Communication

An Overview

Joseph Kehoe[1]

[1]Department of Computing and Networking
Institute of Technology Carlow

CDD101, 2018

Table of Contents

Table of Contents

Communication Overview

Computers in a distributed system may have different:

- Architectures
- Operating Systems
- Data encoding
- programming languages

We need a system of communication that allows them to talk to each other transparantly

Table of Contents

Networking Protocols

Communication between computers is build on existing network protocols

- At the base we have TCP/IP or UDP/IP
- We can use socket programming to use these directly
- Or we can use higher level protocols instead of or alongside this: e.g. HTTP
- This base level protocol is available to all computers
- But we will need to add a layer on top of this base in order to give us the extra features we need

I am going to assume you are familiar with these lower level protocols

Table of Contents

Institiúid Teicneolaíochta Cheatharlach
INSTITUTE of TECHNOLOGY CARLOW
At the Heart of South Leinster

RPC

Allow code to call procedures not resident on the local machine

- Causes a procedure to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction.
- The programmer writes essentially the same code whether the subroutine is local or remote
- Suited to Client Server Architectures
- RPC is a form of Inter Process Communication (IPC)
- RPC is usually synchronous in nature

Institiúid Teicneolaíochta Cheatharlach
INSTITUTE of TECHNOLOGY CARLOW
At the Heart of South Leinster

RPC

Sequence of events

- The client calls the client stub. The call is a local procedure call, with parameters pushed on to the stack in the normal way.
- The client stub packs the parameters into a message and makes a system call to send the message. Packing the parameters is called marshalling.
- The client's local operating system sends the message from the client machine to the server machine.
- The local operating system on the server machine passes the incoming packets to the server stub.
- The server stub unpacks the parameters from the message. Unpacking the parameters is called unmarshalling.
- Finally, the server stub calls the server procedure. The reply traces the same steps in the reverse direction.

Institiúid Teicneolaíochta Cheatharlach
INSTITUTE of TECHNOLOGY CARLOW
At the Heart of South Leinster

IDL

To let different clients access servers, a number of standardized RPC systems have been created.

- Most of these use an interface description language (IDL) to let various platforms call the RPC
- The IDL files can then be used to generate code to interface between the client and servers.

- XML-RPC is an RPC protocol that uses XML to encode its calls and HTTP as a transport mechanism.
- JSON-RPC is an RPC protocol that uses JSON-encoded messages
- JSON-WSP is an RPC protocol that uses JSON-encoded messages
- SOAP is a successor of XML-RPC and also uses XML to encode its HTTP-based calls.

RPC

- Packaging of parameters for transport to remote server is called **Marshalling**
- Parameters are either:
  - Pass by Value
  - Pass by Reference
  - Pass by copy Restore
- Passing pointers is obviously very difficult

Table of Contents

ROI

- Extends RPC to objects
- Improves on RPC
- Objects seperate state from interface (perfect for distributed Objects!)
- Keep state on server and put Interface on client
- Local proxy implements interface and holds reference to remote object
- It marshals data before sending, sends data, receives return values and unmarshalls them

ROI (on server)

- Server has a server "stub" known as a skeleton
- This receives data, unmarshals it and calls the server object implementation. It then marshalls the return values and sends them back to the client proxy
- Often use the **Object Adapter** pattern to wrap an object interface around an existing system
- Object is transient if it cannot be offloaded from server. So if server goes down object is lost

ROI (Objecct References)

- Object is persistent if it can be offloaded (saved) from the server to backup storage.
- This allows the object to be reanimated if the server crashes
- This even allows us to move the object around if we have a system wide object reference
- Need a DNS for object references if this is to work.

ROI (Dynamic Invocation)

- If IDL is compiled at runtime then we have **Static Object Invocation**
- If we can decode interface at runtime then we have **Dynamic Object Invocation**
- Static Call is: $accountRef->deposit(500);$
- Dynamic Call is:$sendMessage(accountRef, deposit, 500)$

Institiúid Teicneolaíochta Cheatharlach
INSTITUTE of TECHNOLOGY CARLOW
At the Heart of South Leinster

ROI (efficiency)

- Remote Object Innvocation can be inefficient
- Esp. if object state is small or parameters are large
- If system can distinguish between local and remote objects then we can circumvent this

Institiúid Teicneolaíochta Cheatharlach
INSTITUTE of TECHNOLOGY CARLOW
At the Heart of South Leinster

Java RMI

- Integrated into the language (high level approach)
- Cloning of remote objects is difficult so local proxy is not cloned during process, just the remote obejct
- We must explicitly get a new reference to the remote cloned object
- Remote objects cannot be **synchronised** - only the proxy can be
- Why? hint: Client crashes during operation
- Anything serialisable can be marshalled
- Proxys can be serialised (so we can pass them around)

See here

Table of Contents

Synchronous and Asynchronous Communication

Persistemt Asynchronous Receiver need not be running when message is sent. Sender is not blocked

Persistent Synchronous Receiver need not be running when message is sent. Sender is blocked until acknowledgement is send from system receiving message (possibly before receiver starts running).

Transient Asynchronous Sender is not blocke but can only send when receiver is running.

Institiúid Teicneolaíochta Cheatharlach
INSTITUTE of TECHNOLOGY CARLOW
At the Heart of South Leinster

Synchronous and Asynchronous Communication

Receipt-based transient Synchronous Receiver must be running for message to be sent and sender is blocked until receiver sends acknowledgement of receipt.

Delivery-based transient synchronous Receiver must be running before message can be sent. Sender is blocked until receiver accepts message and starts working on it.

Response-based transient synchronous Receiver must be running before message is sent. Sender is blocked until request is processed by receiver.

Messaging

- Persistent Asynchronous Messaging
- Message can be send sucessfully even if receiving object/system is not be running at the time
- Sender sends and "forgets"
- Message transfer can take "minutes"
- Leads to loosely coupled systems
- Guarantees delivery but does not guarantee it will be "read"
- Suited to distributed systems (why?)
- SMTP is a good example (email)

Message Passing Interface

- MPI is a standardized and portable message-passing standard designed by a group of researchers from academia and industry to function on a wide variety of parallel computing architectures
- The standard defines the syntax and semantics of a core of library routines in C, C++, Fortran and Python
- Commonly used on clusters

MPI Primitives

MPI_BSEND Append outgoing message to local send buffer

MPI_SEND Send a message and wait until copied to local or remote buffer

MPI_SSEND Send a message and wait until receipt starts

MPI_SENDRECV Send a message and wait for reply

MPI Primitives

MPI_ISEND Pass reference to outgoing message, and continue

MPI_ISSEND Pass refence to outgoing message , and wait until receipt starts

MPI_RECV Receive a message, block if there is none

MPI_IRECV Check if there is an incoming message, but do not block

see here