Implementing Remote Procedure Calls

By Andrew D. Birrell and Bruce Jay Nelson

Presented By: Abdussalam Alawini
Reviewed By Prof. Jonathon Walpole
Contents

1. Introduction
   - Where we are? And where we’re heading?
   - What is “Remote” Procedure Call
   - Design Decisions
   - Goals of RPC
2. Implementing RPC
   - RPC Structure
   - Binding Process
   - RPC NT Protocol
3. Data Integrity and security
   - Exception Handling
   - Use of Processes
   - Security
4. Optimizations and Evaluation
   - Optimizations
   - Performance evaluation
1. Introduction
Where we are? And where we’re heading?

- We saw managing OS control flow over one shared-memory using:
  - Procedural (thread-based) Model
  - Message passing (event-based) Model
- Now we will see a procedural (thread-based) programming abstraction based on message passing (event-based) substrate.
  - Threads passed from one machine to the other and back again.
  - Programmer unaware of the underlying message-based substrate.
What is A “Remote” Procedure Call?

Z=F(x,y)
What is a “Remote” Procedure Call?

Z = F(x, y)

Communication Network

Server

Client
What is a “Remote” Procedure Call?

Communication Network

Server

F(x,y)

Compute F(x,y)

Client
What is a “Remote” Procedure Call?

Communication Network

Client

Server

Compute F(x,y)

result
What is A “Remote” Procedure Call?
Design Decisions (Any Alternatives)

- Message passing
  - Same reliable and efficient message (and reply) problems
  - Procedure calls are the Major transfer mechanism in Mesa. (Remember Duality)
- Remote fork
  - No major changes in design problems
- Shared Address space
  - Integration of remote address spaces
  - Efficiency issues
Goals of implementing RPC

• Simplicity
  • Make RPC as similar to procedure calls as possible
  • Make distrusted computation easier

• Efficiency
  • Make semantic of RPC package as powerful as possible without losing efficiency or simplicity

• Security
  • Secure end-to-end communications with RPC
2. Implementing RPC
RPC Facility Structure

- **Caller Machine**
  - User (user application code module)
  - User-Stub
  - Caller instance of RPCRuntime (RPC communications package)
- **Callee Machine**
  - Server (server code module)
  - Server-stub
  - Callee instance of RPCRuntime
RPC Components Interactions

- **Caller Machine**
  - User Application
  - User-stub
  - Caller RPCRuntime

- **Callee Machine**
  - Libraries (Server code)
  - Server-stub
  - Callee RPCRuntime

- **Network**
  - Normal Local Proc Call
  - Import Interface
  - Export Interface
RPC Components Interactions

Caller Machine
- User Application
- User-stub
- Call Packet
- Caller RPCRuntime

Network

Callee Machine
- Libraries (Server code)
- Server-stub
- Callee RPCRuntime

Import Interface
Pack target spec + proc arguments
Export Interface
RPC Components Interactions

Caller Machine
- User Application
- User-stub
- Caller RPCRuntime

Import Interface

Network

transmit packet “Reliably”

Export Interface

Callee Machine
- Libraries (Server code)
- Server-stub
- Callee RPCRuntime

Call Packet
RPC Components Interactions

Caller Machine

User Application
User-stub
Caller RPCRuntime

Import Interface

Callee Machine

Libraries (Server code)
Server-stub
Callee RPCRuntime

Export Interface

Network

Pass them to server-stub

Call Packet
RPC Components Interactions

Caller Machine

- User Application
- User-stub
- Caller RPCRuntime

Import Interface

Unpack & make local call

Network

Callee Machine

- Libraries (Server code)
- Server-stub
- Callee RPCRuntime
RPC Components Interactions

Caller Machine
- User Application
- User-stub
- Caller RPCRuntime

Import Interface

Do work and return results

Export Interface

Callee Machine
- Libraries (Server code)
- Server-stub
- Results Packet
- Callee RPCRuntime

Network
RPC Components Interactions

Caller Machine
- User Application
- User-stub
- Caller RPCRuntime
- Results Packet

Import Interface

Network

Callee Machine
- Libraries (Server code)
- Server-stub
- Callee RPCRuntime

Export Interface
Who Does What?

- Caller Machine
  - User Application
  - User-stub
  - Caller RPCRuntime

- Programmer
  - Interface
  - Lupine (Auto Generation)
  - Part of Cedar

- Mesa Interface Modules

- Callee Machine
  - Libraries (Server code)
  - Server-stub
  - Callee RPCRuntime

- Network
Binding Process

- How does a client of the binding mechanism specify what he want to be bound to?
  - Naming

- How does the caller specify the callee machine address and the specific procedure he wants to invoke?
  - Locating
Naming (Interface Name)

- **Type:** Which interface the caller expect the callee to implement.
  - Service Name (e.g. Mail-server)
- **Instance:** Which particular implementer of an abstract interface is desired.
  - Machine Address (e.g. Specific mail-server address)
Design ideas

1- Include network address in user application
   • Too early binding

2- Broadcasting Protocol
   • Too much interference with innocent bystanders
   • Not convenient for binding machines not in the same local network.
Locating an appropriate exporter
Using Grapevine Database

<table>
<thead>
<tr>
<th>Type (Group)</th>
<th>Member-list</th>
<th>Instance (Individual)</th>
<th>Connect-site</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileAccess</td>
<td>{Ebbets, Luther, Facc}</td>
<td>Ebbets</td>
<td>3#22#</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Luther</td>
<td>3#276#</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Facc</td>
<td>3#43#</td>
</tr>
</tbody>
</table>

Export Interface Type FileAccess Instance Ebbets

Server 1 (Ebbets) 3#22#
Server 2 (Luther) 3#276#
Server 3 (Facc) 3#43#
Locating an appropriate exporter
Using Grapevine Database

<table>
<thead>
<tr>
<th>Type (Group)</th>
<th>Member-list</th>
<th>Instance (Individual)</th>
<th>Connect-site</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileAccess</td>
<td>{Ebbets, Luther, Facc}</td>
<td>Ebbets</td>
<td>3#22#</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Luther</td>
<td>3#276#</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Facc</td>
<td>3#43#</td>
</tr>
</tbody>
</table>

Import Interface
Type FileAccess
Instance Ebbets
Steps of Exporting an Interface
(Making it available to a client)

Grapevine Database

- Do update

Callee Machine

- RPCRuntime
  - Record in table
  - SetConnect
  - AddMember

- Server-stub
  - Export [FA, Ebbets,...]

- Server
  - Export [FA, Ebbets]

  Call server-sub

  Return
Steps of Exporting an Interface (Making it available to a client)

Grapevine Database

- Do update
- Do update

Callee Machine

RPCRuntime
- Record in table
- SetConnect
- AddMember

Server-stub
- Export [FA, Ebbets, ...]

Server
- Export [FA, Ebbets]

Call ExportInterface (Interface Name, Dispatcher)

Return
Steps of Exporting an Interface
(Making it available to a client)

ExportInterface
- make sure that type and instance is correct

Grapevine
- Record in table
  - SetConnect
    - AddMember

RPCRuntime
- Export [FA, Ebbets,...]

Callee Machine
- Export [FA, Ebbets]

Server-stub

Server
- Interface exported (Can be accessed remotely)
- Return

Do update
Steps of Exporting an Interface
(Making it available to a client)

RPCRuntime
Record exported
interface in a table

Grapevine

Callee Machine
RPCRuntime
Record in
table

Server-stub
Export [FA,
Ebbets, ...]

Server
Export [FA,
Ebbets]

Do update
SetConnect
AddMember
Return

Do update
Steps of Exporting an Interface
(Making it available to a client)

Grapevine Database

- Do update

Callee Machine

- RPCRuntime
  - Record in table
  - SetConnect
  - AddMember

- Server-stub
  - Export [FA, Ebbets,...]

- Server
  - Export [FA, Ebbets]

Return

Set the address of the current machine in the connect-site
Adds the instance to the member-list of the type of the instance
Steps of Exporting an Interface
(Making it available to a client)

Grapevine Database

- Do update
- Do update

Callee Machine

RPCRuntime
- Record in table
- SetConnect
- AddMember

Server-stub
- Export [FA, Ebbets,...]
- Export [FA, Ebbets]

Server
- Interface exported (Can be accessed remotely)
- Return

Interface exported (Can be accessed remotely)
Steps of Binding An Importer to An Exporter Interface
(Getting ready for remote calls)
Call user-stub
“I need to import Interface[FA, Ebbets]”
Call ImportInterface “Here’s the type and instance we need”

Grapevine Database

Caller Machine

User

User-stub

RPCRuntime

Return

Record result

Import [FA, Ebbets]

openFile -> 2

Table lookup

Check UID

In table

2 -> openFile

X= openFile(y)

Callee Machine

RPCRuntime

Server-stub

Server
OK, I will ask grapevine DB and get the NT address of exporting Interface.
Here’s the NT address “connect-site”

Alright, I’ll call the exporter RPCRuntime to get the binding info

X = openFile(y)

openFile -> 2

greenText

table lookup

Check UID

In table

2 -> openFile

X = openFile(y)
Now let see how this will work

X = openFile(y) - openFile -> 2

Check UID In table
2 -> openFile

X = openFile(y)
Now let see how this will work

1- Retrieve interface binding information
2- Prepare call packet

X = openFile(y)

openFile -> 2

transmit

Check UID
In table

2 -> openFile

X = openFile(y)
Grapevine Database

Now let see how this will work

Caller Machine

User

User-stub

RPCRuntime

Callee Machine

RPCRuntime

Server-stub

Server

1-Lookup current exports
2- verify UID
3- send call packet to dispatcher

Alright, I will transmit it to the machine with address "connect-site"

In table lookup

X= openFile(y)

openFile -> 2

transmit

Check UID

2 -> openFile

X= openFile(y)

X= openFile(y)

Now let see how this will work

Alright, I will transmit it to the machine with address "connect-site"

In table lookup

X= openFile(y)

openFile -> 2

transmit

Check UID

2 -> openFile

X= openFile(y)
Now let see how this will work.

After unpacking, Dispatcher uses the info to map to the right procedure.

<table>
<thead>
<tr>
<th>Caller Machine</th>
<th>Callee Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User</strong></td>
<td><strong>Server</strong></td>
</tr>
<tr>
<td>Import [FA, Ebbets]</td>
<td>RPCRuntime</td>
</tr>
<tr>
<td>Return</td>
<td>Get Connect</td>
</tr>
<tr>
<td>X = openFile(y)</td>
<td>Bind [FA, Ebbets]</td>
</tr>
<tr>
<td><strong>User-stub</strong></td>
<td><strong>Server-stub</strong></td>
</tr>
<tr>
<td>Import [FA, Ebbets]</td>
<td>Get Connect</td>
</tr>
<tr>
<td>Record result</td>
<td>transmit</td>
</tr>
<tr>
<td><strong>RPCRuntime</strong></td>
<td><strong>Server</strong></td>
</tr>
<tr>
<td>Lookup</td>
<td>X = openFile(y)</td>
</tr>
<tr>
<td><strong>Grapevine Database</strong></td>
<td><strong>RPCRuntime</strong></td>
</tr>
</tbody>
</table>
Now let see how this will work

Call the local procedure using provided arguments
Binding Mechanism Advantages

- Importing an interface has no effect on the data structure of the exporting machine
- The use of UID means that bindings are implicitly broken if the exporter crashes and restarts.
- Restricting the set of users who can update Grapevine DB.
  - To avoid security problems.
- Several choices of binding time
  - Importer specifies the type only
  - Binding at compile-time, by delaying the instance value
Packet-Level Transport Protocol (Why?)

- Substantial performance gains
- Minimize the elapsed real-time between initializing a call and getting results.
  - Unacceptable to have large amount of state info.
  - Unacceptable to have an expensive handshaking.
- Guarantee procedure in the server has been invoked precisely once.
Simple Calls Example

Proposes
1- Ensure the result packet is for this call
2- Callee can eliminate duplicates

Caller Machine

User
Call

RPC+Stub
Send Call Packet
Wait Ack or Result
Return

Callee Machine

RPC+Stub
Invoke proc
Send results

Server
Do call
Return
Simple Calls Example

Desired Procedure Info:
- UId, Table Index,
- Procedure entry point.

Caller Machine

User
- Call
- Send Call Packet
- Wait Ack
- or Result
- Return

RPC+Stub

Callee Machine

RPC+Stub
- Invoke proc
- Send results
- Do call
- Return

Server

Proc

Call
- CallID | DP info | Arugs

Invoke proc
- Send results
- Do call
- Return

Return

User
- Call
- Send Call Packet
- Wait Ack
- or Result
- Return
Simple Calls Example

Procedure arguments which is the input or output that will be used by RP.

User
- Call
- Send Call Packet
- Wait Ack or Result
- Return

RPC+Stub
- CallID
- DP info
- Arugs

Callee Machine
- RPC+Stub
  - Invoke proc
  - Send results
- Server
  - Do call
  - Return

Callee Machine
- RPC+Stub
- Server
- Do call
- Return

Caller Machine
- Call
- Call
- RPC+Stub
  - Send Call Packet
  - Wait Ack or Result
  - Return

Procedure arguments which is the input or output that will be used by RP.
**Simple Call Example**

**Caller Machine**

- User
- Call
  - Send Call Packet
  - Wait Ack or Result
  - Return

**Call**

- [Machine Id, Process] | Seq#
- CallID | DP info | Arugs

**Callee Machine**

- RPC+Stub
- Server
  - Do call
  - Send results
  - Return

- Invoke proc
  - CallID DP info Arugs

- Monotonic for each activity. No repeats (calls might eliminated as dup.)

- Each activity has at most 1 remote call at any time

**Activity**

- Calling Machine Id, and Process Id

- Each activity has at most 1 remote call at any time

- Monotonic for each activity. No repeats (calls might eliminated as dup.)
Simple Calls Example

RPC compares Seq# in CID to the one it has in a table that maintains seq# of the last call invoked by each calling activity.
Simple Calls Example

If caller Seq# EQ Cseq then Ack if needed
if callerSeq# LT Cseq then drop “repeated”
If callerSeq# GT Cseq then new call packet

[Machine Id, Process] Seq#

If caller Seq# EQ Cseq
then Ack if needed
if callerSeq# LT Cseq
then drop “repeated”
If callerSeq# GT Cseq
then new call packet

Call
Send Call Packet
Wait Ack or Result
Return

Result
CallID results

User
RPC+Stub
Call

RPC+Stub
Server
Invoke proc
Do call
Send results

Return

Callee Machine
Caller Machine
Simple Calls Example

Result packet is sufficient Ack to the caller.

Call

RPC+Stub

User

Result

User

RPC+Stub

Callee Machine

Server

Callee Machine

User

RPC+Stub

Server

User

RPC+Stub
Is There A “Stack Ripping”? 

- In manual stack management the necessary data is taking off the stack and then putting it on the heap.
- In RPC it’s taking off the stack and putting it in a message.
  - Sent to the other side.
- Very similar to taking the state off the stack and putting it in a continuation.
  - Executed by a separate event handler.
Complicated Call Example

Call pkt must be acknowledged
Arguments going to be in 2 packets

**Caller Machine**

**User**
- Call

**RPC+Stub**
- Send CallPkt
  - CallID, Pkt=0, PlsAck,…
- Wait for Ack
  - CallID, Pkt=0,
- Build next pkt
  - CallID, Pkt=1, dontAck,….
- Transmit it
  - CallID, Pkt=1, PlsAck ,….
- Wait for ack
  - CallID, Pkt=1,
- Retransmit
  - CallID, Pkt=2, dontAck ,..
- Wait for ack
  - CallID, Pkt=2, PlsAck ,….
- Ack
  - CallID, Pkt=2,
- Acknowledge
  - CallID, Pkt=2,
- Return

**Callee Machine**

**RPC+Stub**
- Start arg record
  - CallID, Pkt=0,
  - Data
  - CallID, Pkt=1, dontAck,….
  - Data
  - CallID, Pkt=1, PlsAck ,….
- Ack
  - CallID, Pkt=1,
- Acknowledge
  - CallID, Pkt=1,
- Invoke call
  - CallID, Pkt=2, dontAck ,..
- Ack
  - CallID, Pkt=2,
- Acknowledge
  - CallID, Pkt=2,
- Send result
  - Wait for ack
  - Retransmit
  - Wait for ack
  - Idle
- Return

**Server**
- Do call
- Return
Complicated Call Example

Caller Machine

User
- Call

RPC+Stub
- Send CallPkt
- Wait for Ack
- Build next pkt
- Transmit it
- Wait for ack
- Retransmit
- Wait for ack
- Wait for result
- Return
- Acknowledgment

Callee Machine

RPC+Stub
- CallID, Pkt=0, PlsAck,...
- Data
- CallID, Pkt=1, dontAck,...
- Ack
- CallID, Pkt=1,
- Result
- CallID, Pkt=2, dontAck,...
- Ack
- CallID, Pkt=2,

Server

- Start arg record
- Acknowledge
- Wait next pkt
- Invoke call
- Acknowledge
- Wait next pkt
- Do call
- Return

Acknowledge
- Wait next pkt
- Send result
- Wait for ack
- Retransmit
- Wait for ack
- Idle

Send the rest of the data in Data Pkt
Complicated Call Example

Caller Machine

User

Call

Send CallPkt
Wait for Ack
Build next pkt
Transmit it
Wait for ack
Retransmit
Wait for ack
Wait for result
Return

RPC+Stub

Call
CallID, Pkt=0, PlsAck, ...

Ack
CallID, Pkt=0,

Data
CallID, Pkt=1, dontAck, ....

Ack
CallID, Pkt=1,

Result
CallID, Pkt=2, dontAck, ....

Ack
CallID, Pkt=2,

Acknowledgment

Callee Machine

RPC+Stub

Start arg record

Acknowledge
Wait next pkt

Invoke call

Do call

RPC+Stub

Start arg record

Acknowledge
Wait next pkt

Invoke call

Do call

Server

Idle

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, PlsAck, ....

Data
CallID, Pkt=1, Pls Ack, ....

Acknowledgment

Return

Now all arguments are ready, so the dispatching process starts.
Complicated Call Example

In case of lost packet, long call duration or long gaps between calls retransmit the last pkt and ask for ack.

**Caller Machine**

- **User**
  - Call

- **RPC+Stub**
  - Send CallPkt
  - Wait for Ack
  - Build next pkt
  - Transmit it
  - Wait for Ack
  - Retransmit
  - Wait for Ack
  - Wait for result
  - Return
  - Acknowledgment

**Callee Machine**

- **RPC+Stub**
  - CallID, Pkt=0, PlsAck,
  - Data
  - CallID, Pkt=1, dontAck,
  - Ack
  - CallID, Pkt=1,
  - Result
  - CallID, Pkt=2, dontAck
  - Result
  - CallID, Pkt=2, PlsAck
  - Ack
  - CallID, Pkt=2,

- **Server**
  - Start arg record
  - Acknowledge Wait next pkt
  - Invoke call
  - Acknowledge
  - Send result
  - Wait for ack
  - Retransmit
  - Wait for ack
  - Return
  - Idle
Complicated Call Example

Caller Machine

User:
- Call
  - Send CallPkt
  - Wait for Ack
  - Build next pkt
  - Transmit it
  - Wait for ack
  - Retransmit
  - Wait for ack
  - Wait for result
  - Return
  - Acknowledgment

RPC+Stub:
- Call
  - CallID, Pkt=0, PlsAck,
  - Ack
  - CallID, Pkt=0,
  - Data
  - CallID, Pkt=0, dontAck,
  - Ack
  - CallID, Pkt=1, PlsAck,
  - Data
  - CallID, Pkt=1, dontAck,
  - Ack
  - CallID, Pkt=1,
  - Result
  - CallID, Pkt=2, dontAck,
  - Result
  - CallID, Pkt=2, PlsAck,
  - Ack
  - CallID, Pkt=2,

Callee Machine

RPC+Stub:
- Start arg record
  - Acknowledge
  - Wait next pkt
  - Invoke call
  - Acknowledge
  - Send result
  - Wait for ack
  - Retransmit
  - Wait for ack
  - Wait next pkt
  - Idle

Server:
- Do call
  - Return

Satisfying Ack: process waits for results. Caller sends probe pkts periodically (can detect communication failure)
### Complicated Call Example

**Caller Machine**
- **User**
  - Call
- **RPC+Stub**
  - Send CallPkt
  - Wait for Ack
  - Build next pkt
  - Transmit it
  - Wait for ack
  - Retransmit
  - Wait for ack
  - Wait for result
  - Return
  - Acknowledgment

**Callee Machine**
- **RPC+Stub**
  - Start arg record
  - Acknowledge
  - Wait next pkt
  - Invoke call
  - Acknowledge
  - Send result
  - Wait for ack
  - Retransmit
  - Wait for ack
  - Idle

### Acknowledgment
- CallID, Pkt=0
- PlsAck, ...

### Data
- CallID, Pkt=1
- dontAck, ...

### Result
- CallID, Pkt=2
- dontAck, ...

---

Send results
And wait for another call “works as an Ack”
Complicated Call Example

Caller Machine

User
- Call
  - Send CallPkt
  - Build next pkt
  - Transmit it
  - Wait for Ack
  - Retransmit
  - Wait for Ack
  - Wait for result
  - Return

RPC+Stub
- Call
  - CallID, Pkt=0, PlsAck,…

Callee Machine

RPC+Stub
- Start arg record
  - Acknowledge
  - Wait next pkt
  - Invoke call
  - Acknowledge
  - Send result
  - Wait for ack
  - Retransmit
  - Wait for ack
  - Idle

Server
- Do call
  - Acknowledge
  - Send result
  - Wait for ack

Return results to caller process (user code)
4. Data Integrity and security
Exception Handling (Remote Process Exception)

- Two level of exception
  - Communication Failure Exception (Explained with complicated call example), considered to be the primary difference between procedure call and RPC
  - Remote Process Exception
Exception Handling

Caller Machine

User
Call

RPC+Stub
Send Call Packet

Wait Ack, Result or Exception

Callee Machine

RPC+Stub
Invoke proc

Server
Do call

Caller does sends regular call packet as we saw before
Exception Handling

If an exception occurs, the process passes the exception to RPC.

Caller Machine

User
- Call
- Catch exception

RPC+Stub
- Send Call Packet
- Wait Ack, Result or Exception
- Return With exception

Callee Machine

RPC+Stub
- Exception Call
- CallID Exception

Server
- Do call
- Exception
- Send Exception
- Return With exception

Call
- CallID
- DP info
- Arugs

Send Call
- Packet
Exception Handling

Prepare and send exception packet instead of results packet

Caller Machine

User
- Call
- Catch exception

RPC+Stub
- Send Call Packet
- Wait Ack, Result or Exception
- Return With exception

Callee Machine

RPC+Stub
- CallID DP info Arugs
- Invoke proc

Server
- Do call
- Exception
- Send Exception
- Return With exception
Exception Handling

Caller Machine

User

Call

RPC+Stub

Send Call Packet

Wait Ack, Result or Exception

Return With exception

Callee Machine

RPC+Stub

Invoke proc

Send Exception

Server

Do call

Exception

Don’t invoke new call, instead raise an exception in the calling process

User

Invoke proc

Send Exception

Return With exception

Server

Do call

Exception

Don’t invoke new call, instead raise an exception in the calling process.

Callee Machine

Send Call Packet

Wait Ack, Result or Exception

Return With exception
Exception Handling

Caller Machine

User
- Call

RPC+Stub
- Send Packet
  - Wait Ack, Result or Exception
  - Return With exception

Callee Machine

RPC+Stub
- Invoke proc
- Send Exception

Server
- Do call
- Return With exception

If there's a catch phrase, exception will be handled and results will be sent back to the callee machine.
Exception Handling

Caller Machine

User

Call

RPC+Stub

Send Call Packet

Wait Ack, Result or Exception

Catch exception

CallID  DP info  Args

Callee Machine

RPC+Stub

Invoke proc

Send Exception

Server

Do call

Exception

If the catch phrase terminated by a jump, then the callee will be notified
Exception Handling

**Caller Machine**

- User
  - Call
- RPC+Stub
  - Send Call Packet
  - Wait Ack, Result or Exception
  - Return With exception
  - Send EH results
  - Catch exception
    - Call with Exception results

**Callee Machine**

- RPC+Stub
  - Call ID
  - Exception
- Server
  - Do call
  - Exception
  - Return With exception
  - Invoking proc that caused the exception
  - Continue doing call

Exception handling results packet.
Exception Handling

Caller Machine

User
- Call
  - Call with Exception results
  - Catch exception

RPC+Stub
- Send Call Packet
  - Wait Ack, Result or Exception
  - Return With exception
  - Send EH results

Callee Machine

RPC+Stub
- Call with Exception results

Server
- Do call
  - Return With exception

Events processed normally

Call
- CallID DP info Arugs

Exception
- CallID Exception
- Catch exception

Results
- CallID DP info Arugs
  - Invoke proc that caused the exception
  - Continue doing call
Use of Processes

Process 7 made a remote procedure call to process 3

Caller Machine

Callee Machine
Use of Processes

This part is the process identifiers, which is part of the activity info within the CallID

Caller Machine

Callee Machine

SrcProcess  DestProcess

Network
Use of Processes

Interrupt handler in RPC look at the destProcess, if the process is waiting then it transfers the pkt directly.

Idle Server Processes
Processes Waiting For RPC packet

SrcProcess | DestProcess
7 | 3

Network

Caller Machine

Callee Machine
Use of Processes

If there’s no corresponding process waiting for this packet then it will be dispatched to one of the free processes (Current, New, or Dup)

Caller Machine

Callee Machine

Network

idle server processes

Processes Waiting For RPC packet

SrcProcess DestProcess
Use of Processes

RPC will exchange SrcProcess with DestProcess so that it can return the result to right process.
Use of Processes

- Idle Server Processes
- Processes Waiting For RPC packet
- Caller Machine
- Callee Machine

<table>
<thead>
<tr>
<th>SrcProcess</th>
<th>DestProcess</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>
Use of Processes

RPC now transmit the packet back to the caller (result, exception, or Ack)

Caller Machine

Callee Machine

Network

Idle Server Processes

Processes Waiting For RPC packet

<table>
<thead>
<tr>
<th>SrcProcess</th>
<th>DestProcess</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>
Security

- RPC package and protocol provides Encryption-based security of calls
  - End-to-end encryption of calls and results
  - Detect tempts of modification, reply, or creation of calls.
- Restricted set of users who can update Grapevine DB.
  - Exporting services is restricted
- Grapevine DB used as authentication service.
4. Optimizations and Evaluation
Optimizations

- Use of idle processes in caller and callee machines to reduce process creation and process swaps.
- The use of process source and destination allow processes to get the packets they’re waiting for directly.
- Use of subsequent packet for implicit acknowledgments of previous packets.
- Avoid the cost of establishing and termination connection by the implementation of packet-level protocol.
Performance evaluation

- They’ve measured the elapsed time between two machines for 12,000 calls for each of the following procedures:
  - 0-10 arguments/results.
  - 1-100 word array.
  - Caller Resume and unwind exception handling.
### Performance Results

Table I. Performance Results for Some Examples of Remote Calls

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Minimum</th>
<th>Median</th>
<th>Transmission</th>
<th>Local-only</th>
</tr>
</thead>
<tbody>
<tr>
<td>no args/results</td>
<td>1059</td>
<td>1097</td>
<td>131</td>
<td>9</td>
</tr>
<tr>
<td>1 arg/result</td>
<td>1070</td>
<td>1105</td>
<td>142</td>
<td>10</td>
</tr>
<tr>
<td>2 args/results</td>
<td>1077</td>
<td>1127</td>
<td>152</td>
<td>11</td>
</tr>
<tr>
<td>4 args/results</td>
<td>1115</td>
<td>1171</td>
<td>174</td>
<td>12</td>
</tr>
<tr>
<td>10 args/results</td>
<td>1222</td>
<td>1278</td>
<td>239</td>
<td>17</td>
</tr>
<tr>
<td>1 word array</td>
<td>1069</td>
<td>1111</td>
<td>131</td>
<td>10</td>
</tr>
<tr>
<td>4 word array</td>
<td>1106</td>
<td>1153</td>
<td>174</td>
<td>13</td>
</tr>
<tr>
<td>10 word array</td>
<td>1214</td>
<td>1250</td>
<td>239</td>
<td>16</td>
</tr>
<tr>
<td>40 word array</td>
<td>1643</td>
<td>1695</td>
<td>566</td>
<td>51</td>
</tr>
<tr>
<td>100 word array</td>
<td>2915</td>
<td>2926</td>
<td>1219</td>
<td>98</td>
</tr>
<tr>
<td>resume except’n</td>
<td>2555</td>
<td>2637</td>
<td>284</td>
<td>134</td>
</tr>
<tr>
<td>unwind except’n</td>
<td>3374</td>
<td>3467</td>
<td>284</td>
<td>196</td>
</tr>
</tbody>
</table>
Performance summary

- For transferring large amounts of data in one direction, products other than RPC have advantage.
  - Transfer fewer packets in the other direction.
- They haven’t measure the performance of exporting or importing interfaces.
- Used by:
  - Alpine [File server supports multi-machine transactions]
  - Control communication for an Ethernet-based for telephone and audio project
  - Networking games.
Conclusion

- RPC is one of the techniques used in communication between processes outside their address space boundaries.
- Make distributed programming easier.
- RPC protocol for avoiding communications overhead.
- Considered to be the first protocol suite that provides security in open network communications.
- RPC succeed in providing reliability and integrity for data and control transfer.
- More work needs to be done (By now, I think there is a lot of work that has been done)
Discussion