Lecture 6: Remote Procedure Call

Dr Geoff Sharman
Visiting Professor in Computer Science
Birkbeck College

© Geoff Sharman  Oct 08
Lecture 6

- **Aims to:**
  - Describe the basic mechanism for Remote Procedure Call, as used in procedural and Object Oriented languages
  - Explain how RPC is extended to access data and support distributed transactions
  - Discuss performance, availability and application programming issues relating to Transactional RPC
Remote Procedure Call Mechanism

- enables a program in one system to invoke a program in a remote system transparently

- stub and mirror stub ("buts") are pieces of middleware which hide physical communication

- invocation is synchronous; caller must wait
The Basic Idea

- Make it **easier** to write applications which span two (or more) physical systems

- **Hide** details of communications from the application programmer

- **Provide location transparency**: the target program is usually located via a directory and could be anywhere

- **First described in a 1978 patent application**
A Simple Form of RPC

- LINK PROGRAM(B) COMMAREA(DATA) LENGTH (L)

- Enables a COBOL program to invoke a program named B on a remote system, passing data in a communication area.

- Data is passed in a COBOL structure named DATA which has length L (up to 32K bytes).

- Remote program may modify the data, which is passed back.
Message Format on the Wire

<table>
<thead>
<tr>
<th>HDR</th>
<th>REQ</th>
<th>B</th>
<th>L</th>
<th>RSP</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Stub** assembles *message* & invokes *comms*

- **Mirror stub** disassembles *message* & invokes target *program*; assembles *reply message* & invokes *comms*

- **Stub** disassembles *reply message* & returns to caller

- What can go wrong?
What do we Need Worry About?

- Caller and target systems must use **same** codepage and hardware architecture
- Caller and target middleware need to agree on **message formats & protocol**
- Target program needs to know the **structure** of the data area passed
- Target program, server or physical communications may **fail**
Remote Procedure Call in C

- variables data1, data2, data3, ...
  B(data1, data2, data3)

- Enables a C program to invoke program named B on a remote system, passing data parameters (data1, data2, data3)

- Stub assembles parameters into a message

- Mirror stub uses a description written in Interface Definition Language (IDL) to disassemble the message
**Message Format on the Wire**

<table>
<thead>
<tr>
<th>HDR</th>
<th>REQ</th>
<th>B</th>
<th>L</th>
<th>RSP</th>
<th>data1</th>
<th>data2</th>
<th>data3</th>
</tr>
</thead>
</table>

- **Stub** assembles parameters into message – known as “marshalling” or “flattening”; converts data values if required; disassembles reply message & converts data.

- **Mirror stub** disassembles message into parameters using IDL; invokes target pgm; assembles reply message.

- What can go wrong?
What do we Need Worry About?

- Caller and target systems must agree on conversions for codepage and architecture.

- Caller and target middleware need to agree on message formats, e.g. use Open Network Computing (ONC) or Distributed Computing Environment (DCE) standards.

- Mirror stub needs to know the structure of the data area passed – IDL defines.

- Target program, server, or comms may fail.
Statements written in Interface Definition Language defining program B

IDL Compiler

STUB for B
Finding the Target Program

- Hard coding address of target program (in stub) would require program change if target program moved.
- Better to look up address of B in a local or global directory; no program changes when target program moves.
- Most implementations allow B to be located on System A for testing; may use InterProcess Communication (IPC) in this case.
**RPC in Object Oriented Languages**

- get object reference for object O
  - \( O.B(data1, data2, data3) \)

  Enables a C++ or Java method to **invoke method B of object O** on a remote system, passing arguments \( data1, data2, data3 \)

- **Stub** assembles arguments into a message

- **Target skeleton** uses a description written in Interface Definition Language (IDL) to **disassemble the message**
With OO languages, both the stub and the mirror stub are provided by a piece of middleware called an **Object Request Broker (ORB)**.

The **IDL compiler** creates a **proxy object** and a **skeleton method**.

The **ORB** uses the **directory** (aka “implementation repository”) to locate the target object O (get object reference).
**What do we Need Worry About?**

- Caller and target ORBs need to **agree formats**, e.g. use standardised Common Object Request Broker Architecture (CORBA), or Java Remote Method Invocation (RMI)

- The **message protocol** flowing on the wire is called Internet InterORB Protocol (IIOP)

- Even with these standards, some implementations are **not interoperable**

- ORB needs to know the **data formats for each method** – IDL defines
module <myapplication>
{
    interface O /* corresponds to an object */
    {
        void B /* corresponds to a method */
        (in short data1,
         in string data2,
         in string data3)
        void C /* another method */
    }
}
A common, and long established, use of RPC is for accessing data on remote systems. This is made transparent to the application program by providing a remote data access interface, e.g.

```
READ FILE(x) RID(y) INTO(data-area)
```

```
executeQuery(SQL string)
data = getResultSet.next()
```
enables a program in one system to **read a file** or **query a database** on a remote system **transparently**

**stub packages read request** into a message; mirror **stub unpackages** and executes local read request (for SQL, stub is a **database client**)

**invocation is synchronous**; caller must wait
What do we Need Worry About?

- Caller and target need to agree formats – usually achieved by using a database client supplied with Database Management System.

- **Standard programming interfaces**, e.g. Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC) allow portability across different DBMSs.

- Performance?
Remote Data Access

- ODBC/JDBC fetch one record at a time – expensive if many records are needed

- May be better to place data access logic on the data server
What if we need to use transactions, e.g. because we are updating data?

Business logic will issue BEGIN and COMMIT but data could be on another system, so two (or more) systems need to be involved in a single transaction

To manage this we treat remote systems as Resource Managers
 RPC request message must carry a **Transaction Identifier (TID)**

 Mirror stub on System B must **detect TID** and start a **local transaction**

 Local TM must **register** with calling TM
Transactional RPC (III)

- During COMMIT processing, the calling Transaction Manager sends PREPARE message to each participating TM (using an RPC) and waits for response.

- Then sends COMMIT message to each participating TM and waits for response.

- But this all takes time: IN DOUBT “window” now includes communications delays.
Commit Protocol with Remote System

Commit Protocol with Remote System

COMMIT

Transaction Manager

Prepare
Commit
Commit
Prepare

Resource Manager 1

Resource Manager 2

REMOTE SYSTEM
Resource Manager Registration

- During COMMIT processing, resource managers must be told to PREPARE.

- But how does the Transaction Manager know that they participate in a particular transaction?

- Solution: each resource manager must tell the TM by registering its interest.
BEGIN
update RM1
update RM2
COMMIT

allocate TID
Callback TID
Callback TID

Transaction Manager

TID
RM1
RM2

Resource Manager 1
Resource Manager 2
Worried about Performance?

- RPC imposes a performance **penalty** when compared with a local procedure call
  - Local call ~ 10 instructions
  - Remote call ~ 10,000 instructions

- RDA imposes a **penalty for each trip** to data manager and may involve **multiple trips**

- Transaction lifetime is **longer** – reduces concurrency

- IN DOUBT window is **longer** with TRPC
  - Local transaction ~ disk write delay
  - Distributed transaction ~ communications delay
Another Inconvenient Truth

- RPC, RDA and TRPC are intended to make life easier for the programmer by providing Location Transparency.

- But they don't provide Performance Transparency.

- Basic law of nature: you can't have all three!
Summary

You should now be able to:

- Describe the basic mechanism for Remote Procedure Call, as used in procedural and Object Oriented languages
- Explain how RPC is extended to access data and support distributed transactions
- Discuss performance, availability and application programming issues relating to Transactional RPC