Basics of Distributed Object Systems

ECE 6102
Distributed Object Systems

- Extend object-oriented programming to distributed systems, i.e. allow objects to be distributed across multiple nodes
- Local process can operate on objects stored on other nodes in exactly the same way as operating on a local object (remote method invocation - RMI)
- Client/server model: local process is client, remote process hosting object is server
- Often allow interoperability of heterogeneous systems, e.g. different programming languages, different OSes
- Advantages: modularity, reusability, extensibility, interoperability
Examples of Distributed Object Systems

- **Java/RMI**
  - Java/RMI relies on a protocol called the **Java Remote Method Protocol**
  - Portable across operating systems
  - Requires Java Virtual Machine (JVM) implementation
  - Uses TCP/IP for communication

- **DCOM**
  - Supports remote objects by running on a protocol called the **Object Remote Procedure Call**
  - Is language independent
  - Requires a COM platform, i.e. Windows machine

- **CORBA**
  - Uses a protocol called **Internet Inter-ORB Protocol (IIOP)**
  - Platform independent and language independent
  - Well-suited for complex heterogeneous systems (e.g. DoD systems)
Distributed Object Systems vs. Other Technologies

- Web Services do not necessarily imply a distributed object system - think distributed systems built using HTTP and XML
- Simple Object Access Protocol (SOAP) was created to allow distributed object systems to be implemented via Web Services - details of object type and method invocation are encoded in an XML file
- .NET supports both basic Web Services and full distributed object systems (.NET was mainly intended to support basic Web Services but .NET remoting provides full distributed object support)
Distributed Object Systems: Summary

- Single language, multiple platforms: Java RMI
- Single platform, multiple languages: DCOM, .NET remoting
- Multiple platforms, multiple languages: CORBA, SOAP (supported by .NET to allow its components to work with components running on non-Windows platforms)
A SOAP message is an XML document containing:

- An Envelope element, which identifies the XML document as a SOAP message
- A Header element, which contains application-specific header information
- A Body element, which contains call and response information
- A Fault element, inside the Body, which contains errors and status information
Skeleton SOAP Message

<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
<soap:Fault>
...
</soap:Fault>
</soap:Body>
</soap:Envelope>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
  <m:GetPrice xmlns:m="http://www.w3schools.com/prices"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Item xsi:type="xsd:string">Apples</Item>
  </m:GetPrice>
</soap:Body>

</soap:Envelope>
SOAP Body - Response

<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
  <m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Price xsi:type="xsd:float">1.90</Price>
  </m:GetPriceResponse>
</soap:Body>

</soap:Envelope>
SOAP Operation

Client

C++ or Java App

SOAP Module

Server

C++ or Java App

Stateful Server w/ SOAP Module

HTTP
Common Features of Distributed Object Systems

- Interface Definition Language (IDL): language for describing object interfaces, i.e. what methods the object has and the number and types of parameters for each method (OMG IDL for CORBA or WSDL for Web Services)
- IDL allows clients and servers to execute remote method calls properly
- Dynamic invocation: ability for clients to discover object interfaces dynamically, allows clients to execute remote method calls without compile-time knowledge of an object interface
CORBA Features

- Object Request Broker (ORB) contains mechanisms to:
  - find object implementation for a given remote method invocation
  - pass invocation request to object implementation
  - return invocation result to client
  - support various other services
- Object reference is used by client to access remote object
- OMG Interface Definition Language (IDL)
  - Language-independent specification of object interfaces
  - IDL compilers convert IDL to specific language code, e.g. C++ or Java
  - IDL compilers generate stubs (client) and skeletons (server) that interact for proper execution of a remote method invocation
CORBA Features (continued)

- Stubs/skeletons: one stub or skeleton for each method of an object, skeleton simply calls appropriate method on the referenced object.
- Dynamic interfaces: dynamic invocation interface (DII) and dynamic skeleton interface (DSI)
  - one DII or DSI per process, each one can invoke any method on any object, DSI must be capable of invoking any method on server
- It is possible for a stub to call DSI and DII to call a skeleton (server chooses to have a separate skeleton for each method or one DSI for all methods, method calls access whichever type is available)
CORBA Features (continued)

- Remote method invocation example (stub/skeleton case):
  - Client invokes method on a remote object reference
  - Code for method call on client resides in stub
  - Stub passes method call arguments to ORB
  - ORB sends object reference and arguments to skeleton on server
  - Skeleton invokes method call and gathers outputs
  - Skeleton passes method call result and outputs to ORB
  - ORB sends result and outputs back to stub
  - Stub processes result and outputs and returns to client application
CORBA Features (continued)

- Interface repository stores object interface information (methods and parameters) that is accessible at run time.
- Dynamic object invocation
  - Client gets object interface information from interface repository at run time.
  - Client uses DII to invoke a method call on an object.
- General Inter-Orb Protocol (GIOP) specifies a standard transfer syntax and set of message formats for communication between ORBs.
- Internet Inter-Orb Protocol (IIOP) specifies how GIOP messages are exchanged using TCP/IP.
Implementation Repository

- Maintain a registry of known servers
- For each server, record which host it is running on and which port it uses
- Start servers on demand if they are registered with the Implementation Repository
- Periodically invoke a one-way method on the ping object to monitor the registered servers
ORB Implementation Types

- Client- and server-resident ORB: ORB is a separate process on each client/server system, stubs/skeletons interact with ORB through standard IPC mechanisms (one ORB per OS instance)
- Server-based ORB: ORB is a centralized server with which all clients and servers communicate (one ORB per distributed system)
- System-based ORB: ORB is part of the underlying OS on each machine (one ORB per OS instance)
- Library-based ORB: ORB is implemented within libraries that are linked into client and server processes (one ORB per client or server process - possibly multiple ORBs per OS instance)
Service Discovery

- UDDI (Universal Description, Discovery, and Integration) specifications for Web Services describe how to create service registries that can be queried automatically or browsed

- CORBA Trading Object Service supports lookup, registration and discovery of services based on service type