

# Distributed Objects & RMI

1

## Topics for this lecture

- Middleware
- Interfaces (IDL)
- The Distributed Object Model
- RMI: Remote Method Invocation
  - An extension to local method invocation that allows remote method invocation also
- Middleware
  - CORBA
  - COM and DCOM
- SOAP

2

## Rationale for this lecture

- We need to consider programs running with different processes on different computers
- Programs need to invoke methods in other process on other computers
- How do they do it while ensuring:
  - Scalability
  - Transparency
  - Reliability
  - Security
  - Openness

3

## Introduction

- Sockets API used for sending & receiving calls (simple IO)
- Remote Procedure Calls (RPC)
  - Was designed to provide a procedural interface for distributed (i.e., remote) services
  - The aim was to make the distributed nature of service transparent to the programmer
  - However, it is no longer considered particularly good
- Remote Method Invocation (RMI)
  - Essentially combines RPC with Object Orientation
  - Allows objects living in one process to invoke methods of an object living in another process

4

## Middleware

- Middleware abstracts much of the detail and heterogeneity in systems
- Middleware provides, location transparency, independence from the communication protocols, operating systems, hardware etc.

5

## Middleware

- What middleware hides / provides
  - Provides location transparency:
    - A programmer could make a call to method without knowing where the method is physically located (same host, different host)
  - Underlying communication protocols are hidden
    - RR could be either UDP or TCP
  - Marshalling details hidden and automatic
  - Independence from different OS
  - CORBA allows different programming languages to talk
    - Using interface definition language (IDL)

6

# Interfaces

- What is an interface

- Not a GUI

- Interface in this unit refers to the means of connection for two different systems

- In software

- Units of code are modularized – that is organized into modules

- Modules may use other modules via either calling their methods or through direct access to their data

- They do this using interfaces

7

# Interfaces

```
public int sumNumbers(int a, int b){
    return a+b;
}
```

← Here we have a method that sums two integers together

And here we have another method that references and calls the above →

```
private void myCalc(){
    System.out.println(sumNumbers(10,10));
    System.out.println(sumNumbers(5,7));
}
```

8

# Interfaces

## •Interfaces

- The calling method does not need to know or care how the code is written
- So long as the interface remains the same method B may call method A

Here we have changed the INTERNAL implementation of sumNumbers

```
public int sumNumbers(int a, int b){
    int result = a+b;
    return result;
}
```

However, no change is required in the client code as the INTERFACE remained the same

```
private void myCalc(){
    System.out.println(sumNumbers(10,10));
    System.out.println(sumNumbers(5,7));
}
```

9

# Interfaces

## •Interfaces in a DIS

- Pass by reference is not appropriate
  - As a local memory address has no meaning to another computer
- Arguments are either **Input** or **Output** arguments
  - Input** arguments contains values that will be used by the server
  - Output** arguments are used for the server returning values to the client
- No direct attribute access is generally permit
  - Everything occurs through messages

10

# Interfaces

- Interfaces in a DIS

- IDL: Interface Definition Language

- Provides an agreed convention for describing the interfaces between methods
    - This includes distinguishing between **Input** and **Output** arguments

11

# Interfaces

- An example CORBA IDL

```

struct Person {
    string name;
    string place;
    long year;
};
Interface PersonList{
    readonly attribute string listname;
    void addPerson(in Person p);
    void getPerson(in string name, out Person p);
    long number();
};

```

Contains marshalling information

The diagram shows four arrows originating from the text 'Contains marshalling information' on the right. One arrow points to the 'struct Person' definition. Three arrows point to the 'Interface PersonList' definition, specifically highlighting the 'readonly attribute string listname;', the 'void addPerson(in Person p);' method, and the 'void getPerson(in string name, out Person p);' method, which all contain input/output annotations.

12

## Interfaces

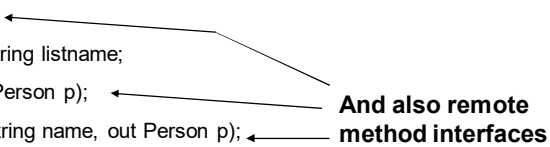
- An example CORBA IDL

```

struct Person {
    string name;
    string place;
    long year;
};

Interface PersonList{
    readonly attribute string listname;
    void addPerson(in Person p);
    void getPerson(in string name, out Person p);
    long number();
};

```



**And also remote method interfaces**

13

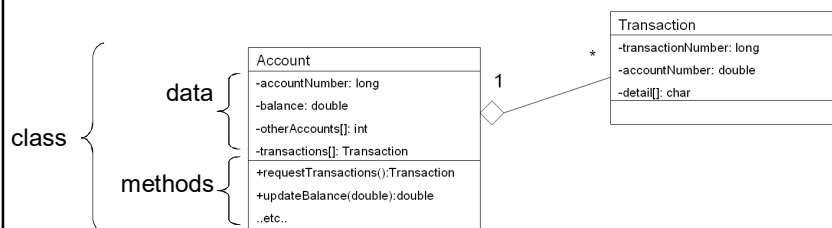
## Distributed Object Communication

- The Object model concerns single computers
  - The object model embodies many desirable design concepts, such as openness, scalability, etc.
- The Distributed Object model concerns multiple distributed computers
  - How the object model is extended for use in a DIS
- And... what are the key differences between the distributed object model and the object model?

14

# Distributed Object Communication

- The object model
  - Objects have their own data + their own methods
  - Objects interact via methods
  - Objects may allow other objects to access internal data
    - However in a DIS this is not possible due to non-locality of reference



15

# The Object Model

- The object model
  - Object references
    - All objects are actually references to objects
    - Methods are accessed via the object references
  - Interfaces
    - A definition of a method or a set of methods
    - Does not refer to the implementation of the method but
    - Contains types of arguments, return types of the method, etc.
  - Actions
    - Actions occur from method invocations
    - Invocations might supply arguments (data) in making an invocation
      - i.e. result = sumNumbers(10,2);
    - Once received the receiver executes the method and returns any results that are required
    - The receiver state might change as a result
    - Requests to other methods might also be made

16



## The Object Model

### –Exceptions

- Things can go wrong
  - Data might be inconsistent
  - Hardware might not be available
- Exceptions catch errors that occur so that the software does not crash in an uncontrolled manner
- When an error occurs control jumps to the handler for the error
  - Example, `try{some code}catch (some error){some other code}`

### –Garbage collection

- Objects are instantiated and either de-instantiated or cease being referenced anywhere in the program
- The memory allocated to these objects must be made available
- Thus, a system for collecting dead objects and data is required
- This is referred to as garbage collection

17

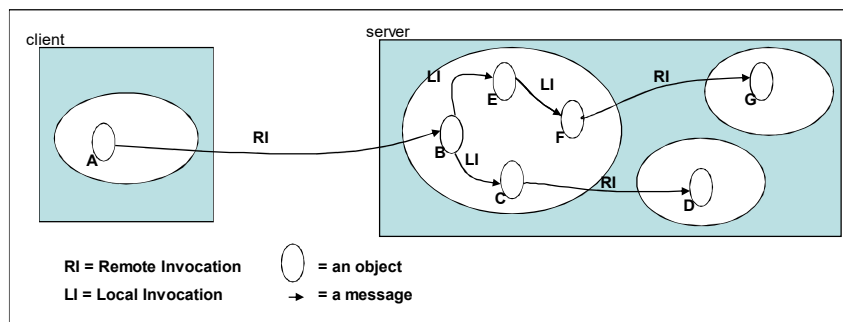
## Distributed Object Model

- In the OM objects are encapsulated
  - They are accessed via their interfaces only
  - As such objects can run in separate processes
- Distributed objects also run as separate processes communicating via their interfaces
- However, in a DIS processes can run at different computers
  - Perhaps communicating via RMI
- In a DIS a program would use a series of local and remote processes during its execution

18

## Distributed Objects

- A client server example



19

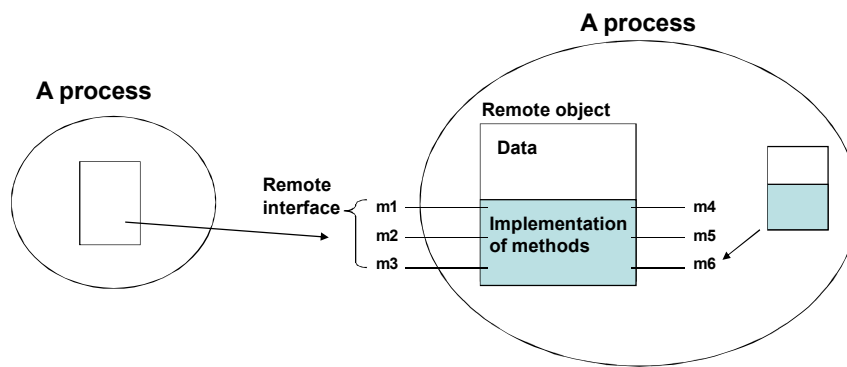
## The Distributed Object Model

- Objects may be defined to receive local or non-local requests
- Local invocations are requests made from local processes to other local processes
  - i.e. B to G, B to E, etc.
- Remote invocations are requests to non-local (remote) processes
  - i.e. A to B, F to G etc.
- In order to make a remote request invokers **must** have a Remote Object Reference
  - i.e. A must have Remote Object Reference for B, etc.
- Remote objects **must** have a remote interface that specifies methods that can be invoked remotely
  - i.e. B, D and G

20

# The Distributed Object Model

- Remote interface example



21

# The Distributed Object Model

- Extensions to the the Object Model (see slides 12,13)

- Object references

- Similar to the classic Object Model

+++Uses a remote object reference, essentially analogous to a regular local object reference

- Interfaces

- Similar to the classic Object Model

+++ defines methods that can be remotely invoked

- Actions

- Similar to the classic Object Model

+++Actions may require methods from remote objects, these remote objects have to be available

- Exceptions

- Similar to the classic Object Model

+++There are more reasons to fail and more exceptions, remote process failure, timeouts, etc.

- Garbage collection

+++remote garbage collection requires coordination is required between local garbage collectors

- Similar to the classic Object Model

22

# RMI

## •RMI design issues

### –Invocation semantics

- Maybe semantics
  - No guarantees, it may or may not have executed
- At least once semantics
  - Semi-robust method executes at least once but maybe more than once
  - Requires idempotent server operations
- At most once semantics
  - Robust whereby a method executes at most once

### –Transparency

- Access, location, migration, failure, etc.

23

# RMI

## •How RMI is implemented?

### –RMI is composed of many objects and modules that together provide the RMI implementation:

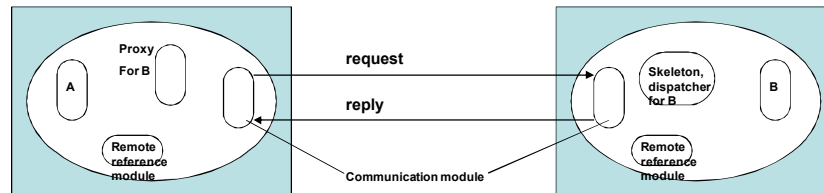
- Communication module
- Remote reference module
- RMI software
  - Proxy
  - Dispatcher
  - Skeleton
- A binder
- Garbage collection
- Activation mechanisms
- And there are more

24

# RMI

## •How RMI is implemented

Example: Object A makes a remote method invocation of object B



### –Client

- Contains the proxies for the remote objects
- The remote reference module
- Can also have a binder to reference remote objects
- A communication module

### –Server

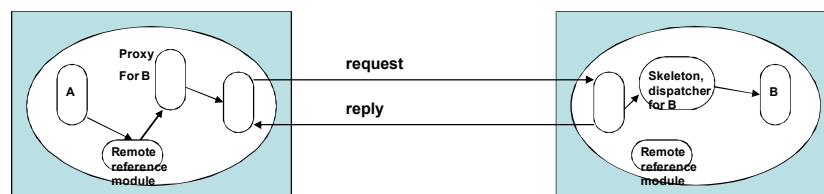
- Contains skeletons and dispatchers
- Communication module

25

# RMI

## •How RMI is implemented

Example: Objects A makes a remote method invocation of object B



### –An example of a request

26

# RMI

- Communication module
  - Implements the Request-Reply protocol that is used for communication
- Remote reference module
  - Determines whether references are local or remote
  - Responsible for creating object references
  - Contains a Remote Object Table
    - Stores references to local and remote objects

27

# RMI

- Proxy
  - Makes remote methods transparent to local processes
  - Remote object references are directed to the proxy which passes them to the remote method
  - Thus, there can be one for each remote object in the Remote Object Table
  - The proxy marshals data and un-marshals results
  - The proxy implements the interfaces of remote methods and send calls to them using the interfaces

28

## RMI

- Dispatcher

- Receives messages from the communication module, selects and passes on the message to the right skeleton for the remote object

- Skeleton

- Un-marshals the *request* and calls the relevant remote method
  - Marshals the response into the *reply*
  - Sends to the requesting proxy (through the communication module)

29

## RMI

- Servant

- provides the implementation for a remote class.
  - Runs in a server process
  - is the recipient of the remote request from the skeleton

30

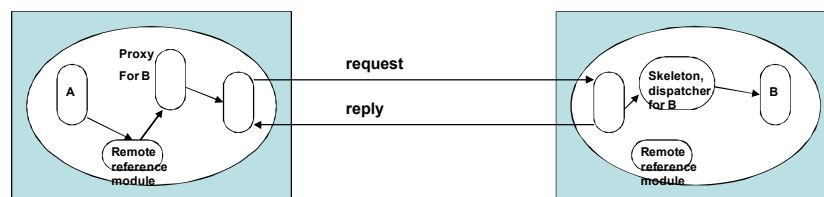
# RMI

- Proxies, dispatchers and skeletons
  - These tend to be generated by middleware software using compilers
  - CORBA IDL is compiled into skeletons and stubs
  - Similarly for Java RMI when the remote interface is used.

31

# RMI

Example: Objects A makes a remote method invocation of object B



–An example of a request

32



## RMI

- Static invocation
  - Proxies are generated pre-runtime using the remote interfaces
- Dynamic invocation
  - Useful when all remote interfaces are not known at design time
  - Client supplies a remote object reference, name of method and arguments
  - The middleware then determines which remote object is required

33

## RMI

- Binder
  - A client service that obtains and stores remote object reference
    - Java RMIregistry, CORBA Naming service
  - It stores mappings from textual names to remote object references
  - Servers need to register the services they are exporting with the client binder
- Sever
  - Can choose to execute a thread for each RMI

34

# RMI

## Activation of remote objects

### –Active and passive objects

- An Active object is an instantiated in a running process
- A Passive object is an object not currently active but can be made Active
  - i.e. its Implementation, and marshalled state are stored on disk

### –Activator is responsible for

- Registering passive objects that are available for activation
- Keeping track of the locations of servers for remote objects that have already been activated
- Starting named server processes and activating remote objects within them

35

# RMI

## •Persistent object stores

- An object that lives between activations of processes is called a persistent Object
- Stores the state of an object in a marshalled (serialised) form on disk
  - Recall in Java the implements serialisable

## •Location service

- Objects can move (Migrate) from one system to another during their lifetime
- Solution is to maintains mapping between object references and the location of an object
- May involve broadcasts to track down objects

36

## RMI

### –Distributed garbage collection

- Local hosts have their own garbage collectors
- Traditionally an object is dead when it is no longer referenced
- However, in a DIS a remote method might still reference the object

### •Java's solution

- Servers keep a list of references
- When a client proxy is created *addReference* to server list
- When a client proxy is deleted *removeReference* from server list

37

## Events and Notification

### •Rationale

- Objects can react to changes in other objects
- Local event driver applications are common place
  - Windows is event driven
    - User events trigger actions (onClick, etc.)
  - Changes to data might precipitate an update
- Distributed event
  - Similar but needs to account for distribution
  - Change state of a remote object might require a change in state of other remote objects

38

## Events and Notification

- Operates using the publish/subscriber pattern
  - Objects publish a list of events that maybe subscribed too
  - Subscribers register their interest in those events
  - The publisher is then responsible for publishing notifications to all of the events subscribers when the event occurs
  - A notification is an object that represents the event
  - This scheme helps with heterogeneity
    - A subscriber simply has to implement an interface to process notifications
  - Choice of delivery is also important (reliable or unreliable)

39

## CORBA

### • What is it?

*“CORBA is a middleware that allows application programs to communicate with one another irrespective of their programming languages, their hardware and software platforms, the networks they communicate with and their implementers”*

[Coulouris et al., 2001]

- Sponsored by the OMG (Object Modelling Group)

40

## **CORBA basics**

- Key components:
  - ORBs
  - IDL
  - CORBA Services

41

## **CORBA: Architecture**

- Key concepts
  - ORB core
  - Object adaptor
  - Skeletons
  - Client stub/proxies
  - And some others:
    - Repositories (implementation & interface)
    - Dynamic invocation interface
    - Dynamic interface

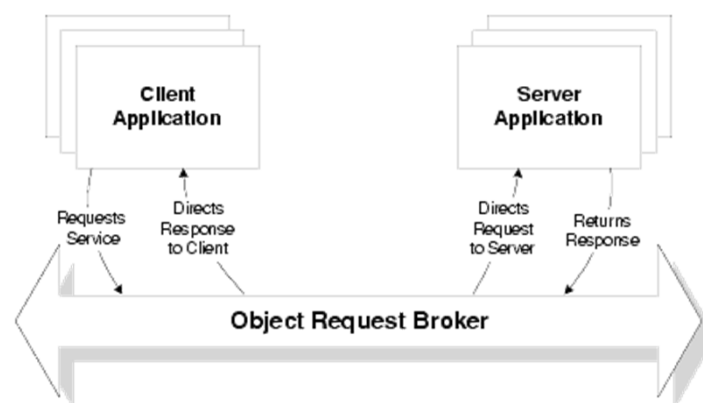
42

## CORBA: Architecture

- Object Request Broker (ORB)
  - The ORB allows clients to invoke operations in remote objects
  - Remote objects are called as CORBA objects but are essentially regular objects wrapped in CORBA IDL
  - Clients make a request of the ORB
  - The ORB is responsible for:
    - locating objects,
    - activating objects
    - and communicating client requests to objects

43

## CORBA: Architecture



44

## **CORBA: Architecture**

- The ORB provides:
  - Access transparency
    - Object methods are invoked in the same manner whether they are local or remote to the requesting object
  - Location transparency
    - The location of an object is not known and does not need to be known when invoking that object's methods
  - Implementation transparency
    - The way an application is implemented (language, Activation Policy, platform, operating system etc.) is hidden
  - Distribution transparency
    - The communications network, specific node - server mappings, and relative locations of communicating objects are hidden

45

## **CORBA: Architecture**

- Skeletons
  - RMI messages are sent via skeletons to appropriate server implementations (known in CORBA as servants)
  - Skeletons performs un-marshalling and marshalling duties
- Client stubs/proxies
  - Stubs for procedural languages
  - Proxies used in OO languages
  - Marshal client arguments of invocations

46

## CORBA: Architecture

- The Object Adaptor
  - Bridges the gap between CORBA IDL and programming languages
  - Its responsibilities:
    - Creating remote object references for CORBA objects
    - Dispatching each RMI via a skeleton
    - Activating the object

47

## CORBA: Architecture

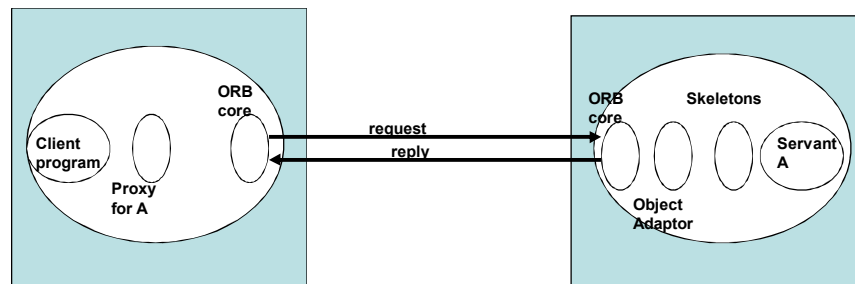
- Portable object adaptor (POA)
  - Separates the creation of CORBA objects from the creation of servants
  - An object adapter that can be used with multiple ORB implementations
  - Allows persistent objects - at least, from the client's perspective

48



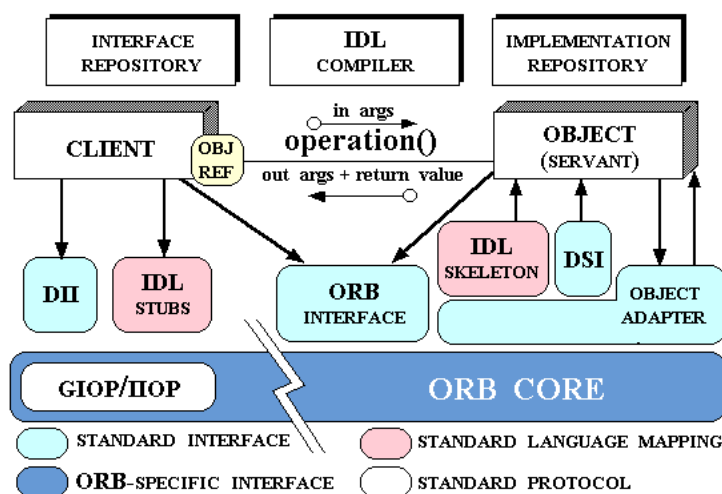
## CORBA: Architecture

### •The CORBA Architecture



49

## CORBA: Architecture



"Distributed Object Computing with CORBA," <http://www.cs.wustl.edu/~schmidt/corba-overview.html>

50

## **CORBA: Architecture**

- Interface repository
  - Provides similar facilities to Java reflection
  - Stores information about interfaces for client / server use
  
- Implementation repository
  - stores information about the object implementations
  - maps them to the names of object adaptors
  - Entries contain:
    - the object adaptor name
    - the pathname of the object implementation
    - the hostname and port number of the server

51

## **CORBA: Architecture**

- Dynamic implementation interface (DII)
  - Server interface's) may not have been available at client deployment
  - Clients can make requests without using stubs/proxies
  - Client discover services at run-time (using trading service)
  - ORB plays crucial role supplying interfaces and performing marshalling

52

## CORBA: Architecture

### Dynamic Skeleton Implementations (DSI)

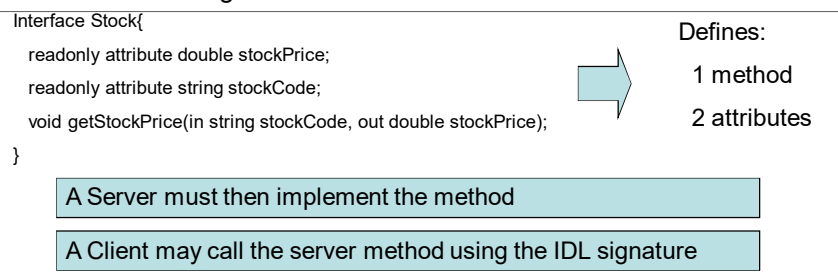
- The server side equivalent to DII
- A skeleton might not be present on the server in a development environment
- This facility allows requests for an object that does not have skeletons on the server
- A sever has to allow invocations on the interfaces of the CORBA objects
- Examine the contents of the invocation and discover the target object and method.

53

## CORBA: IDL

### •CORBA IDL

- Is used to define the remote methods for use by clients
- Servers define IDL for the methods that will use the ORB
- Server implements the IDL
- Clients program to the IDL specification
- Provides facilities for defining modules, interface types, attributes and method signatures



54

## IDL: more complex

```

1 exception NoCashException {
2   string reason;
3 };
4
5 interface Account {
6   readonly attribute double balance;
7   readonly attribute string owner;
8   void add(in double amount);
9   void remove(in double amount) raises (NoCashException);
10 };
11
12 module bank {
13   exception OpenException {
14     string reason;
15   };
16 };
17
18 interface Position {
19   void open(in string name,in double init) raises (OpenException);
20   double close(in ::Account account);
21   ::Account get(in string name);
22 };
23 };

```

55

## CORBA: Protocols

### •CORBA Protocols

- CORBA GIOP (General Inter ORB Protocol) handles external data using CDR
  - This provides hardware/software independence
- GIOP also handles R-R protocol for different operating systems
- IIOP implements R-R protocol of TCP-IP

56

## CORBA: remote references

### •IOR (Inter-Operable References)

–Contains:

- Interface repository identifier
- IIOP
- Host domain name
- Port number
- Adapter name
- Object name

57

## CORBA: remote references

IOR (Inter-Operable Reference)

IDL interface type id		Protocol & address details			Object key
Interface repository identifier	IIOP	Host domain name	Port number	Adaptor name	Object name

58

## CORBA: Services

•CORBA provides a set of services for use by distributed objects

•These include:

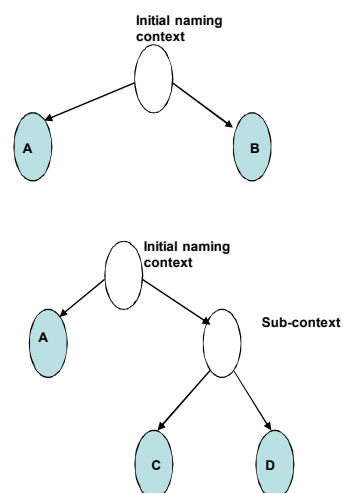
- Naming service
- Security service
- Trading service
- Transaction and concurrency service
- Persistent object service
- Event service

59

## CORBA: Services

•Naming service

- The CORBA naming service is essentially it is a binder
- Each ORB has an initial naming context
- Objects may bind to the initial naming context
- Context's may also exist within the initial naming context
- Objects may then bind to sub-context also
- Objects are referenced through their context and names



60

## CORBA: Services

- Security service

- Essentially this service authenticates principals
- Client sends credentials in requests
- Upon receipt the server decides whether to process has the right to access the remote object
- If sufficient rights exists the server execute the remote invocation
- Also ensures non-repudiation and secure communications

- Transaction services and concurrency control

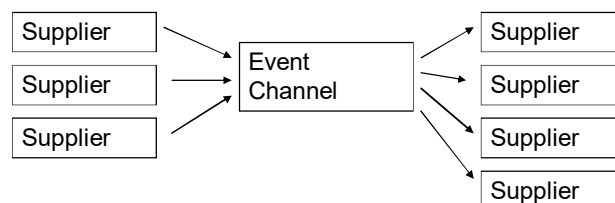
- Enforces concurrency control into RMI transactions
- Uses commit, rollback, and a two-phase commit cycle

61

## CORBA: Services

- Event Services

- This service defines a means for implementing the Publisher/Subscriber pattern (suppliers and consumers in CORBA)



- Push: suppliers push events to consumers
- Pull: consumers request events from suppliers

62

## **CORBA: Services**

- Persistent object service
  - Each CORBA object is responsible for saving its state when it is de-activated
  - The POS (Persistent Object Service) provides a means for CORBA objects to save their state
  - Can use PSDL or persistence transparency technique

63

## **CORBA: Services**

- Trading service
  - Rather than accessing objects by name trading services allow objects to be accessed by attributes
  - In other words, this service allows clients to access remote objects based on the which of the remote objects is best able to meet the request of the client rather than the client specifying a specific object
  - Example
    - If we want to find a plumber we can either use the name of the company
    - Or some criteria,
      - Local area='london', service='emergency'

64



## Microsoft DDE to DCOM

- History
  - DDE (Dynamic Data Exchange)
  - OLE (Object Linking and Embedding)
  - COM (Compound Object Model)
  - ActiveX
  - DCOM (Distributed Computing Environment DCE + COM)
- We are going to talk briefly about COM and DCOM

65

## COM

- Its interfaces are defined in IDL
- Supports dynamic discovery
- Uses Unique component ID mechanism
- Invocation from one component to another occurs using virtual tables, known in COM as vtables
  - Essentially this is just a list of pointers
  - clients have pointers to the *vtable* which point to server functions
  - This is all stored in binary so it is language independent

66

## COM: interfaces

- Interfaces in COM have Globally Unique Identifier (GUID)
- Its using the GUID that clients can locate required interfaces
  - vtable look-up
  - Vtable is a fast method!

67

## COM: Objects

- In COM objects have states and methods
- Contain code that implements a service
- Objects can be in different languages
- Access to objects occurs through defined interfaces
- COM supports multiple interfaces for objects

68

## COM: Objects

- A COM object must have the IUnknown interface
- This provides functions
  - QueryInterface, AddRef and Release
- But.. a COM object could have more interfaces but must have IUnknown

69

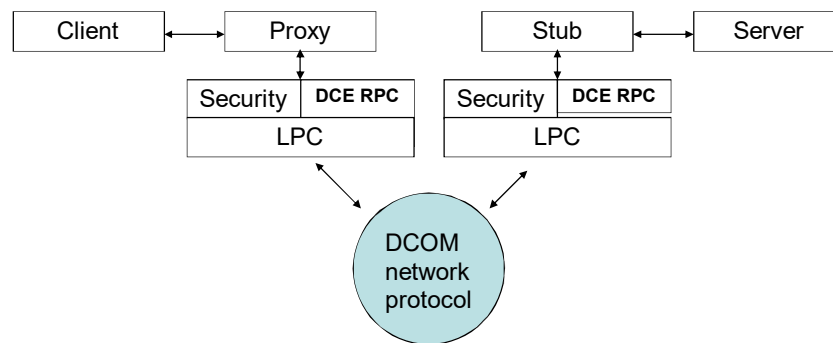
## DCOM

- Distributed COM (DCOM) extends COM for distributed systems allowing remote calls
- Based on Distributed Communication Exchange (DCE)
- Can use proxies and stubs
- Can also use the DCOM network protocol

70

# DCOM

- The DCOM intra-network model



71

# DCOM

- Within process communication
  - Fast, does not use proxies or stubs
- Inter-process communication
  - Slower, uses proxies and stubs
- Inter-network communication
  - Slowest, also uses the DCOM network protocol

72

## DCOM and CORBA

- Shared functionality with CORBA
  - Platform independence
  - Evolvable functionality
  - Interface reimplementation or extension
  - Interface reuse
  - Local/remote transparency
  - Remote handled by DCOM via RPC

73

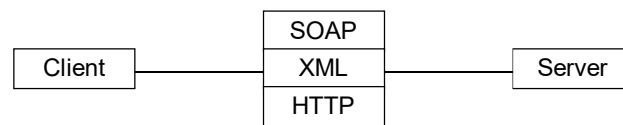
## DCOM and CORBA

- Unique features
  - Low overheads of object interaction
  - Negligible for in-process communication
  - Cross-process and cross-network is handled by DCOM

74

# SOAP

- SOAP (Simple Object Access Protocol)
- A scheme for using XML as RR messages
- Services are offered through web standards
- SOAP is built on XML and uses HTTP

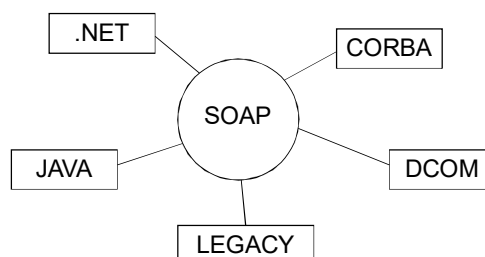


- Underneath that is TCP/IP and network protocols

75

# SOAP

- Interoperability through wrapping
- Wrap other technologies in SOAP messages



76

## SOAP

- Why SOAP?

- Direct connections between businesses is not allowed for security reasons
- Businesses use different middleware
  - Connecting them together using a global middleware is not realistic
  - And, middleware is often not compatible
- Publishing systems that require integration as web services using SOAP is a solution

77

## SOAP

- Essentially uses RPC through HTTP

1. Client transforms RPC into an XML document
2. Server transforms the XML into a procedure call
3. Server transforms the procedure's response into an XML document
4. Client transforms the XML document into the response to the RPC

- All uses XML to serialize arguments according to the SOAP specification

78

## SOAP

- SOAP is only a protocol it is not like CORBA or COM
- SOAP
  - Single function or method invocations
  - Stateless
  - clients make single method or function calls, one per HTTP request.
  - SOAP gets the data from a Web server in a method call, which uses the data offline locally, it then sends the updated data back to the server in another stateless remote call
- DCOM or CORBA
  - create a persistent connection
  - Multiple property accesses and method calls
  - DCOM and CORBA both support stateful remote connections

79

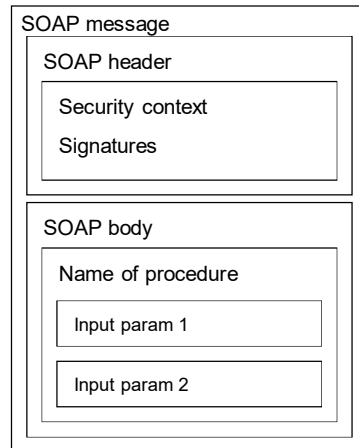
## SOAP

- SOAP is based on message exchanges
- Everything exists within an SOAP envelope
- Envelope contains Header and Body elements divided into blocks
- Body contains an XML document for a web-service
- Header used for coordination information, identifiers, security information

80

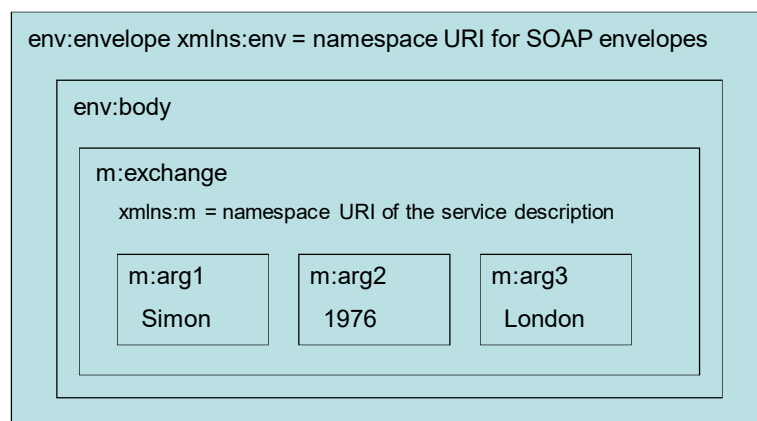


# SOAP



81

# SOAP



82

## SOAP

- *env* is the top element and is of type *envelope*
  - Envelope definition is defined using XML schemas and referenced through the xmlns (xml namespace)
- *body*
  - The body of the message
  - Again defined through an XML schema
- *m* is the message and is of type *exchange*
  - Exchange is defined through an XML schema for the service that it represents
  - Finally we have three arguments
    - These fit the definition of the service m

83

## SOAP

- SOAP binds to a transport protocol
- Typically HTTP using GET or POST
  - GET does not use a SOAP message but the response is a SOAP message
  - POST uses SOAP message for request and response
- Any other transport protocol could be used

84

## SOAP: request

```

POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "GetLastTradePrice"
<SOAP-ENV:Envelope xmlns:SOAP-ENV= "http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

From the: Simple Object Access Protocol (SOAP) 1.1. ©W3C Note 08 May 2000

85

## SOAP: request

```

POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePriceDetailed
      xmlns:m="Some-URI">
      <Symbol>DIS</Symbol>
      <Company>DIS Corp</Company>
      <Price>34.1</Price>
    </m:GetLastTradePriceDetailed>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

From the: Simple Object Access Protocol (SOAP) 1.1. ©W3C Note 08 May 2000

86

## SOAP: request

```

HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Header>
    <t:Transaction
      xmlns:t="some-URI"
      xsi:type="xsd:int" mustUnderstand="1">
      5
    </t:Transaction>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse
      xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

From the: Simple Object Access Protocol (SOAP) 1.1. ©W3C Note 08 May 2000

87

## Conclusions

- Middleware
- Interfaces (IDL)
- The Distributed Object Model
- RMI: Remote Method Invocation
  - An extension to local method invocation that allows remote method invocation also
- Middleware
  - CORBA
  - COM and DCOM
- SOAP

88

**END**