RPC IMPLEMENTATION: BASIC STEPS

Instructor: Prasun Dewan
Department of Computer Science
University of North Carolina at Chapel Hill
dewan@cs.unc.edu
**Simulating Remote Procedure Call**

- **Goal is to call a method with params**
- Send expressions encoding method and parameters
- These are assigned to corresponding remote variables
- Side effect of assignment is to call method with parameters
- vs. local call?
Activation Record vs. RPC Messages

Activation Record (parameters, return value, return address, and other info)

Activation Record (Parameters, return value, return address, and other info)

(Un)marshalling of call: translation between call and message

Call results in marshalling and unmarshalling of call/return value rather than push and pop of activation record on stack

Activation Record (Parameters, return value, return address, and other info)

Calling Process

Call message (operation id, parameters, source)

Called process

Return message (return value)
**DATA SET UP**

- Server exports external description of data (RA) port
  - Statically or dynamically

- Server starts and makes data port connectable

- Client determines external description
  - Statically or dynamically

- Client starts and establishes connection
RPC Setup

Server exports external description of calls serviced on RA (data) port

- Statically or dynamically

Client finds this information

- Statically or dynamically
PROCESSING OF CALL MESSAGE AND RETURN VALUE

1. Generate call message for call using exported information
2. Process call message in caller process
3. Communicate call message on data port and handle errors
4. Process call message in callee process
5. Invoke called operation
PROCESSING OF CALL MESSAGE AND RETURN VALUE

1. Possibly create return message in callee process
2. Communicate return message to caller process
3. Process return message in callee process and handle errors
4. Return value to caller operation
AUTOMATION OF REMOTE PROCEDURE CALL

- Process/Thread
  - <method>
  - <params>
  - Port (Mailbox)
    - <method>
    - <params>
    - Process/Thread
      - <method>
      - <params>
      - <method>
        - (<params>)

- App @ Calling Site
- RPC System @ Calling Site
- Division of labor?
- App @ Callee Site
- RPC System @ Callee Site
RPC AWARENESS IN APPLICATION: TWO EXTREMES

RPC clients and servers handle all aspects of RPC and are completely aware of all of the steps listed below.

RPC clients and servers are completely unaware of distribution details.

Java RMI falls close to complete transparency.

GIPC RMI supports a wider spectrum that falls closer and further.

Can one have complete transparency? (Answered at the end – actually deleted from this overview of basic steps.)
RPC vs. RMI

RPC: O-O or Conventional

RMI: O-O
Main O-O Examples

Java RMI

GIPC RMI

Will use GIPC as an example of detailed implementation
GIPC vs. RMI

Allow remote assignment and remote procedure call on same port

Notion of connected client and server port visible

Distinguishing between regular and RPC messages?
DISTINGUISHING RPC AND REGULAR MESSAGES

- Use special GIPC types for RPC Messages
- GIPC RPC traps instances of these types and passes others to application listeners of underlying data ports
- Applications cannot pass instances of these types directly
- Should allow GIPC developers to change the nature and processing of these messages without changing the complete RPC system

Nature of these messages?
PROCESSING OF CALL MESSAGE AND RETURN VALUE

Generate call message for call using exported information

Possibly create return message in callee process

Calling Process

Call message (operation id, parameters, source)

Non RPC Message

Non RPC Message

Return message (return value)

Called process
**Method Return Value**

```
RPCReturnValue  Object returnValue
```
CALL MESSAGE

Call <ObjectIDType>
- transient Method method
- ObjectIDType targetObject
- Object[] args

extends <String>

SerializableCall
- SerializedMethod serializedMethod

SerializesMethod
- transient Method method
- String methodName
- String className
- String[] parameterTypeNames
**SERIALIZING A METHOD**

```
Marshaller<ObjectIDType>

Call<ObjectIDType> marshallCall(
    ObjectIdType aTargetObject, Method aMethod, Object[] anArgsList)

ObjectIdType targetObject

transient Method serializableMethod

Object[] args
```
**Port vs. GIPC-RMI**

- **Client Object**
  - **Server Proxy**
  - \( m(p^1, \ldots, p^N) \)

- **Server Object**
  - **Server Port**
  - \( m(p^1, \ldots, p^N) \)

**Synchronization?**
Processing of Call Message and Return Value

1. Generate call message for call using exported information
2. Process call message in caller process
3. Communicate call message on data port and handle errors
4. Process call message in callee process
5. Invoke called operation
PROCESSING OF CALL MESSAGE AND RETURN VALUE

1. Possibly create return message in callee process
2. Communicate return message to caller process
3. Process return message in callee process and handle errors
4. Return value to caller operation
OBJECT FORWARDERS AND SERIALIZERS

Sender <Object> (Object Port)

Send Forwarder <Object, Object>

Serializer <Object, ByteBuffer>

Sender<ByteBuffer> (Buffer Skeleton)

Receive Notifier<Object>

ReceiveForwarder <Object, Object>

Deserializer <ByteBuffer, Object>

Receive Listener <ByteBuffer> (Buffer Skeleton)
Object Forwarders and Serializers

Sender <Object> (RPC Port)

Call Message and Return Message SendTrapper <Object, Object>

Sender <Object> (Object Port)

Receive Notifier<Object>

Call Message and Return Message ReceiveTrapper <Object, Object>

Receive Listener <Object> (RPC Port)

Shared Data Structures

Send trapper may need to block sender to implement synchronous operation

Send and receive trapper may need to share information about remote references sent and received
**OBJECT FORWARDERS AND SERIALIZERS (Review)**

Send trapper may need to block sender to implement synchronous operation
OBJECT FORWARDERS AND SERIALIZERS (REVIEW)

SendTrapper and ReceiveTrapper at Caller Share an Instance of Synchronizing SentCallCompleter

Coordination between Caller and Called Site?
Duplex Sent CallCompleter (Public and Non Public Methods, New Names)

ADuplexSentCallCompleter

- Object returnValueOfRemoteMethodCall(String aRemoteEndPoint, Object aCall)
- Object returnValueOfRemoteFunctionCall(String aRemoteEndPoint, Object aCall)
- Object returnValueOfRemoteProcedureCall(String aRemoteEndPoint, Object aCall)
- processReturnValue(String aSource, Object aMessage);

returnValueOfRemoteMethodCall() called, after call message sent: receive return value and block caller if necessary

Renamed: returnValueOfRemoteFunction(Procedure)Call → getReturnValueOfRemoteFunction(Procedure)Call, processReturnValue → returnValueReceived
FUNCTION CALL SYNCHRONIZATION AND MATCHED CLIENT/SERVER OBJECTS

- ReceiveTrapper
  - receive()

- SendTrapper
  - send()
  - put()
  - get()

- RPCReturnValue Queue (BoundedBuffer)
  - put()
  - get()
  - receive()

- Received Call Invoker
  - message
  - Received()

- handle Function Return()

- remote function call is blocking
  - Tm(...)
  - RemoteObject

- Proxy
  - Tm(...)

- Marshall
  - marshall Call

- ReturnValueOfRemoteFunctionCall

- SentCall Completer
  - process Return Value()
PROCEDURE CALL SYNCHRONIZATION AND MATCHED CLIENT/SERVER OBJECTS

Remote procedure call is not blocking

Received Call Invoker

Received

handle

Procedure

Return()

message

Received()

RemoteObject

Remote procedure call is not blocking

Proxy

Marshaller

SendTrapper

send()

marshall

Call

SendTrapper

Call

SendTrapper

receive()
SYNCHRONOUS CALLBACK IN REMOTE CALL

RPCMessage Receiver

messageReceived()

m^(1)(...)

Remote function call is blocking

RemoteObject

Selector

select()

NIO Selection Thread

Call

Return Val

RPCReturn Value Queue
(BoundedBuffer)

put()

get()

Proxy

T m^(2)(...)

Call

Call
GIPC Asynchronous Invoker (Factory Selected)

How many return value queues and queue size?

Id in Call and Return Value?

Remote function call is blocking

RPCMessage Receiver

messageReceived()

Selector

select()

RemoteObject

m₁(...)

GIPC Asynchronous MethodInvoker

Factory determines if asynchronous method invoker is created

Remote function call is blocking

NIO Selection Thread

RPCReturnValue Queue (BoundedBuffer)

put()

get()
GLOBAL RECEIVED VALUE INSTEAD OF BUFFER

Object returnValueOfRemoteFunctionCall(String aSource, Call aCall)

Multiple calls can be made concurrently to a server by different threads

Received Return Value

wait()

Clear and return received value

processReturnValue(String aSource, Object aMessage)

Store return value

wait()
GLOBAL BOUNDED BUFFER SERVERS

Object returnValueOfRemoteFunctionCall(String aSource, Call aCall)

- Calls return in same order as they wait
- return next value in global (synchronized) bounded buffer
- These would occur in same order at a particular callee
- As long as caller and callee sites serialize them

processReturnValue(String aSource, Object aMessage)

- Concurrent calls can be made to different callees
- Put return value in global bounded buffer?
**Per Callee Bounded Buffer**

**Object `returnValueOfRemoteFunctionCall(String aSource, Call aCall)`**

Calls return in same order as they wait

Local bounded buffer $\leftarrow$ lookup(aDestination)

return next value in local bounded buffer

These would occur in same order at a particular callee

As long as caller and callee sites serialize them

**processReturnValue(String aSource, Object aMessage)**

Concurrent calls can be made to different callees

Local bounded buffer $\leftarrow$ lookup(aSource)

Put return value in local bounded buffer
METHOD RETURN VALUE

RPCReturnValue

Object returnValue

No Id sent to match with call
SUMMARY OF METHOD CALL HANDLING

- Synchronization needed between caller and callee.
  - In GIPC for function calls.
  - In RMI for both function and procedure calls.
- Cannot use shared data structures for synchronization, so use internal messages generated by the RPC system:
  - AnRPCReturnValue in GIPC,
- After sending call message caller invokes a method to wait for synchronization message from callee
  - In GIPC by blocking on a bounder buffer associated with the callee site
- On receiving synchronization message, callee site produces the return value in the bounded buffer associated with callee site