Remote Procedure Calls (RPCs) and Remote Method Invocation (RMI)
Process communication

Message passing

socket.write("DEPOSIT Alice $20\n")

Remote procedure calls (RPC)s

deposit("Alice", 20)

Remote method invocation

aliceAccount.deposit(20)
Example: Python xmlrpc Server and Client

```python
from xmlrpc.server import SimpleXMLRPCServer

def is_even(n):
    return n % 2 == 0

server = SimpleXMLRPCServer(("localhost", 8000))
print("Listening on port 8000...")
server.register_function(is_even, "is_even")
server.serve_forever()

import xmlrpc.client

with xmlrpc.client.ServerProxy("http://localhost:8000/") as proxy:
    print("3 is even: %s" % str(proxy.is_even(3)))
    print("100 is even: %s" % str(proxy.is_even(100)))
```
Example: Python RPC with Thrift

```python
# Make socket
transport = TSocket.TSocket('localhost', 9090)

# Buffering is critical. Raw sockets are very slow
transport = TTransport.TBufferedTransport(transport)

# Wrap in a protocol
protocol = TBinaryProtocol.TBinaryProtocol(transport)

# Create a client to use the protocol encoder
client = MultiplicationService.Client(protocol)

# Connect!
transport.open()

RPC

product = client.multiply(4,5)
print '4*5=%d' % (product)

# Close!
transport.close()
```
Example: Go RPC client using rpc

```go
var {
    addr = "127.0.0.1:" + strconv.Itoa(Port)
    request = &core.Request{Name: Request}
    response = new(core.Response)
}

// Establish the connection to the address of the
// RPC server
client, _ = rpc.Dial("tcp", addr)
def er c.client.Close()

// Perform a procedure call (core.HandlerName == Handler.Execute)
// with the Request as specified and a pointer to a response
// to have our response back.
_ = c.client.Call(core.HandlerName, request, response)
fmt.Println(response.Message)
```
Example: Java RMI

```java
import com.mkyong.rmiinterface.RMIInterface;

public class ClientOperation {

    private static RMIInterface look_up;

    public static void main(String[] args)
        throws MalformedURLException, RemoteException, NotBoundException {

        look_up = (RMIInterface) Naming.lookup("//localhost/MyServer");
        String txt = JOptionPane.showInputDialog("What is your name?");

        String response = look_up.helloTo(txt);
        JOptionPane.showMessageDialog(null, response);
    }
}
```
Client Steps

1. Create a transport connection to server
2. [Look up name of RPC] and get proxy handle
3. Call proxy with arguments
4. Process results
5. [Deal with errors]

```python
import xmlrpc.client

with xmlrpc.client.ServerProxy("http://localhost:8000/") as proxy:
    print("3 is even: \$s" % str(proxy.is_even(3)))
    print("100 is even: \$s" % str(proxy.is_even(100)))
```
Proxy

Provides *transparency* by behaving like a local function/method to the invoker

- The proxy “implements” the same interface

Instead of executing an invocation, the proxy forwards it to a remote

- Marshals a request message
  - Target object reference
  - Method ID
  - Argument values
- Sends request message
- Unmarshals reply and returns to invoker
Proxy steps

1. Send function name
2. Marshall arguments
3. Wait for response
4. Deal with errors
5. Unmarshall results
6. Return

```python
def send_add(self, num1, num2):
    self._oprot.writeMessageBegin('add', TMessageType.CALL, self._seqid)
    args = add_args()
    args.num1 = num1
    args.num2 = num2
    args.write(self._oprot)
    self._oprot.writeMessageEnd()
    self._oprot.trans.flush()

def recv_add(self):
    iprot = self._iprot
    (fname, mtype, rseqid) = iprot.readMessageBegin()
    if mtype == TMessageType.EXCEPTION:
        x = TApplicationException()
        x.read(iprot)
        iprot.readMessageEnd()
        raise x
    result = add_result()
    result.read(iprot)
    iprot.readMessageEnd()
    if result.success is not None:
        return result.success
    raise TApplicationException(TApplicationException.MISSING_RESULT,
                                "add failed: unknown result")
```
Marshalling & Unmarshalling

External data representation: an agreed, platform-independent, standard for the representation of data structures and primitive values.

- CORBA Common Data Representation (CDR)
- Sun’s XDR
- Google Protocol Buffers
- Language-based serialization

Marshalling: taking a collection of data items (platform dependent) and assembling them into the external data representation (platform independent).

Unmarshalling: the process of disassembling data that is in external data representation form, into a locally interpretable form.
Example: Google Protocol Buffers

message Test1 {
  required int32 a = 1;
}

message Test2 {
  required string b = 2;
}
Example: JSON-RPC

REQUEST

```json
{
    "jsonrpc": "2.0",
    "method": "subtract",
    "params": [42, 23],
    "id": 1
}
```

RESPONSE

```json
{
    "jsonrpc": "2.0",
    "result": 19,
    "id": 1
}
```
Server Side

Dispatcher is the front end processing all incoming requests and directing them to the appropriate skeleton implementation based on name.

Skeleton:
1. Reads and unmarshalls arguments
2. Calls real implementation
3. Marshalls and writes results

```python
def process(self, iprot, oprot):
    (name, type, seqid) = iprot.readMessageBegin()
    if name not in self._processMap:
        iprot.skip(TType.STRUCT)
        iprot.readMessageEnd()
        x = TApplicationException(TApplicationException.UNKNOWN_METHOD,
            'Unknown function %s' % (name))
        oprot.writeMessageBegin(name, TMessageType.EXCEPTION, seqid)
        x.write(oprot)
        oprot.writeMessageEnd()
        oprot.trans.flush()
        return
    else:
        self._processMap[name](self, seqid, iprot, oprot)
    return True
```
Server Side

**Dispatcher** is the front end processing all incoming requests and directing them to the appropriate **skeleton** implementation based on **name**

**Skeleton:**
1. Reads and unmarshalls arguments
2. Calls **real implementation**
3. Marshalls and writes results

```python
def process_add(self, seqid, iprot, oprot):
    args = add_args()
    args.read(iprot)
    iprot.readMessageEnd()
    result = add_result()
    try:
        result.success = self._handler.add(args.num1, args.num2)
        msg_type = TMessageType.REPLY
        except TTransport.TTransportException:
            raise
        except TApplicationException as ex:
            logging.exception('TApplication exception in handler')
            msg_type = TMessageType.EXCEPTION
            result = ex
        except Exception:
            logging.exception('Unexpected exception in handler')
            msg_type = TMessageType.EXCEPTION
            result = TApplicationException(TApplicationException.INTERNAL_ERROR,
                                            'Internal error')
        oprot.writeMessageBegin("add", msg_type, seqid)
        result.write(oprot)
        oprot.writeMessageEnd()
        oprot.trans.flush()
```
Failure Modes of RMI/RPC

- Correct function
  - Execute
  - Reply

- Crash before reply
  - Execute, Crash
  - Request

- Crash before execution
  - Crash
  - Request

- Lost request
  - Execute
  - Reply

- Channel fails during reply
  - Execute

- Client machine fails before receiving reply
  - Execute
  - Reply
What to do if error?

Raise exception to application?

Retry RPC?

Semantics:
- Exactly once (desired)
- At least once
- At most once
Idempotent Operations

Idempotent operations are those that can be repeated multiple times, without any side effects.

Examples (x is server-side variable)
- x=1;
- x=(argument) y;

Non-examples
- x=x+1;
- x=x*2

Idempotent operations can be used with at-least-once semantics.
Distributed Objects and RMI

Process

Host A

remote invocation

A

remote obj

Object

Process

Process

local invocation

C

E

remote invocation

F

remote obj

Host B

aliceAccount.transferFrom(tlabAccount, $100)
Remote Reference Module

Translates local and remote object references

Response:

- {  
  - “postID”: 1234,  
  - “contents”: “What is on the midterm”,  
  - “response”: {  
    - “objType”: “responseObject”,  
    - “objRef”: “12345”  
  }  
}
Remote Reference Module

Remote object table
- An entry for each remote object held by any process. E.g., B at P2.
- An entry for each local proxy. E.g., proxy-B at P1.

RRM looks up remote object references inside request and reply messages in table
- If reference not in table, create a new proxy and add it to the table
- Then (in either case), replace reference by proxy found in table
Proxy object is a hollow container of Method names.

Remote Reference Module translates between local and remote object references.

Dispatcher sends the request to Skeleton Object

Skeleton unmarshals parameters, sends it to the object, & marshals the results for return
Generation of Proxies, Dispatchers and Skeletons

Programmer only writes object **implementations** and **interfaces**
- E.g., CORBA: programmer specifies interface in CORBA IDL
- E.g., Java RMI: programmer defines set of remote object methods as a Java interface

**Proxies, dispatchers, skeletons** generated automatically from the specified **interfaces**
- Compiler to generate code (e.g., Thrift)
- Reflection to do this automatically