Remote Procedure Calls & Distributed Objects

CS425 /ECE428 – DISTRIBUTED SYSTEMS – SPRING 2019

Material derived from slides by I. Gupta, M. Harandi, J. Hou, S. Mitra, K. Nahrstedt, N. Vaidya
Bank Database Example

Bank Database: Think of two simultaneous deposits of $10,000 into your bank account, each from one ATM.

- Both ATMs read initial amount of $1000 concurrently from the bank server
- Both ATMs add $10,000 to this amount (locally at the ATM)
- Both write the final amount to the server
- What's wrong?

The ATMs need mutually exclusive access to your account entry at the server

How are “transactions” executed between a client ATM and a bank server?
Middleware Layers

Applications

RPCs and RMIs, e.g., CORBA

Request reply protocol

External data representation

Operating System

Middleware layers = Provide support to the application

Run at all servers @user level

RMI = Remote Method Invocation
CORBA = Common Object Request Brokerage Architecture
Local Objects

Within one process’s address space

Object
- consists of a set of data and a set of methods.
- E.g., C++/Java object

Object reference
- an identifier via which objects can be accessed.
- i.e., a pointer (C++)

Interface
- Signatures of methods
  - Types of arguments, return values, exceptions
  - No implementation
- E.g., hash table:
  - insert(key, value)
  - value = get(key)
  - remove(key)
Remote Objects

May cross multiple process’s address spaces

Remote method invocation
- method invocations between objects in different processes (processes may be on the same or different host).
- *Remote Procedure Call (RPC):* procedure call between functions on different processes in non-object-based system

Remote objects
- objects that can receive remote invocations.

Remote object reference
- an identifier that can be used globally throughout a distributed system to refer to a particular unique remote object.

Remote interface
- Every remote object has a remote interface that specifies which of its methods can be invoked remotely. E.g., CORBA interface definition language (IDL).
A Remote Object and Its Remote Interface

Example Remote Object reference=(IP,port,objectnumber,signature,time)
Local invocation = between objects on same process.
Has \textit{exactly once} semantics

Remote invocation = between objects on different processes.
Ideally also want \textit{exactly once} semantics for remote invocations
But difficult (why?)
Failure Modes of RMI/

- Execute
  - Correct function
  - Crash before reply
  - Crash before execution
  - Reply

- Request
  - Lost request
  - Channel fails during reply
  - Client machine fails before receiving reply

execute function

NIKITA BORISOV - UIUC
## Invocation Semantics

### Fault tolerance measures

<table>
<thead>
<tr>
<th>Retransmit request message</th>
<th>Duplicate filtering</th>
<th>Re-execute procedure or retransmit reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Re-execute procedure</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Retransmit old reply</td>
</tr>
</tbody>
</table>

- **CORBA**: No, Not applicable, Not applicable, Maybe (ok for idempotent operations)
- **Sun RPC**: Yes, No, Re-execute procedure, At-least-once
- **Java RMI, CORBA**: Yes, Yes, Re-transmit old reply, At-most-once

Idempotent = same result if applied repeatedly, w/o side effects
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.
Proxy and Skeleton in Remote Method Invocation

Diagram:
- Client: Object A, proxy for B
- Server: Remote object B, skeleton & dispatcher for B's class
- Communication module
- Remote reference module
- Process P1
- Process P2
- Middleware
Proxy and Skeleton in Remote Method Invocation

Process P1 ("client")

Process P2 ("server")

Object A

Object B

Skeleton

Request

Reply

Communication module

Remote reference module

Remote object B

Remote reference module

Skeleton & dispatcher for B’s class

Client

Server

Process P1 ("client")

Process P2 ("server")
Proxy

Provides *transparency* by behaving like a local object to the invoker

- The proxy “implements” the methods in the interface of the remote object that it represents. But,...

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

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

**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
}
```
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 and Skeleton in Remote Method Invocation
What about Server Side?
Dispatcher and Skeleton

Each process has one dispatcher, and a skeleton for each local object (actually, class)

The dispatcher receives all request messages from the communication module.
- Uses the method id to select the appropriate method in the appropriate skeleton, passing on the request message.

Skeleton “implements” the methods in the remote interface.
- Un-marshals the arguments in the request message and invokes the corresponding method in the remote object (the actual object).
- It waits for the invocation to complete and marshals the result, together with any exceptions, into a reply message.
Summary of Remote Method Invocation (RMI)

- **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
- Can be poly
Summary

Local objects vs. Remote objects

RPCs and RMIs

RMI: invocation, proxies, skeletons, dispatchers