CMPSCI 677 Operating Systems **Spring 2014**

Lecture 25: April 28

Lecturer: Prashant Shenoy *Scribe: Nischal Bommareddy*

25.1 Distributed Middleware

25.1.1 Corba

CORBA stands for Common Object Request Broker Architecture. Object Request Broker(ORB) Layer is the core component of this middleware and handles all the communication between the objects. All the requests made by the objects flow through the ORB layer. This layer can be implemented as a library or as a separate process. CORBA provides a wide variety of services and is a fairly heavyweight middleware.

25.1.1.1 Services

Some of the services provided by CORBA include

Query Service: Applications can use this service to query data

Transactional Service: These services can be used to implement transactions inside the application

Licensing Service: These services are used by commercial applications

Security Service: These services can be used to implement encryptions

25.1.1.2 Object Model

Clients and servers communicate using an RMI based interface. The ORB layer at the bottom performs RPCs and RMIs. IDL Proxy on the client allows the methods and functions to be written in Interface Definition Language. When compiled through an IDL compiler, a stub code is generated which handles setting up of connections.

25.1.1.3 Object Invocation Models

The different types of object invocation models include

Synchronous model: In this model, the caller is blocked until a response is returned or an exception is raised

One-way model: In this model, delivery is best effort and caller continues immediately without waiting for the response

Deferred Synchronous: This model involves two asynchronous one-way calls, one in forward direction and the other in backward direction

25.1.1.4 Event Notification Services

In CORBA, Event channel is used to used to implement publish-subscribe applications (PUB-SUB model).There are two main ways to implement the publish-subscribe model.

Push Based Paradigm: In this model, consumers subscribe to a particular type of data or events. Publishers create the data and push the data into the event channel. Event channel can be viewed as a buffer. Event channel will deliver the data to the corresponding consumers who subscribed for this type of data

Pull Based Paradigm: In this model, Event channel will keep polling the publisher for data. If there is new data, Event channel will store the data. Subscribers in return poll the Event channel and pull the data from the Event channel.

25.1.1.5 Asynchronous Method Invocation

In CORBA, asynchronous method invocations can be made in two ways

Call back model: This model can be implemented using a Callback interface. A callback function is registered and on receiving the response, ORB will call this registered function to contact the client

Polling model: In this model the reply will be buffered at the ORB. Client application will periodically poll the ORB through the polling interface (non blocking interface) for the response.

CORBA is a heavy weight middleware and was not commercially successful. Performance of the application is really low as a simple call requires a call to the middleware.

25.1.2 DCOM - Distributed Component Object Model

This is Microsoft's middleware stack. This middleware layer powers .NET. The three different layers responsible for the evolution of DCOM are COM,OLE (Object Linking and Embedding) and ActiveX.

25.1.2.1 History of DCOM

In order to support compound documents(Word Documents with excel spreadsheets and powerpoint images) OLE and COM were developed. OLE Layer allows one kind of document or object to be embedded into another. If the embedded object is to be edited, double-clicking the object will open the object in corresponding Microsoft application. The Microsoft applications communicate through RPCs and pass the objects to one another. OLE layer was succeeded by COM Layer which allows any application to communicate with another one. COM is replaced by ActiveX. ActiveX was used to write interactive applications for browsers(Internet Explorer). DCOM implements all the above things and works in a distributed environment(all the objects need not be present on the same machine) .

25.1.2.2 Architecture

Architecture of DCOM is similar to that of CORBA. The Type library of DCOM is similar to CORBA interface repository. Each method stores its interface in Type library. Type library communicates with the Service Control Manager(SCM). SCM is similar to the CORBA implementation repository. The interfaces are registered in SCM. COM layer is similar to the ORB layer. It is responsible for the interprocess communication. The stub layer resides on the top of the COM layer.

25.1.2.3 Persistent Objects

By default DCOM objects are transient. If DCOM objects are made persistent, Middleware keeps a copy of the object's state on the disk. If the process is restarted, the object can be reloaded. Persistent objects are implemented using monikers. Monikers allows you to go back and read the object state and reload the object

25.1.2.4 Distributed Coordination

Distributed Coordination model involves designing distributed applications that are loosely coupled.

Coordination models The four main types of communication models for applications are:

Direct communications: Both the entities in the communications are coupled in time and space

Mail box: Both the entities are coupled in space, but decoupled in time. Email is the example of the this type of communication. The sender and the receiver know who they are sending the mail to.

Meeting room: Both the communicating entities are coupled in time but decoupled in space.

Generative communication: Communicating entities are decoupled in both time and space

25.1.2.5 Jini

Jini is a Java based middleware. Clients can discover the services on the fly and use them when they are available. Discovery of services is done using loose coordination model. This model is referred to as bulletin board model communication. Bulletin board model uses JavaSpace which is the shared data store that stores tuples.

25.1.2.6 Overview of Jini

Applications insert tuples in the shared data space. The tuple instances are not addressed to anyone. At a later point of time, any application can search the JavaSpace for tuples matching the requirement criteria. After finding the tuple satisfying the required criteria, the application can either remove the tuple from the java space or leave it there for some other application.

Asynchronous callbacks can also be used to implement this model. Applications register with the bulletin board and request notifications whenever a tuple matching the requirement criteria is put in the shared data space. On receiving a matching tuple, the bulletin board sends notifications to the applications that have registered for this type of tuple.

25.1.3 Implementation Mechanisms

There are two ways to implement the bulletin board model

- In the first implementation, JavaSpace is replicated on all the machines. Whenever a tuple is put up on one of the bulletin board, the WRITE is broadcasted to maintain consistency across all the machines. READs are done locally. If a tuple is deleted from a bulletin board, all the other bulletin boards must be notified
- The second implementation involves a partitioned JavaSpace. A WRITE operation is done locally. A READ query must be broadcasted to every JavaSpace.

Jini was originally designed for embedded systems to enable them to interact with other smart embedded devices and configure themselves. Bulletin board based zero configuration mechanisms are used in setting up distributed network services.