)
  - encapsulates the network info of the server
  - performs the delegation of the method invocation to the server.
  - marshals the method arguments and
  - unmarshals the return values from the method execution.



- RMI infrastructure with two layers:
  - the remote reference layer
    - separates out the specific remote reference behavior from the client stub;
    - handles certain reference semantics like connection retries, which are unicast/multicast of the invocation requests;
  - the transport layer facilitates
    - the actual data transfer during method invocations,
    - the passing of formal arguments, and
    - the return of back execution results.

RMI architectural model (3)

- RMI skeleton:
  - generated using the RMI compiler (rmic)
  - receives the invocation requests from the stub and
  - processes the arguments (unmarshalling) and
  - delegates them to the RMI server.
  - marshals the return values and passes them back to the RMI stub via the RMI infrastructure.
- RMI server: the Java remote object that
  - implements the exposed interfaces and executes the client requests
  - receives incoming remote method invocations from the respective skeleton, which passes the parameters after unmarshalling.
  - return values are sent back to the skeleton, which passes them back to the client via the RMI infrastructure.



Advantages of Java RMI

- Developing distributed applications in RMI is simpler than developing with Java sockets: there is no need to design a protocol
- RMI is built over TCP/IP sockets, but the added advantage is that it provides an object-oriented approach for inter-process communications.
- provides an efficient, transparent communication mechanism that frees the programmers of all the application-level protocols necessary to encode and decode messages for data exchange.
- RMI enables distributed resource management, best processing power usage, and load balancing in a Java application model.
- RMI-IIOP (RMI over IIOP): protocol developed for enabling RMI applications to interoperate with CORBA components.

Disadvantages of Java RMI

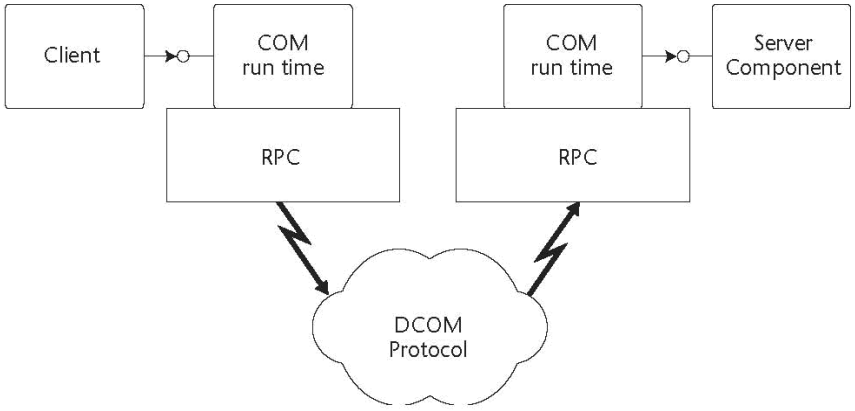
- RMI is limited only to the Java platform.
  - It does not provide language independence in its distributed model as targeted by CORBA.
- RMI-based application architectures are tightly coupled because of the connection-oriented nature.
  - Achieving high scalability in such an application model becomes a challenge.
- RMI does not provide any specific session management support.
  - In a typical client/server implementation, the server has to maintain the session and state information of the multiple clients who access it.
  - Maintaining such information within the server application with-out a standard support is a complex task.

Microsoft DCOM

- Microsoft Component Object Model (COM)
- a way for Windows-based software components to communicate with each other by defining a binary and network standard in a Windows operating environment.
- enables COM applications to communicate with each other using an RPC mechanism, which employs a DCOM protocol

Architectural model of Microsoft DCOM

- skeleton and stub approach
- the stub
  - encapsulates the network location information of the COM server object and
  - acts as a proxy on the client side.
- the servers
  - can potentially host multiple COM objects,
  - register themselves against a registry,
- clients discover servers using a lookup mechanism



Advantages and disadvantages of DCOM

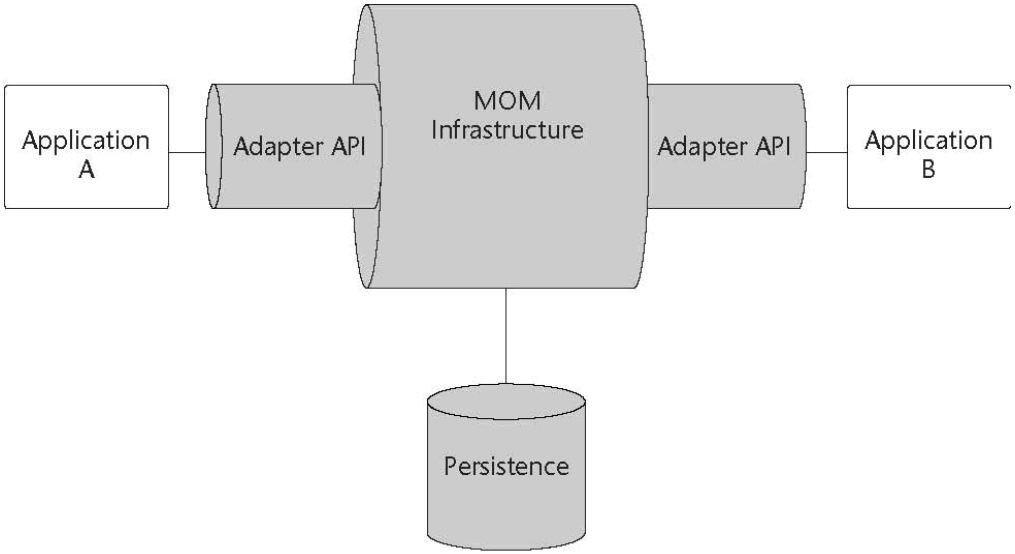
- successful in providing distributed computing support on the Windows platform.
- Common limitations of DCOM:
  - Platform lock-in : limited to Microsoft application environments
  - State management
  - Scalability
  - Complex session management issues

Message-Oriented Middleware (MOM)

- CORBA, RMI, and DCOM adopted a tightly coupled mechanism of a synchronous communication model (request/response).
- MOM is based upon a loosely coupled asynchronous communication model
  - the application client does not need to know its application recipients or its method arguments.
- Enables applications to communicate indirectly using a messaging provider queue.
  - The application client sends messages to the message queue (a message holding area), and
  - The receiving application picks up the message from the queue.
  - The application sending messages to another application continues to operate without waiting for the response from that application.

MOM-based architectural model

- applications interacting with its messaging infrastructure use custom adapters.
- for reliable message delivery, messages can be persisted in a database/file system as well.



MOM Implementations and limitations

- SunONE Message Queue, IBM MQSeries, TIBCO, SonicMQ, and Microsoft Messaging Queue (MSMQ).
- JMS: Java Message Service, is developed as part of the Sun Java Community Process (JCP) and also is currently part of the J2EE
  - JMS provides Point-to-Point and Publish/Subscribe messaging models with the following features: complete transactional capabilities, reliable message delivery, security.
- Common challenges while implementing a MOM-based application environment :
  - Most of the standard MOM implementations have provided native APIs for communication with their core infrastructure
    - this has affected the portability of applications across such implementations and has led to a specific vendor lock-in.
  - The MOM messages used for integrating applications are usually based upon a proprietary message format without any standard compliance.

Common Challenges in Distributed Computing

- Context:
  - CORBA, RMI, and DCOM successful in integrating applications within a homogenous environment inside a LAN
  - Internet scale demands the interoperability of applications across networks
- Maintenance of various versions of stubs/skeletons in the client and server environments is extremely complex in a heterogeneous network environment
- Quality of Service (QoS) goals like Scalability, Performance, and Availability in a distributed environment consume a major portion of the application's development time.
- Interoperability of applications implementing different protocols on heterogeneous platforms almost becomes impossible
  - E.g. a DCOM client communicating to an RMI server or an RMI client communicating to a DCOM server.
- Most of these protocols are designed to work well within local networks
  - They are not very firewall friendly or able to be accessed over the Internet.

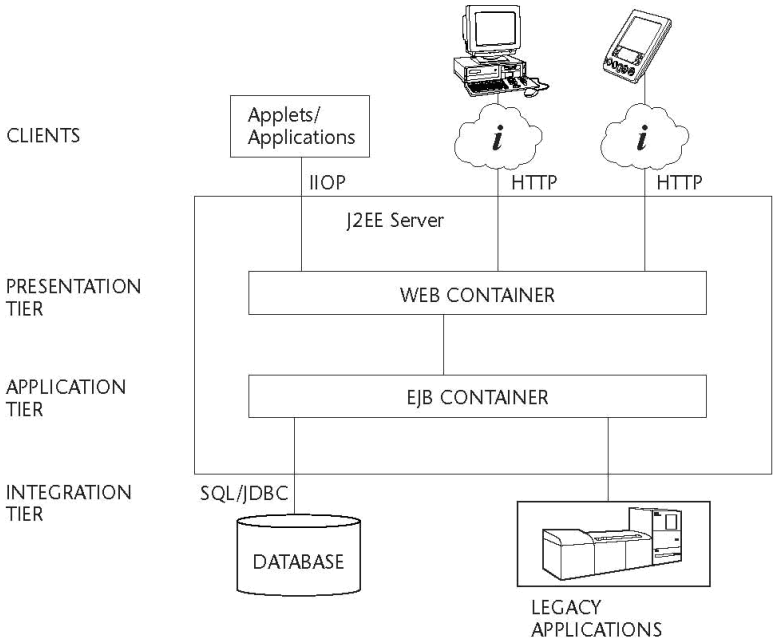
Towards Internet scale and role of J2EE

- Internet-based enterprise application model:
  - the focus is moved the complex business processing toward centralized servers in the back end (cloud computing?)
- first generation of Internet servers:
  - based upon Web servers that hosted static Web pages
  - provided content to the clients via HTTP (HyperText Transfer Protocol).
- second generation: server-side scripting
- third generation: business-to-business (B2B) and business-to-consumer (B2C) on Internet
- In this context J2EE
  - provides a programming model based upon Web and business components that are managed by the J2EE application server.
  - the application server consists of many APIs and low-level services available to the components.
  - low-level services provide security, transactions, connections and instance pooling, and concurrency services,
    - which enable a J2EE developer to focus primarily on business logic rather than plumbing.

Typical J2EE architecture

Three logical tiers

- Presentation tier composed of Web components, which handle
  - HTTP requests/responses,
  - Session management,
  - Device independent content delivery, and
  - the invocation of business tier components.
- Application tier (Business tier) deals with the core business logic processing (workflow & automation).
  - retrieve data from the information systems with well-defined APIs provided by the application server.
- Integration tier deals with connecting and communicating to
  - back-end Enterprise Information Systems (EIS),
  - database applications and
  - legacy applications,
  - or mainframe applications.



Role of XML in Distributed Computing

- Extensible Markup Language (XML)
- Defines portable data in a structured and self-describing format
- Embraced by the industry as a communication medium for electronic data exchange.
  - Has been widely adopted and accepted as a standard by major vendors in the IT industry, including Sun, IBM, Microsoft, Oracle, HP.
  - Provides a new way of application-to-application communication on the Internet.
- Promotes inter-operability between applications
- Enhances the scalability of the underlying applications
- Promotes a new form of the distributed computing technology solution referred to as Web services.