Today’s Topics

- External Data Representation and Marshalling (Sec. 4.3)
- Request reply protocol (failure modes) (Sec. 4.4)
- Distributed Objects and Remote Invocations (5.1)
  - Communication between Distributed Objects (5.2)
  - Remote Procedure calls (5.3)
  - Events and Notifications. (5.4)

All figures are in the book “Distributed Systems Concepts and Design” by Coulouris, Dollimore and Kindberg
## Middleware Layers

<table>
<thead>
<tr>
<th>Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications, services</td>
</tr>
<tr>
<td>RMI and RPC</td>
</tr>
<tr>
<td>Request-reply protocol, marshaling and external data representation</td>
</tr>
<tr>
<td>Network</td>
</tr>
</tbody>
</table>

- **Middleware** is the two layers in the middle.
External Data Representation and Marshaling

Remember the problem in lecture one,

- **Heterogeneity.** (Everybody is different).
  - Different operation systems.
  - Different programming languages.
  - Different hardware architectures.
- Problem when data structures must be sent in message in distributed system.
Example, Big Endian/Little Endian

Most numbers that you store are larger than a byte and there is some choice about how you lay out multibyte numbers in memory.

- **Big Endian** in big endian you store the most significant byte in the smallest memory address. (For example DECs)
- **Little Endian** in little endian you store the least significant byte in the smallest memory address. (For example Suns, Motorolas).
- For example to store the number **0x90AB**

<table>
<thead>
<tr>
<th>Address</th>
<th>Big Endin Value</th>
<th>Little Endin Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>90</td>
<td>AB</td>
</tr>
<tr>
<td>1001</td>
<td>AB</td>
<td>90</td>
</tr>
</tbody>
</table>
Example, Representing characters

- There are different set of codes to represent characters.
  - ASCII: 1 byte
  - Unicode: 2 byte
What to do?

One of the two following methods can be used:

- The values can be converted to an agreed external format before transmission and converted to the local form on receipt.
- The values are transmitted in the sender’s format together with an indication of the format used, and the recipient converts the values if necessary.

An agreed standard for the representation of data structures and primitive values is called an *external data representation*. 
Marshalling - Unmarshalling

- **Marshalling** is the process of taking a collection of data items and assembling them into a form suitable for transmission.

- **Unmarshalling** is the process of disassembling them on arrival to produce an equivalent collection of data items at the destination.
Two Examples

We will briefly discuss two alternative approaches to external data representation and marshalling.

- **CORBA’s common data representation (CDR)**, an external representation of data.

- **Java’s object serialisation** which flattens objects or tree of objects.

Needs no involvement of the application programmer.
CORBA

If middleware is supposed support many different systems and you have to convert the data then how do you get that to work?

- CORBA has a common external format CDR.
- CDR can represent all the data types that can be used as arguments and return values in remote invocations.
- Values transmitted in serders order (big-/little-endian).
CORBA Interface Definition Language

- The interface of objects are described in CORBA IDL.
- From a CORBA IDL specification the marshalling & unmarshalling operations are generated by the CORBA interface compiler.
- For each new platform a CORBA IDL compiler has to be provided.
Java Object Serialization

- In Java RMI, both objects and primitive data values may be passed as arguments and results of method invocations.
- If an object implements the interface `Serializable` then the system provides a default serialisation method.
- *Serialization* refers to the activity of flattening an object into a serial form.
Java Object Serialization (cont.)

- The contents of all the data members of the class is serialized.
- Serialization and deserialization of the arguments and results of remote invocations are generally carried out automatically by the middleware.
- A programmer declares a variable as `transient` when it should not be serialized.
Request Reply Protocol

- Communication designed for client-server.
- Common format for protocols. Send the request to the remote server/object and wait for the reply of acknowledgement.
- The primitives:
  - `doOperation` - specify the remote object and which method to invoke.
  - `getRequest` - acquire service requests.
  - `sendReply` - send reply message to the client.
Request-reply communication

Client

- doOperation
- (wait)
- (continuation)

Server

- getRequest
- select object
- execute method
- sendReply

Request message

Reply message
Failure in Request Reply Protocol

- If primitives implemented over UDP datagrams:
  - lost;
  - might not be received in the same order that they were sent.
- If client does not know if request has been executed by server.
  - Messages can be lost.
  - Server needs more time than clients timeout.
- Client can resend message until a reply is received or sure that server does not respond.
Distributed Objects and Remote Invocation

Middleware layers:

<table>
<thead>
<tr>
<th>Applications, services</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMI and RPC</td>
</tr>
<tr>
<td>Request-reply protocol, marshaling and external data representation</td>
</tr>
<tr>
<td>Network</td>
</tr>
</tbody>
</table>
Distributed Objects and Remote Invocation

We will look at four things:

- The distributed object model.
- Remote method invocation (RMI).
- Remote procedure call (RPC).
- Distributed event-based systems.
The distributed object model

- Remote objects are objects that can receive remote invocations.
- Objects can invoke methods of a remote object if they have access to its remote object reference.
- Every remote object has a remote interface that specifies which of its methods can be invoked remotely.

Remote interfaces:

- Remote interfaces do not have constructors.
- In CORBA user defines the remote interfaces in IDL.
- In Java RMI user extends interface Remote.
Middleware - Transparency

- Remember, one of the goals of middleware is to make things as invisible to the application.
- One of the design goals is location transparency.
- Remote invocations is vulnerable to failure.
- Remote invocations can throw exceptions which a client program handle.
Implementation of RMI

- **Communication module**: One in the client and one in the server. Perform the request-reply protocol.

- **Remote reference module**: Translates between local and remote object references and creates remote object references.
  - For each process it maintains a *remote object table* - the mapping between local object references and remote object references.

- **Proxy**: Acts like a local object to the invoker, but instead of executing the invocation it forwards it in a message to a remote object. Does the marshalling and unmarshalling.
Implementation of RMI (cont.)

- **Dispatcher:** The dispatcher selects the method to invoke in the skeleton.

- **Skeleton:** A skeleton method unmarshals the arguments in the request messages and invokes the corresponding method in the object.

- An interface compiler generates proxies, dispatchers and skeletons automatiskt.
The role of proxy and skeleton in remote method invocation

Client

Object A proxy for B

Request

Reply

Server

Skeleton

Dispatcher for B's class

Remote object B

Communication module

Remote reference module

Communication module

Remote reference module
Remote Procedure Calls

- **Client:**
  - **Stub procedure:** acts like a local procedure, marshals the request and unmarshals the reply. Similar to proxy in RMI.

- **Server:**
  - **Dispatcher:** calls server’s stub procedure.
  - **Stub procedure:** Calls the “real service procedure”.
    Unmarshals the request and marshals the reply. Similar to skeleton in RMI.

Like for RMI the software for RPC can be generated by an interface compiler.
Role of client and server stub procedures in RPC in the context of a procedural language
Problems with local references

- If you pass a parameter which is a reference (pointer) to a local piece of data, then what do you do? Possible solutions include:
  - Don’t pass pointers!
  - Copy the local data and send it over the network.
  - Modify the server so that it requests the data on demand.
Events and Notifications

• The idea: one object can react to a change occurring in another object.

• Publish-subscribe paradigm. Objects publish a list of types of events that it might generate.

• Other object subscribe to certain events. When the objects generate events they notify the objects.

• Often used in GUI toolkits, for example a button on the screen generates an even that could be monitored by different objects.
Distributed event-based systems characteristics

- **Heterogeneous**: Components that were not designed to interoperate can be made to work together.

- **Asynchronous**: Notifications are sent asynchronously to all the objects that have subscribed to them.