Web Technologies and Applications

Winter 2001

CMPUT 499: CORBA and SOAP

Dr. Osmar R. Zaïane

University of Alberta

Course Content

• Introduction	• Databases & WWW
• Internet and WWW	• SGML / XML
• Protocols	• Managing servers
• HTML and beyond	• Search Engines
• Animation & WWW	• Web Mining
• Java Script	• **CORBA & SOAP**
• Dynamic Pages	• Security Issues
• Perl Intro.	• Selected Topics
• Java Applets	• Projects

Objectives of Lecture 15

CORBA and SOAP

- Introduce protocols for exchanging data and procedures on the Internet.
- Understand the mechanism for remote procedure calls in an environment without pre-arranged inter-operations.
- Compare a simple protocol over HTTP (SOAP) with a more complex protocol over TCP/IP (CORBA)

Outline of Lecture 15

- Distributed Objects
- Remote Procedure calls with XML-RPC
- SOAP
- CORBA and IIOP

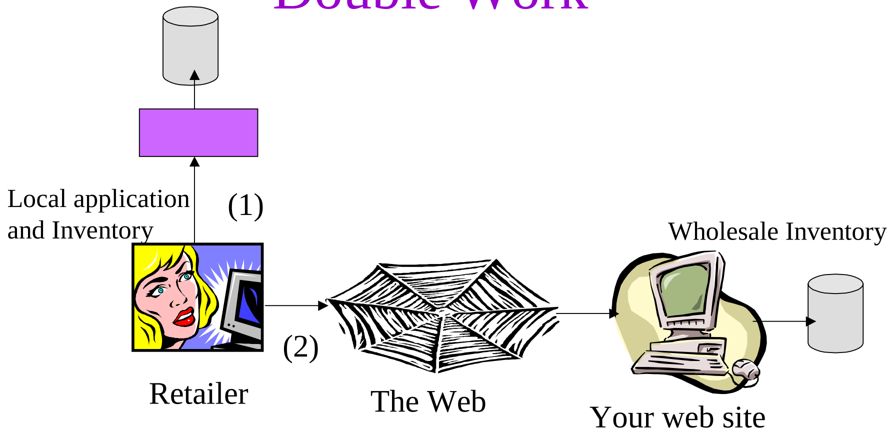
A Not So Fictive Case

- Imagine you are in the wholesale business. You are the middle man between the manufacturers and the retailers.
- Your company receives orders electronically.
- You want to provide real time information to business partners
- Retailers can check availability: whether you have specific items in stock, while suppliers can proactively ship you goods if your stock is low.

What are the solutions?

- You could build a web site that retailers can access to check availability of products.
- Advantages
  – We know how to build a web site
  – There are many tools to help develop such an e-commerce site.
- Disadvantages
  – Web site completely independent from retailers' inventory
  – Retailers would have to check manually the web site after already checking their own database.
  – Retailers need to be trained for this web site
  – On-line retailers may have to redirect their customers to the wholesale web site.

Double Work



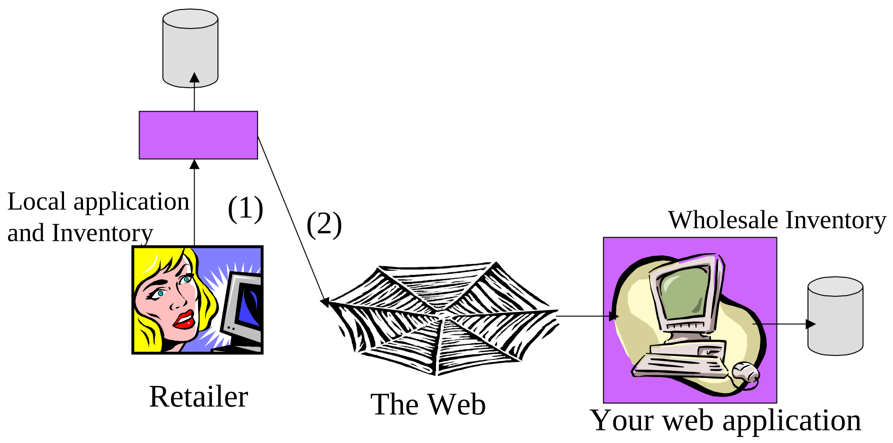
Distributed Objects

- What about an integrated solution?
- To integrate different applications running on the same or different computers, we use a middleware for distributing objects.
- There are many technologies: COM/DCOM from Microsoft, CORBA from OMG, RMI with Java, SOAP with XML, etc.

Different Technologies

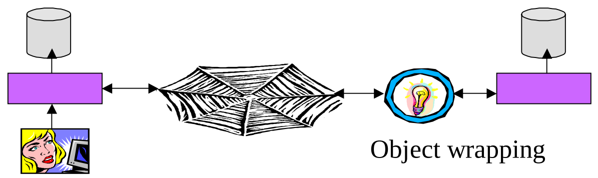
- COM, DCOM and COM+ (Distributed / Component Object Model) are mainly for Windows applications
- Java RMI (Remote Method Invocation) since Java 1.1 allows an object in a java program on one system to invoke the methods of another object in a java program across the network on a different system. http://www.java.sun.com/products/jdk/rmi/
- XMLTP (XML Transfer Protocol) http://xmltp.org/
- We will see SOAP (Simple Object Access Protocol) and CORBA (Common Object Request Broker Architecture).
- See lightweight protocols at http://www.lwprotocols.org/

Integration over the Internet



Concepts of Middleware

- Objects are sent from one application to the other via a middleware.
- The middleware wraps objects with a network layer
- Some technologies rely on TCP/IP, other on HTTP



Simplicity of HTTP servers

- Most corporate firewalls accept HTTP and SMTP traffic and bloc other communications
- There is an advantage in using HTTP as vehicle to distribute objects between web applications since it bypasses the firewalls
- Instead of HTML we can use XML which can efficiently transport structured information

Outline of Lecture 15

- Distributed Objects
- Remote Procedure calls with XML-RPC
- SOAP
- CORBA and IIOP

Remote Procedure Calls

- Beginning of 1980s the Remote Procedure Call (RPC) concept was introduced to make some programming features of a system transparent.
- With RPC, a client can call to request a server on a different machine (or the same machine) to carry out some actions. The network is made transparent to the programmer.
- To a large extent, calling RPC is as simple as calling any function, with extra network related error handling.

RPC Example

- When a machine A calls a procedure on machine B with an RPC, the calling process on A is suspended and the execution of the called procedure takes place on B.
- Ex: When writing a program to read n bytes from a file: *count= read(file,buffer,n)*, compiling and linking the program, the *read* routine in assembler is inserted into the object code. At run time the *read* issues a kernel trap and the file is read.

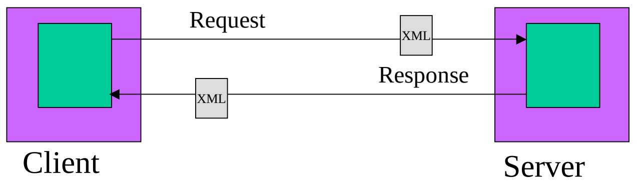
  With an RPC for a *read*, the program is linked to a version of the *read* called **client stub.** When the client stub is executed it packs (marshals) the parameters into a message and asks the kernel to send a message to the server. At the server-side a server stub de-marshals the message and executes the routine.

Marshaling = packing parameter values in a message

XML-RPC Protocol

- XML-RPC proposed by Userland Software
- Extremely simple. It uses XML as a vehicle to transmit and receive remote procedure calls.
- It defines the bare minimum to get RPCs across the network.
- The protocol is based on HTTP with the POST method. A request is an XML document containing a method name and parameters and the response is also an XML document with some returned values.

XML-RPC Big Picture



XML-RPC Header Requirements

- The header is and HTTP header
- The ***Content-Type*** is text/xml
- The ***Content-Length*** must be specified and correct
- A ***User-Agent*** must be specified
- A ***Host*** must be specified

XML Document Content (Payload)

- The XML document should have **<methodCall>** as a root.
- <methodCall> contains the name of the method invoked (**<methodName>**) and a list of parameter values (**<params>**)
- Each parameter is defined as a pair name-value with **<param>** and **<value>**

Data in Requests and Responses

- Data types supported are:
  - String (default)
  - Integer
  - Float
  - Dates
  - Binary
  - Boolean
- As well as composite data types such as:
  - Array
  - Struct
- Requests can have as many parameters as needed but responses have either one return value or an error code.However, composite types can express multiple values.

Scalar Values

• String	<string>	ASCII string All characters are allowed except <,> and & (&lt; &gt; &amp;)
• Integer	<int> or <i4>	4-byte signed integer
• Float	<double>	Double precision signed floating
• Boolean	<boolean>	0 (false) or 1 (true)
• Binary	<base64>	Base64-encoded binary ex: eW91IGNhbid0IHJ1YQWgd
• Date	<dateTime.iso8601>	Data/time ex: 20010328T11:32:55

If no type is specified, the type is string by default.

Composite Values with Structures

- We can have structures with the tag **<struct>**
- A structure contains members (**<member>**) and each member contains a name (**<name>**) and a value (**<value>**)

```
<struct>
        <member>
                <name>ProductName</name>
                <value>Shovel</value>
        </member>
        <member>
                <name>Price</name>
                <value><double>10.99</double></value>
        </member>
</struct>
```

- A structure can be recursive and contain a structured member with **<struct>** or **<array>** in **<value>**

Composite Values with Arrays

- We can have an array with the tag **<array>**
- A array contains a data element (**<data>**) which contains a set of values (**<value>**)

```
<array>
   <data>
         <value><string>cmput499</string></value>
         <value><int>19</int></value>
         <value><boolean>1</boolean></value>
         <value><double>89.56</double></value>
   </data>
</array>
```

- An array can be recursive and contain other arrays or structures, and values don't have to be of the same type

XML-RPC Request

```
POST /xmlrpcInterface HTTP/1.0
User-Agent: Yoyodyne XML-RPC Client 1.0
Host: xmlrpc.emailservice.com
Content-type: text/xml
Content-length: 195

<?xml version="1.0"?>
<methodCall>
        <methodName>email.getNumberNewMessages</methodName>
         <params>
                <param>
                <value><string>myUserName</string></value>
                </param>
         </params>
</methodCall>
```

email.getNumberNewMessages("myUserName");

Example from xml.com

Request Invocation with Java

```
XmlRpcClient xmlrpc=
        new XmlRpcClient ("http://xmlrpc.emailservice.com:80/xmlrpcInterface");
Vector params=new Vector();
params.addElement("myUserName");
Integer result=(Integer)xmlrpc.execute ("email.getNumberNewMessages", params);
```

Example from xml.com

XML-RPC Response

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 148
content-Type: text/xml
Date: Wed, Jul 28 1999 15:59:04 GMT
Server: Yoyodyne XML-RPC Server 1.0

<?xml version="1.0"?>
<methodResponse>
        <params>
                <param> <value><int>10</int></value> </param>
        </params>
</methodResponse>
```

There is always exactly only one <param> returned

Example from xml.com

Response in Case of Error

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 418
content-Type: text/xml
Date: Wed, Jul 28 1999 15:59:04 GMT
Server: Yoyodyne XML-RPC Server 1.0

<?xml version="1.0"?>
<methodResponse>
        <fault>
          <value>
            <struct>
              <member>
                <name>faultCode</name>
                <value><int>23</int></value>
              </member>
              <member>
                <name>faultString</name>
                <value><string>Unknown username</string></value>
              </member>
            </struct>
          </value>
        </fault>
</methodResponse>
```

Example from xml.com

Outline of Lecture 15

- Distributed Objects
- Remote Procedure calls with XML-RPC
- SOAP
- CORBA and IIOP

Simple Object Access Protocol

- The success of XML-RPC has lead to the design a new flexible lightweight protocol for exchange of information in a distributed environment. (SOAP)
- SOAP is also XML-based and uses the HTTP framework.
- The current version made available by W3C is 1.1. It is not yet endorsed by the W3C.
  http://www.w3.org/TR/2000/NOTE-SOAP-20000508
  Still open to discussions

SOAP and Firewalls

- Since SOAP is piggybacking HTTP, it is particularly effective at passing through firewalls.
- Firewalls prevent many RPC and interprocess communications protocols from reaching their destinations.
- Most corporate firewalls allow web browsing which allows SOAP to slip through transparently.

SOAP Request Header

- In addition to the usual HTTP header, the header for a SOAP request requires a SOAPMethodName entry
- The SOAPMethodName is the URI followed by the method name

```
SOAPMethodName: http://soapl.develop.com/cgi-bin/ServerDemo.pl?class=Geometry#calculateArea
```

The Payload Format

- A SOAP request is an XML document.
- It contains an envelope <**SOAP:Envelope**> which contains and optional header element <**SOAP:Header**> and a mandatory body element <**SOAP:Body**>. The order is important.
- The header is a mechanism for adding features to SOAP messages without prior agreement between communication parties. Often used for intermediaries on the message path.

Envelope and Types

<SOAP:Envelope> must include the following three standard namespace declarations:

```
xmlns:xsi="http://www.w3.org/1999/XMLSchema/instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema/instance"
xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1“
```

As we shall see, other declarations are also possible.

- SOAP supports all of the simple types and types derived from simple types defined in W3C XML Schema Primer. (http://www.w3.org/TR/xmlschema-0/#CreatDt)
- The data type is passed in the "xsd:type" attribute.
- NULL values are indicated with an attribute "xsi:null='true'".
- If a simple type does not include an "xsd:type" attribute, then it's up to the application to determine the data type based on the needs of the application.

Structure Types

- Structure types contains elements for each member of the structure. The names of the elements are the names of the members.

```
<corner>
  <x xsd:type="float">100</x>
  <y xsd:type="float">200</y>
</corner>
```

- Structures can contain simple types, structures, or arrays.

Array Types

- Array types use the attribute **“SOAP:arrayType=‘item[]‘”** to indicate that the contents of the array element is an array of <item> elements.

```
<stuff SOAP:arrayType='item[]'>
    <item xsd:type="integer">42</item>
    <item>
        <x xsd:type="float">100</x>
        <y xsd:type="float">200</x>
    </item>
    <item xsd:type="string">All is well.</item>
    <item xsi:null="true"/>
</stuff>
```

- Arrays can contain simple types, structures, or arrays, and they can be mixed as shown above.

SOAP 1.1 and Enumeration

- In SOAP 1.1 you can have enumerations: set of distinct names

```
<element name="EyeColor" type="tns:EyeColor"/>
<simpleType name="EyeColor" base="xsd:string">
        <enumeration value="Green"/>
        <enumeration value="Blue"/>
        <enumeration value="Brown"/>
</simpleType>

<Person>
    <Name>Henry Ford</Name>
    <Age>32</Age>
    <EyeColor>Brown</EyeColor>
 </Person>
```

Request Example

```
POST http://soapl.develop.com/cgi-bin/ServerDemo.pl?class=Geometry HTTP/1.0
SOAPMethodName: http://soapl.develop.com/cgi-bin/ServerDemo.pl?class=Geometry#calculateArea
Host: soapl.develop.com
Content-Type: text/xml
Content-Length: 494

<?xml version="1.0"?>
<SOAP:Envelope
    xmlns:xsi="http://www.w3.org/1999/XMLSchema/instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema/instance"
    xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
   <SOAP:Body>
         <calculateArea>
            <origin>
                  <x xsd:type="float">10</x>
                  <y xsd:type="float">20</y>
            </origin>
            <corner>
                  <x xsd:type="float">100</x>
                  <y xsd:type="float">200</y>
            </corner>
         </calculateArea>
   </SOAP:Body>
</SOAP:Envelope>
```

Example from lwprotocols.org

Response Example

```
HTTP/1.1 200 OK
Date: Thu, 20 Apr 2000 19:41:55 GMT
Server: Apache/1.3.9 (Unix) mod_perl/1.21
Content-Length: 326
Connection: close
Content-Type: text/xml

<SOAP:Envelope
    xmlns:xsi="http://www.w3.org/1999/XMLSchema/instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema/instance"
    xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
       <SOAP:Body>
          <calculateAreaResponse>
                <area xsd:type='float'>16200</area>
          </calculateAreaResponse>
       </SOAP:Body>
</SOAP:Envelope>
```

Example from lwprotocols.org

Fault Example

```
HTTP/1.1 200 OK
Date: Thu, 20 Apr 2000 19:57:15 GMT
Server: Apache/1.3.9 (Unix) mod_perl/1.21
Content-Length: 436
Connection: close
Content-Type: text/xml

<SOAP:Envelope
    xmlns:xsi="http://www.w3.org/1999/XMLSchema/instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema/instance"
    xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
   <SOAP:Body>
         <SOAP:Fault>
            <faultcode>300</faultcode>
            <faultstring>Invalid Request</faultstring>
            <runcode>No</runcode>
            <detail>This class doesn't support method rotate</detail>
         </SOAP:Fault>
   </SOAP:Body>
</SOAP:Envelope>
```

Example from lwprotocols.org

Example from W3C SOAP Notes

```
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>DEF</Symbol>
              <Company>DEF Corp</Company>
              <Price>34.1</Price>
           </m:GetLastTradePriceDetailed>
         </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

SOAP and Legacy Systems

- XML and SOAP are in many ways to integrate new systems with legacy systems.
- Creating a wrapper around legacy databases or legacy systems in general establishes a layer of indirection for system integration.
- The wrapper communicates with SOAP between the XML world and the legacy database.

Order Entry using SOAP

```
POST /LeeAnne HTTP/1.1
Host: www.leeanne.com
Content-Type: text/xml
Content-Length: 98
SOAPMethodName: http://www.leeanne.com/usingxml/OrderEntry

<SOAP:Envelope xmlns:SOAP="urn:SOAP:schemas-soap-org:soap.v1">
 <SOAP:Body>
  <oem:OrderEntry
    xmlns:oem="http://www.leeanne.com/usingxml/OrderEntry">
   <Title>Using XML 1.0</Title>
   <Author>Lee Anne Phillips</Author>
   <ISBN>0-7897-1996-7</ISBN>
  </oem:OrderEntry>
 </SOAP:Body>
</SOAP:Envelope>
```

A wrapper could translate this transaction and pass it to the legacy system.

This SOAP transaction could be answered by a confirmation with a shipping date or an out-of-order.

Order Confirmation using SOAP

```
HTTP/1.1 200 OK
connection: close
Content-Type: text/xml
Content-Length: 145
SOAPMethodName: http://www.leeanne.com/usingxml/OrderConfirm

<SOAP:Envelope xmlns:SOAP="urn:SOAP:schemas-soap-org:soap.v1">
 <SOAP:Body>
  <oem:OrderConfirm
    xmlns:oem="http://www.leeanne.com/usingxml/OrderConfirm">
   <Title>Using XML 1.0</Title>
   <Author>Lee Anne Phillips</Author>
   <ISBN>0-7897-1996-7</ISBN>
   <Confirm>12345678</Confirm>
   <ShipDate>20000711T163000-8</ShipDate>
  </oem:OrderConfirm>
 </SOAP:Body>
</SOAP:Envelope>
```

Using XSL, this SOAP transaction could be displayed on a web browser.

SOAP vs XML-RPC

- XML-RPC is simpler but less flexible.
- SOAP is more efficient in particular for requests.
- Many application already use XML-RPC since it came before SOAP. However many are migrating to SOAP.
- There are object libraries already written for SOAP (in Java). These make the implementation even simpler by hiding the protocol:
  - http://www.alphaworks.ibm.com
  - http://develop.com/SOAP/

Outline of Lecture 15

- Distributed Objects
- Remote Procedure calls with XML-RPC
- SOAP
- CORBA and IIOP

Heterogeneous World

- A corporate intranet might be made of mainframes, unix workstations, personal computers running a variety of OS flavors.
- The network protocols are as diverse: TCP/IP, Ethernet, Novell Netware, ATM…
- Heterogeneity is due to:
  - **Engineering tradeoffs**: different solutions across the enterprise
  - **Cost effectiveness**: best system at the lowest price in ≠ times
  - **Legacy systems**: systems too critical or too costly to replace
- Dealing with heterogeneity in distributed computing enterprise & develop open applications is very challenging

CORBA

- CORBA stands for Common Object Request Broker Architecture.
- It is defined and managed by the Object management Group (OMG)
- CORBA is known for Object Orientation, Interoperability, Heterogeneity and Transparent-Distribution.
- Not a product. It is a standard used to exchange data in a heterogeneous environment, large scale enterprise applications distributed on a network.

CORBA con’t

- CORBA makes it easier to implement new applications that must place components on different hosts on the network or use different programming languages.
- CORBA encourages the writing of open applications, applications that can be used as components of large systems, each application is made up of components and integration is supported by allowing other applications to communicate directly with these components.

CORBA con’t

- OO, Interoperability, and Distribution Out-of-the-box
- Interoperability across languages (Java, C/C++, Ada, Smalltalk, Common LISP, COBOL, etc.)
- Interoperability and Portability across Operating-Systems and Networks (CORBA is available on virtually every OS that you might want to use)
- Distribution / Location Transparency are Fundamental

Enterprise Computing

CORBA addresses 3 major difficulties in large scale enterprise level computing:

1. Allows each project & department to make independent decisions vis-à-vis OS and programming language to use.
2. Uses an O-O rather than procedural approach (such as RPC)
3. Allows the use of O-O design and implementation techniques at the enterprise level

O-O and CORBA

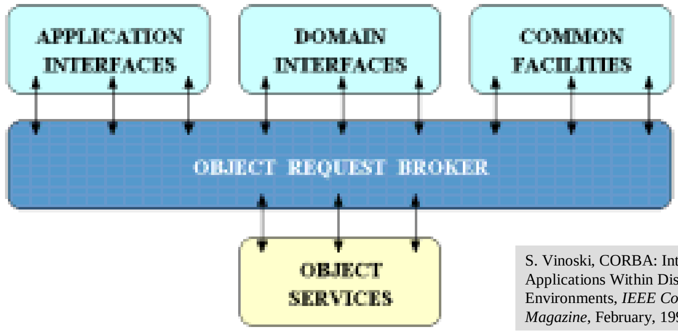
- CORBA enforces an overall Object-Oriented architecture even if components don’t have to be implemented in O-O programming language.
- While the CORBA O-O interface is visible, the internal implementation of the components is hidden

The big Picture

- Object Management Architecture (OMA) is composed of two parts:
  - **Object Model**: defines how distributed objects are described
  - **Reference Model**: describes how objects interact
- In the OMA Object Model an object is an encapsulated entity with a distinct immutable identity whose services can be accessed only through well-defined interfaces.
- Implementation and location are hidden to clients issuing requests

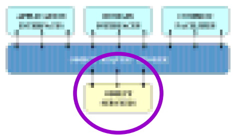
OMA Reference Model

- The Object Request Broker (ORB) facilitates communication between clients and objects.
- There are 4 interface categories



S. Vinoski, CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, *IEEE Communications Magazine*, February, 1997.

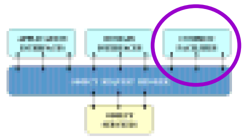
Object Services



- These are domain-independent interfaces
- Used by many distributed object programs
- Services providing for the discovery of other available services. Example:
  - **Naming service**: allows clients to find objects based on names
  - **Trading service**: allows clients to find objects based on properties
  - Event notification, security, lifecycle management, …

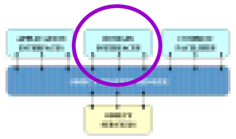
Always required

Common Facilities



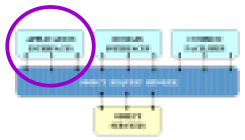
- Also domain-independent interfaces
- Oriented towards end-user applications
- Example an interface to distribute or link components of documents based on a document model
- DDCF (Distributed Document Component Facility) of Apple, IBM, Novell, etc.

Domain Interfaces



- Services oriented towards specific applications domains
- There are standards for domains such as manufacturing, telecommunications, medical domain, financial domain, etc.
- Separate application domains

Application Interfaces



- Interfaces developed specifically for a given application
- These interfaces are not standardized
- Some useful services could be standardized if they are broadly accepted and used ina specific domain ➔ becomes a domain interface

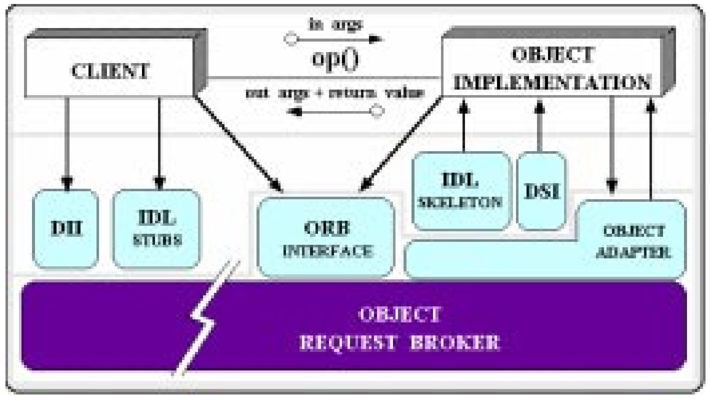
The Notion of Client-Server

- With CORBA there is not rigid notion of a client and a server; components communicate with others on a peer-to-peer basis
- Client and server are roles filled on a per-request basis
- A component can be a client and a server at the same time: client for other services and server for the services it provides

CORBA ORB Architecture

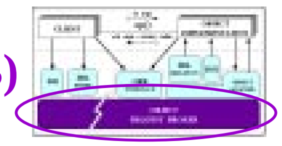
the program entity that invokes an operation on an object implementation

a CORBA programming entity that consists of an *identity*, an *interface*, and an *implementation*



S. Vinoski, CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, *IEEE Communications Magazine*, February, 1997.

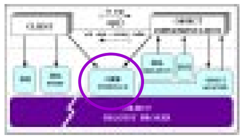
Object Request Broker (ORB)



The ORB provides a mechanism for transparently communicating client requests to target object implementations. The ORB simplifies distributed programming by decoupling the client from the details of the method invocations. This makes client requests appear to be local procedure calls. When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller.

Vinoski

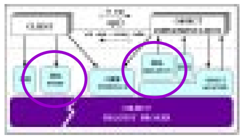
ORB Interface



An ORB is a logical entity that may be implemented in various ways (such as one or more processes or a set of libraries). To decouple applications from implementation details, the CORBA specification defines an abstract interface for an ORB. This interface provides various helper functions such as converting object references to strings and vice versa, and creating argument lists for requests made through the dynamic invocation interface described later.

Vinoski

CORBA IDL stubs & skeletons



CORBA IDL (Interface Definition Language) stubs and skeletons serve as the “glue” between the client and server applications, respectively, and the ORB. The transformation between CORBA IDL definitions and the target programming language is automated by a CORBA IDL compiler (Language Mappings). The use of a compiler reduces the potential for inconsistencies between client stubs and server skeletons and increases opportunities for automated compiler optimizations.

Vinoski

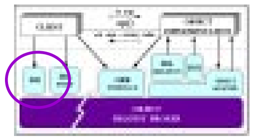
Interface Definition Language (IDL)

- Essentially a declarative language that defines types of objects by separating interfaces, named operations and parameters
- A means by which the object implementation tells clients what operations are available and how to invoke them.
- IDL is mapped to a particular language
- When compiled, it produces stubs and skeletons

Built-in Types and other Features

- Long (32 bit), long long (64 bits), short (16 bits) float, double, long double, char, octet, boolean, enum.
- Constructed types with *struct*
- Template types:
  - string (string<n>),
  - sequence (sequence<string,n>)
  - fixed (fixed<5,2>)
- Object references
- Interface Inheritance

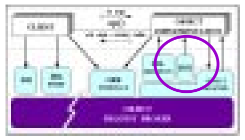
Dynamic Invocation Interface (DII)



This interface allows a client to directly access the underlying request mechanisms provided by an ORB. Applications use the DII to dynamically issue requests to objects without requiring IDL interface-specific stubs to be linked in. Unlike IDL stubs (which only allow RPC-style requests), the DII also allows clients to make non-blocking *deferred synchronous* (separate send and receive operations) and *oneway* (send-only) calls.

Vinoski

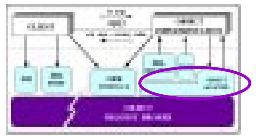
Dynamic Skeleton Interface (DSI)



This is the server side's analogue to the client side's DII. The DSI allows an ORB to deliver requests to an object implementation that does not have compile-time knowledge of the type of the object it is implementing. The client making the request has no idea whether the implementation is using the type-specific IDL skeletons or is using the dynamic skeletons.

Vinoski

Object Adapter



• This assists the ORB with delivering requests to the object and with activating the object. More importantly, an object adapter associates object implementations with the ORB. Object adapters can be specialized to provide support for certain object implementation styles (such as OODB object adapters for persistence and library object adapters for non-remote objects).

• An implementation must be registered with the OA.

• The OA maps requests to the appropriate implementation

Vinoski

Communication Scenario

- IDL compilation results in stubs(client-side) and skeletons (server-side)
- Client gets an object reference, and makes requests using it
- ORB translates the request into a form suitable for transmission
- OA assists ORB in determining the exact implementation
- IFR helps ORB do appropriate type checking. The IFR (Interface Repository) is a service that provides persistent objects that represent IDL information in a form available at run-time (provides type info to issue requests).
- ORB decodes request, passes it on to the skeleton
- Skeleton services request, stub and skeleton pass it back to the client (along with exception info, if any)

References

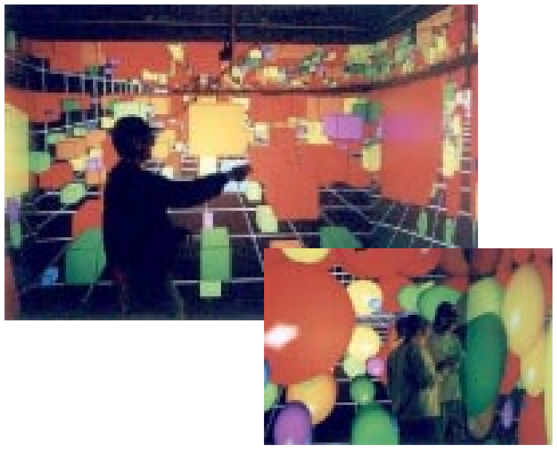
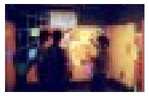
Steve Vinoski, CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, *IEEE Communications Magazine*, February, 1997.
http://www.cs.wustl.edu/~schmidt/PDF/vinoski.pdf

http://www.cs.wustl.edu/~schmidt/tutorials-corba.html

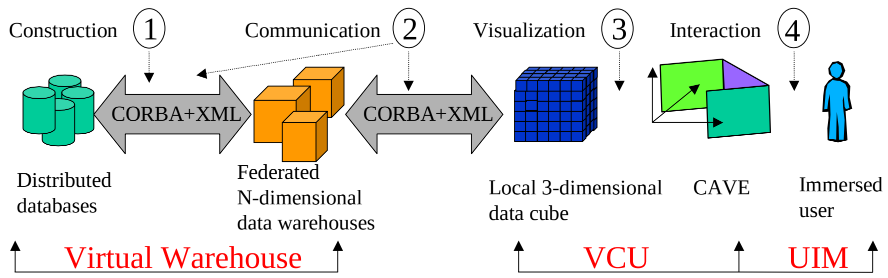
http://www.omg.org/

http://www.iona.com/

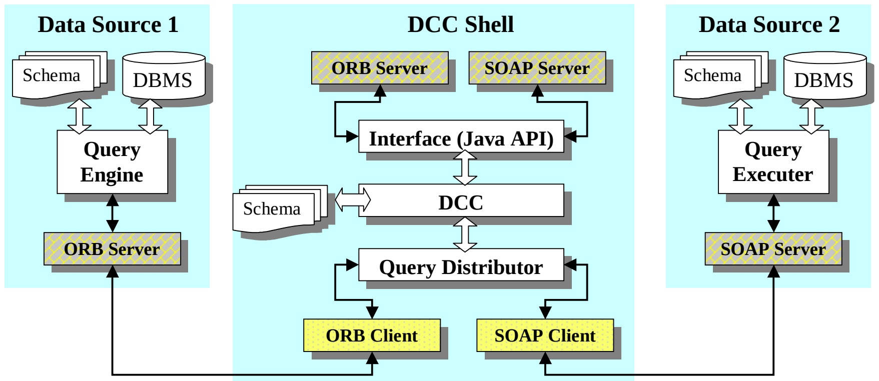
Example DIVE-ON Project

Example DIVE-ON Project



Example DIVE-ON Project


Data Source 1
Schema
DBMS
Query Engine
ORB Server
DCC Shell
ORB Server
SOAP Server
Interface (Java API)
Schema
DCC
Query Distributor
ORB Client
SOAP Client
Data Source 2
Schema
DBMS
Query Executer
SOAP Server