LECTURE 7: DISTRIBUTED OBJECT- & WEB-BASED SYSTEMS – PART 1
SECTION 7.1: DISTRIBUTED OBJECT-BASED SYSTEMS
Distributed Object-based Systems

• **Introduction**
  - Use object-oriented model to give *distribution transparency*.
  - Clients are offered services/resources as object interfaces with operations/methods that they can invoke.
  - *Distributed objects* claim ‘easy’ to hide distribution behind interface.
  - OO is a useful paradigm for building systems.
    - Objects encapsulate data (aka *state*), & operations on those data, (aka *methods*).
    - Methods are made available through an *interface*.
    - Can only access/change object’s state by invoking methods available in interface.
    - An object may implement multiple interfaces and for an interface definition, can be several objects offering an implementation of it.
    - Interface separates impln details from user or redirects to different implns.
Distributed Objects (/2)

**Architecture**

- Separation between interfaces & objects implementing them is crucial
  - Allows clients to see interface at one machine, with server on another machine.

⇒ Use an Interface Definition Language (IDL) to define interfaces, generate supporting code (stubs and skeletons)

⇒ Contract between Client + Server

---

**Organization of a Distributed Object with a Client-Side Proxy**

Client binds to dist’d obj ref => proxy/stub loads into client’s address space
Distributed Objects (/3)

• **Architecture**
  1. Data & operations *encapsulated* in an object,
  2. Operations implemented as methods grouped into interfaces
  3. Object offers only its *interface* to clients
  4. Server location is hidden from client (no distinction between remote+local)
  5. *Client stub* (proxy) implements interface, marshals call
  6. *Server skeleton* handles (de)marshalling and object invocation (+other stuff)

• **Types of objects I**
  • *Compile-time objects*: Language-level objects, from which proxy and skeletons are automatically generated.
  • *Runtime objects*: Implementable in any language, but need object adapter to make implementation appear as an object.

• **Types of objects II**
  • *Transient objects*: live only due to server: if server exits, so will the object.
  • *Persistent objects*: live independently of server: if server exits, object state & code remain (passively) on disk
Distributed Objects (/4)

**Example: Enterprise Java Beans (EJB)**

- **Def:** Java object hosted by special server that allows for different means of calling the object by remote clients.
- **EJB Server** splits application func from sys-oriented func (resp for obj storage/lookup..)
- **Four Different Types of EJBs**
  - **Stateless session bean:** Transient object, called once, does its work and is done.
    E.g.: execute SQL query, return result.
  - **Stateful session bean:** Transient object, but keeps client-related state until session end.
    E.g.: shopping cart.
  - **Entity bean:** Persistent, stateful object, can be invoked over many sessions.
    E.g.: object maintaining client info on last number of sessions.
  - **Message-driven bean:** Reactive objects, often triggered by message types. Used to implement publish/subscribe forms of communication.
Distributed Objects (/5)

- **Processes: Object servers**
  - **Servant**: Object implementation, sometimes only implements methods:
    - Collection of C or COBOL functions, that act on structs, records, DB tables, etc.
    - Java or C++ classes
  - **Skeleton**: Server-side stub handles n/w I/O:
    - De-marshalls incoming requests, calls relevant servant code
    - Marshalls results and sends reply message
    - Generated from interface specifications
  - **Object adapter**: “Manager” of a set of objects:
    1. Inspects (as first) incoming requests
    2. Ensures referenced object is ‘activated’ (requires identification of servant)
    3. Passes request to appropriate skeleton, following specific ‘activation’ policy
    4. Responsible for generating object references
Distributed Objects (/6)

- **Client-to-object binding:**

  - Object reference
    - Having an object reference allows a client to *bind* to an object:
    - Reference denotes server, object, and communication protocol vs Memory
    - Client loads associated stub code
    - Stub is instantiated and initialized for specific object

- Two ways of binding
  - *Implicit*: Methods are Invoked directly on referenced object
    ```
    Distr_object* obj_ref;
    obj_ref = ...;
    obj_ref→do_something();
    // Declare a systemwide object reference
    // Initialize the reference to a distrib. obj.
    // Implicitly bind and invoke a method
    ```
  - *Explicit*: Client must explicitly bind to object first before invoking it
    ```
    Distr_object obj_ref;
    Local_object* obj_ptr;
    obj_ref = ...;
    obj_ptr = bind(obj_ref);
    obj_ptr→do_something();
    // Declare a systemwide object reference
    // Declare a pointer to local objects
    // Initialize the reference to a distrib. obj.
    // Explicitly bind and get ptr to local proxy
    // Invoke a method on the local proxy
    ```

- Remote-object references allow us to pass references as parameters.
- This was difficult with ordinary RPCs.
Distributed Objects (/7)

• Remote Method Invocation (RMI)
  – Java Remote Method Invocation (RMI) system allows an object running in one JVM to call methods on objects running in another.
  – RMI gives applications *transparent, lightweight* access to *remote objects*.
  – RMI defines a high-level protocol and API.
  – Programming distributed applications in Java RMI is simple:
    • It is a single-language system.
    • Remote object coder must consider behaviour in a concurrent environment.

• Java RMI Applications
  – RMI is supported by two java packages `java.rmi` & `java.rmi.server`
  – An application that uses RMI has 3 components:
    • an *interface* that declares headers for remote methods;
    • a *server* class that implements the interface; and
    • one or more *clients* that call the remote methods.
Distributed Objects (/8)

• A Java RMI application needs to do the following:
  
  - **Locate remote objects**: An application can use one of two mechanisms to obtain references to remote objects:
    1. An application can register its remote objects with RMI's simple naming facility the `rmiregistry`, or
    2. The application can pass and return remote object references as part of its normal operation.
  
  - **Communicate with remote objects**:
    • Details of communication between remote objects are handled by RMI;
    • To coder, remote communication looks like standard Java method call.
  
  - **Load class bytecodes for objects that are passed around**:
    • RMI provides necessary mechanisms to load object’s code* & send its data.
    • Reason for this is that RMI allows caller to pass objects to remote objects.

  *i.e. object translated/’serialized’/’marshalled’ into bytecode
Distributed Objects (/9)

- **RMI Architecture**
  - **Stub**: lives client-side; pretends to be the remote object
  - **Skeleton**: lives on server; talks with true remote object
  - **Reference Layer**: determines if referenced object is local or remote
  - **Transport Layer**: - packages remote invocations;
    - dispatches messages between stub & skeleton
Distributed Objects (/10)

- Java RMI Basics: (Assumes client stub, server skeleton in place)
  - Client invokes method at *stub*
  - *Stub* marshals request and sends it to server
  - Server ensures referenced object is active:
    - Create separate process to hold object
    - Load the object into server process
    - ...
  - Object *skeleton* demarshalls request & referenced method is invoked
  - If request contains object reference, invocation is applied recursively (i.e., server acts as client)
  - Result is marshalled and passed back to client
  - Client *stub* demarshalls reply & passes result to client application
Distributed Objects (/10)

• RMI: Parameter passing
  • **Object reference**: Much easier than in the case of RPC:
    – Server can simply bind to referenced object, and invoke methods
    – Unbind when referenced object is no longer needed

• **Object-by-value**: Client may also pass a complete object as parameter value:
  – An object has to be marshalled:
    • Marshall its state
    • Marshall its methods, or give ref to where an implementation can be found
  – Server unmarshalls object (n.b. now have copy of original object)
  – **Object-by-value passing tends to introduce nasty problems**
Distributed Objects (/11)

• RMI Parameter Passing
  – Note: System-wide object reference usually contains:
    • Server address
    • Port to which adapter listens, and
    • Local object ID.
  – Extra: Info on protocol between client & server (TCP, UDP, SOAP, etc.)
Distributed Objects (/12)

• RMI Registry
  – A simple server-side bootstrap naming facility allowing remote clients to get a reference to a remote object
    • Servers name & register their objects to be accessed remotely with the RMI Registry.
    • Clients use the name to find server objects and obtain a remote reference to those objects from the RMI Registry.
  – Registry service is background program with a list of registered server names on a host and invoked by: `rmiregistry port` &
  – Registry service is provided by a Naming object providing two key methods:
    • **Bind**: to register a name and server
    • **Lookup**: to retrieve the server bound to a name
RMI Inheritance

```
Java.rmi.RemoteObject
    ^
Java.rmi.UnicastRemoteObject
   \    \extends
    \     MyServer

Java.rmi.Remote
    ^
MyServerInterface
   \    \implements
    
```

Lecture 7: Distributed Obj & Web-based Systems  CA4006 Lecture Notes (Martin Crane 2018)
Security Manager

- RMI programs must install a *security manager*
  - Otherwise RMI will not download classes

```java
if (System.getSecurityManager() == null) {
    System.setSecurityManager(new SecurityManager());
}
```

- Security policies specify actions that are *unsafe*
  - For every unsafe action there is a corresponding *checkXXX()* method
  - Actions not allowed throw a *SecurityException*

- Only one security manager can be installed
  - By default, an application has no security manager installed

- Policies are specified using *.*.policy files
  - Server and client application must specify their policy file
    - Default file: *java.home/lib/security/java.policy*
  - Use `-Djava.security.policy` property specify a file
Example: Simple Remote Database Access using RMI

Client

setSecurityManager
lookup(DB_name)
db_binding
read()

rmiregistry

databaseInterface_stub

read()
data

databaseInterface_skeleton

read()
data

read()
data

Database (Server)
import java.rmi.*;
import java.rmi.server.*;
public class Database extends UnicastRemoteObject implements DatabaseInterface {
    private int data = 0; // the database

    public Database(int value) throws RemoteException {
        data = value;
    }

    public int read () throws RemoteException {
        return data;
    }

    public void write (int value) throws RemoteException {
        data = value;
        System.out.println("New value is: " + data);
    }
}
import java.rmi.*;
import java.rmi.server.*;
public class DatabaseServer {

    public static void main (Strings[] args) {
        try {
            // create Database Server Object
            Database db = new Database(0);

            // register name and start serving
            String name = "rmi://fuji:9999/DB";
            Naming.bind(name, db);
            System.out.println (name + " is running");
        } catch (Exception ex) {
            System.err.println (ex);
        }
    }
}
import java.rmi.*;
public class DatabaseClient {
    public static void main (String[] args) {
        try {
            // set RMI Security Manager
            System.setSecurityManager(new RMISecurityManager() {
                public void checkConnect(String host, int port) {};
                public void checkConnect(String host, int port, Object Context) {};
            });
            // get database object
            String name = "rmi://fuji:9999/DB";
            DatabaseInterface db = (DatabaseInterface)Naming.lookup(name);
            int value, rounds = Integer.parseInt(args[0]);
            for (int i = 0; i < rounds; i++) {
                value = db.read();
                System.out.println("read: " + value);
                db.write(value+1);
            }
            } catch (Exception ex) {
                System.err.println (ex);
            }
        }
    }
}
RMI Example (/4): Building the Application

- Steps involved in Building the Application:

1. Compile the code:
   ```
javac Database.java DatabaseClient.java DatabaseInterface.java DatabaseServer.java
   ```

2. Generate stub and skeleton class files:
   ```
rmic Database
   ```
   (note: not needed for Java 5 or later)

3. Start the RMI registry (if don’t specify port, 1099 is the default):
   ```
rmiregistry 9999 &
   ```

4. Start the Server:
   ```
java -Djava.security.policy=java.policy DatabaseServer
   ```

5. Start the Client:
   ```
java -Djava.security.policy=java.policy DatabaseClient 10
   ```
SECTION 7.2: DISTRIBUTED WEB-BASED SYSTEMS
Introduction to Web Services

• WS offered by one electronic device to another, communicating via web

• Web technology (e.g. HTTP) was originally designed to support human comms (websites), is now used for M2M chatter, e.g. in XML and JSON.
  – HTTP defines message format, how sent and what Web servers & browsers do in turn – often just used as a transport

• Dichotomy/design forces:
  – CRUD (Create, Read, Update, Delete) vs Generic RPC
  – Enterprise vs scalable web
  – Client/UI vs M2M
  – Proprietary vs standards-based

• Main flavours
  – AJAX (Asynchronous JavaScript And XML) -> Javascript+JSON (client-oriented)
  – RESTful REST-compliant WS, where service changes forms of Web resources (URIs) using a uniform set of stateless operations (aka ‘CRUD’)
  – SOAP* typically using HTTP with XML serialization with other Web-related standards.
  – W3C Web Services Architecture – SOAP+WSDL interface defn language

*Simple Object Access Protocol, now largely falling out of use, though with some specialist applications
Background to Web Services

- Apache Web servers
  - **Observation**: More than 45% of 1.8 billion* Websites are Apache.
  - Server is internally organised roughly according to steps needed to process an HTTP request.
  - The anatomy of an Apache Web Server is shown below:

Background to Web Services (/2)

- **Server Clusters**
  - **Essence:** To improve performance & availability, WWW servers are often clustered in a way that is transparent to clients.
  - Below a server cluster is used with a front end to implement a WS.
Background to Web Services (/3)

• Problem with Server Clusters:
  – Front end gets easily overloaded, thus need for special measures.
    1. *Transport-layer switching*:
      – Front end simply passes TCP request to a server, according to some performance metric (e.g. load balancing).
    2. *Content-aware distribution*:
      – Front end reads the content of HTTP request and selects best server.

Role of a TCP Handoff in Server Clusters:

1. Switch gets a TCP connection request.
2. Finds best server & sends on request to that.
3. Server sends ‘ACK’ to client with switch’s IP address as source.
4. Must do this as Client was expecting to hear from switch.
Background to Web Services (/4)

• Naming: The Naming Service
  – Names play a very important role in all computer systems.
  – For sharing resources, uniquely identifying entities, referring to locations...
  – Important issue for naming:
    • a name must be resolvable to its entity it refers to,
    • for *Name resolution* need to implement a *Naming System*.
  – Naming in distributed systems & non-distributed systems differs in the implementation.
  – In Chord, DS naming system implementation is itself often distributed.
  – How this distribution is done dictates efficiency & scalability of the naming system.
Background to Web Services (/5)

- **Naming: Names in General**
  - *Name in DS*: string of bits/characters used to refer to it.
  - *Entities*
    - In DS can be anything (e.g. resources such as hosts, printers, disks & files).
    - Other examples of explicitly named entities are processes, users, mailboxes, Web pages, messages, network connections.
  - Entities can be operated on
    - e.g., a printer offers an interface with operations for printing docs & others
    - e.g. network connection offers data send/receive, set QoS parameters etc.
  - Operating on entities need an *Access Point*, another DS entity:
    - The name of an access point is called an *address*.
    - Address of entity’s access point entity is called an *address of that entity*.
  - Note: A *location-independent name* for an entity $E$, is independent from the addresses of the access points offered by $E$. 
Naming: Names in General (cont’d)

- Entities can offer more than one access point
  - e.g. phone is person’s access point, with phone number as address
  - people have many phone numbers, for their many addresses.
- In DS, a typical access point is a host running a specific server.
  - address is e.g. IP address+port (i.e. server's transport-level address).
- Entities may change access points over course time.
  - laptop moves location, it’s often assigned a different IP address
  - similarly, changing jobs or ISPs, means changing e-mail addresses.
• Naming: Identifiers
  – Pure name
    • A name that has no meaning at all; it is just a random string.
    • Pure names can be used for comparison only.
  – Identifier: A name having the following properties:
    • P1: Each identifier refers to at most one entity
    • P2: Each entity is referred to by at most one identifier
    • P3: An identifier always refers to the same entity (prohibits reusing an identifier)
  – Observation
    • Identifier needn’t necessarily be a pure name i.e. can have content
Background to Web Services (/8)

• Naming: Uniform Resource Locator (URL)
• Often contain information on how/where to access a document.
• Some URLs
  – Using only a DNS Name
  – Combining a DNS name with a port number
  – Combining a DNS name with a port number

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Host name</th>
<th>Port</th>
<th>Pathname</th>
</tr>
</thead>
<tbody>
<tr>
<td>http://</td>
<td><a href="http://www.cs.vu.nl">www.cs.vu.nl</a></td>
<td>80</td>
<td>/home/steen/mbox</td>
</tr>
</tbody>
</table>

(a)
From URLs to URIs and IRIs*

• A Uniform Resource Identifier (URI) is a compact sequence of characters that identifies an abstract or physical resource [RFC3986]

• Syntax:
  URI = scheme “:” hier-part [“?” query] [“#” fragment]

• Example
  foo://example.com:8042/over/there?name=ferret#nose

• Note: scheme not the same as protocol
  From: http://www.slideshare.net/mediasemanticweb/linked-data-michael-hausenblas-2009-03-05

*Internationalized Resource Identifier

© TCD
SECTION 7.2.1: SOAP-BASED WEB SERVICES
Web Services: SOAP-Based

• The Principle of a Web Service
  – Standardization dictates how those services are described such that they can be looked up by a client application.
  – Also, need to ensure that service call proceeds according to server application rules.
  – This is no different from what is needed to realize a remote procedure call.
Web Services: SOAP-Based (/2)

- **Standardization** needed so client can look up/access services.
  - Three Components:
    - **Directory Service**: Stores service descriptions.
      - Adheres to Universal Description, Discovery & Integration standard (UDDI).
      - As its name suggests, this prescribes DB layout with service descriptions.
      - Allows Web service clients to browse for relevant services.
    - **Interface**: Services described in Web Services Definition Lang (WSDL).
      - Formal language akin to IDLs used to support RPC-based communication.
      - Description contains precise definitions of interfaces provided by a service.
        - e.g. procedure specification, data types, (logical) location of services, etc.
      - A WSDL description is one that can be automatically translated to client-side and server-side stubs, akin to in ordinary RPC-based systems.
    - **Communication**: Simple Object Access Protocol (SOAP) is used
      - Specification of how communication takes place.
      - SOAP is used, which is essentially a framework for standardizing communication between two processes.
Web Services: SOAP-Based (/3)

• **Service-Oriented Architectures**

• So far, a Web service is offered in terms of a single invocation.
  • In practice, more complex invocation structures needed before a service can be considered as completed.
    e.g. book order requires selecting a book, paying, and ensuring its delivery.
  • So must model actual service as a transaction with multiple ordered steps.
  • Means dealing with a complex service built from number of basic services.

• **SOA principles for organising s/w not restricted to Web services use**
  • Loose Coupling (independent & self-contained)
  • Discoverability
  • Abstract service description (independent of implementation)
  • Encapsulation (autonomy and abstraction)
  • Compositionality (can be composed of other services)
  • Additional for web services: based on open standards & vendor neutral
Web Services: SOAP-Based (/4)

- **Java Web Services**: Java supports web services thro JAX-WS
  - **JAX-WS** = Java API for XML-Web Services.
  - Java Web Services can be deployed in the following ways:
    - Core Java only
    - Core Java with the current Metro release (helps when building a client)
    - Stand-alone web container (e.g. Tomcat)
    - Java application server (e.g. Glassfish – useful for implementing EJB)
  - Can implement SOAP-based web service as a single Java class
  - But usually consists of the following:
    - SEI (**Service Endpoint Interface**): Declares methods (web service operations)
    - SIB (**Service Implementation Bean**)
      - Defines the methods declared in the interface
      - Can be either **POJO** (Plain Old Java Object) or **EJB** (Enterprise Java Bean)
Web Services: SOAP-Based (/4)

• Writing a Web Service Client
  – Web service client is a program using Web service, e.g. Java application
  – How to access the Web services:
    • Send a **HTTP POST** request with request as SOAP message to server
    • Better: use `wsimport` to generate Java stubs to do this for you
  – However, `wsimport` needs a description of Web services offered by the Web server:
    • Use WSDL document generated by the Web server
    • URL of this document can be obtained by looking at Web services section at `http://localhost:4848`
SOAP Now

• XML no longer in vogue
• Complexity/verbosity of implementation tools
• JSON native support in Javascript
• REST ascendant
• Do we believe in contracts any more?