Curso LINUX

AREA 1: Intercomunicación

Intercomunicación de procesos

- Introducción
- Memoria compartida (IPC)
- Pipes
- RPC
- CORBA
- RMI
- DCOM
- SOAP / Web Services

Introducción

- En entornos multiproceso se hace necesario comunicar éstos de alguna manera
- Los procesos, idealmente, podrían ejecutarse en máquinas distintas
- Hay métodos de intercomunicación para procesos remotos o locales
- Cada método propone una abstracción distinta para comunicar procesos

Pipes

- Una de las maneras más sencillas de comunicación
- Un pipe (|) permite redirigir la salida estandar de un proceso a la entrada estándar de otro
- Permite “encadenar” acciones
- Tiene utilidad limitada
- Uso restringido a scripts shell
- Realmente útil para crear comandos propios

IPC : Memoria Compartida

- Compartir memoria es un sistema eficiente de intercambiar datos
- Solo está disponible en local (procesos en la misma máquina)
- La cantidad total de memoria compartida está definida en el inicio del sistema y no se puede incrementar sin reiniciar la máquina
- `/etc/sysctl.conf` (`kernel.shmax`)

Memoria Compartida

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

key_t key; /* clave a pasar a shmget() */
int shmflg; /* shmflg a pasar shmget() */
int shmid; /* valor retornado shmget() */
int size; /* tamaño requerido shmget() */

key = ...
size = ...
shmflg) = ...

if ((shmid = shmget (key, size, shmflg)) == -1) {
   perror("shmget: shmget falló"); exit(1); } else {
   (void) fprintf(stderr, "shmget: shmget devolvió %d\n", shmid);
   exit(0);
  }
...
```

Adjuntando memoria compartida

- `shmat()` y `shmdt()` se usan para adjuntar o liberar memoria compartida
- `void *shmat(int shmid, const void *shmaddr, int shmflg);`
- `shmat()` devuelve un puntero al inicio del segmento de memoria compartida asociada con el shmid.
- `int shmdt(const void *shmaddr);`
- `shmdt()` libera la memoria apuntada por shmaddr

Ejemplo

- 2 programas:
- shm_server.c : crea una cadena de texto y una porción de memoria compartida
- shm_client.c : adjunta la porción de memoria e imprime la cadena

shm_server.c

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>

#define SHMSZ     27

main()
{
    char c;
    int shmid;
    key_t key;
    char *shm, *s;

    key = 5678;

    if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0) {
        perror("shmget");
        exit(1);}

    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1); }

    s = shm;
    for (c = 'a'; c <= 'z'; c++)
        *s++ = c;
    *s = NULL;
    while (*shm != '*')
        sleep(1);
    exit(0);
}
```

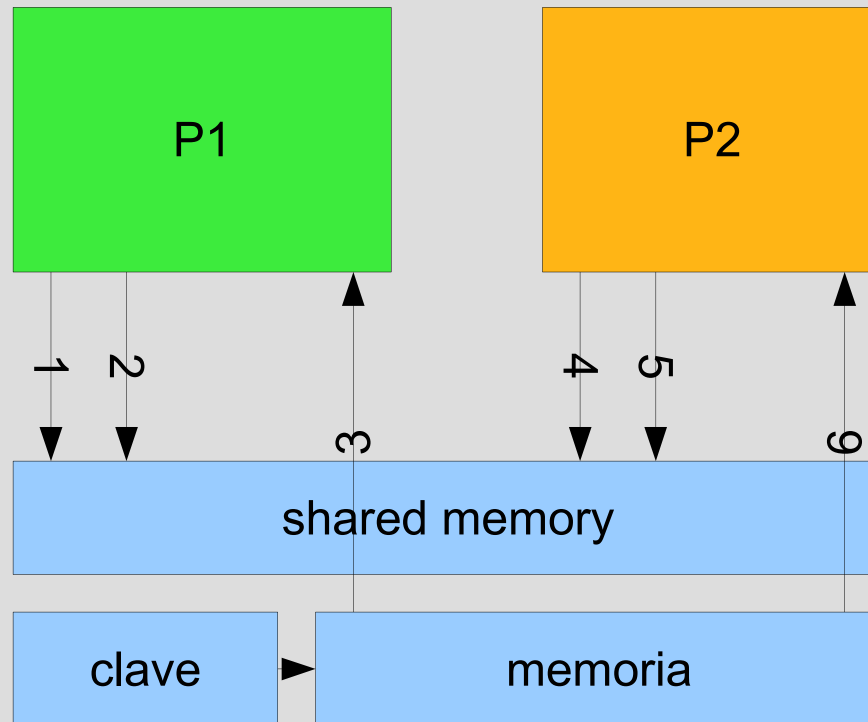
shm_client.c

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>

#define SHMSZ     27

main()
{
    int shmid;
    key_t key;
    char *shm, *s;

    key = 5678;

    if ((shmid = shmget(key, SHMSZ, 0666)) < 0) {
        perror("shmget");
        exit(1);
    }

    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
    }

    for (s = shm; *s != NULL; s++)
        putchar(*s);
    putchar('\n');
    *shm = '*';
    exit(0);
}
```

Ejercicio

- Modificar el shm_server para que acepte un texto por el teclado y sea el que ponga en la región compartida. Cada vez que se vuelva a poner a NULL que vuelva a preguntar
- Modificar el shm_client para que esté leyendo continuamente la memoria compartida y escriba siempre que encuentre una cadena != NULL, poniendola inmediatamente a NULL

IPC - Esquema



1. shmget
2. shmat
3. acceso a memoria

4. shmget
5. shmat
6. acceso a memoria

IPC Notas

- Permite comunicar solo procesos en la misma máquina y mismo espacio de direcciones
- Hay que controlar el acceso simultaneo (semáforos, mutex)
- Es el sistema más rápido de comunicación entre procesos, pero puede ser costoso en memoria

Invocación remota

- Usualmente es necesario interactuar con procesos de otras máquinas
- La primera solución fue crear servidores y protocolos que permitieran escuchar / ejecutar
- Cada protocolo era distinto y había que codificar la parte de comunicaciones en cada proceso

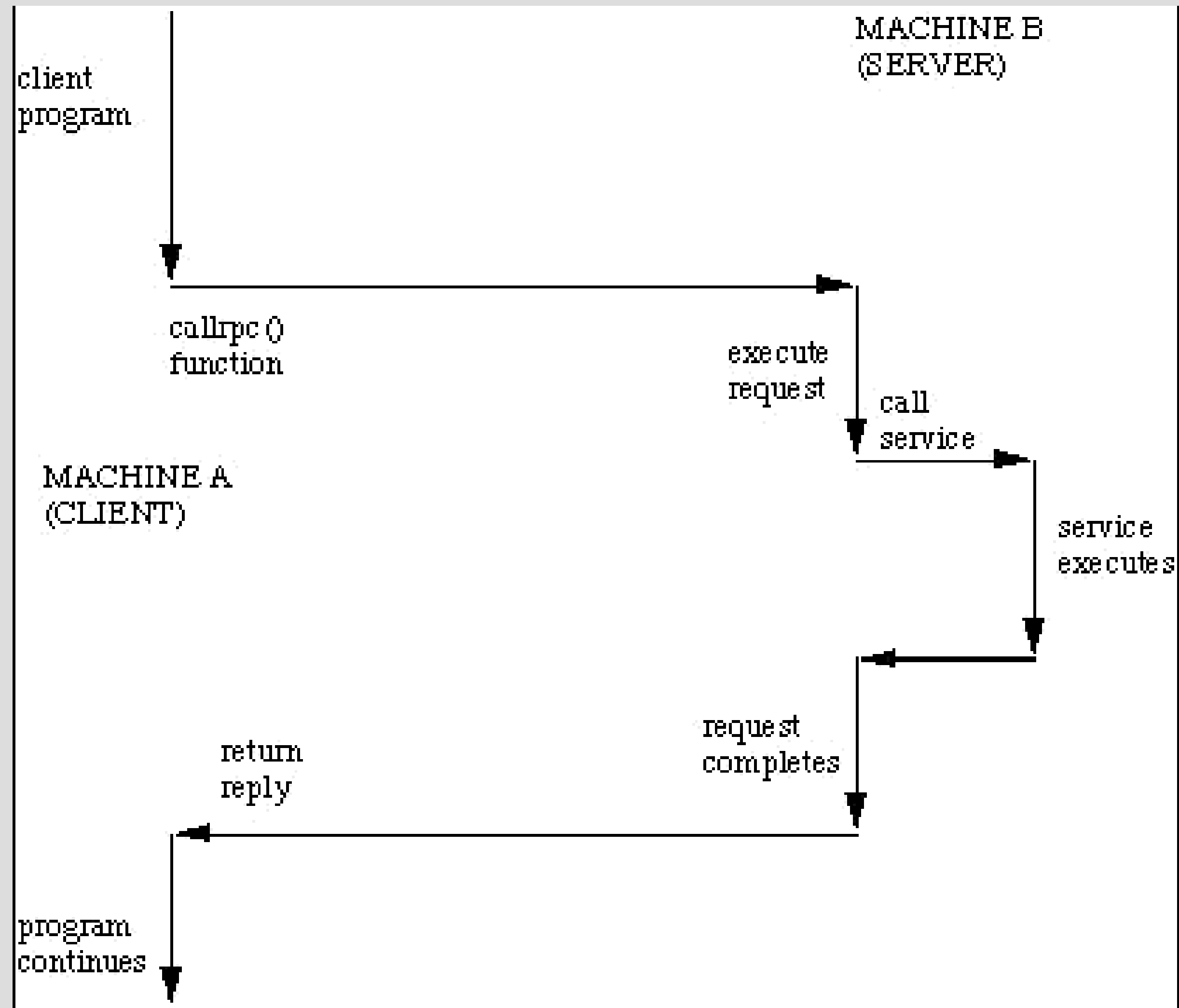
Invocación remota

- Viendo la necesidad de estándares, se propusieron varios modelos de ejecución remota
- El objetivo es “abstraer” la capa de comunicaciones y permitir invocar métodos o funciones residentes en otras máquinas
- También hay que tener en cuenta el intercambio de datos

RPC

- Remote Procedure Call, Llamada a Procedimiento Remoto
- protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos.
-

Esquema RPC



Desarrollo RPC

- Para desarrollar una aplicación RPC se necesita:
  - Especificar el protocolo para la comunicación cliente - servidor
  - Desarrollar el programa cliente
  - Desarrollar el programa servidor
- Los programas se compilarán por separado e incluirán las librerías RPC

RPC : definir el protocolo

- Existen herramientas para compilar protocolo RPC (`rpcgen`)
- Se debe identificar:
  - nombre de los procedimientos
  - tipos de los datos
- `rpcgen` utiliza su propia sintaxis y compila programas `.x`

rpcgen

- `rpcgen rpcprog.x` genera:
- `rpcprog_clnt.c` : client stub
- `rpcprog_svc.c` : server stub
- `rpcprog_xdr.c` : filtros XDR (representación de datos)
- `rpcprog.h` : cabeceras XDR.

RPC: codificar cliente y servidor

- Compilar con `#include "rpcprog.h"`
- Crear cliente: rpcprog.c
- Crear servidor: rpcsvr.c
- Compilar:

```
gcc -c  rpcprog.c
gcc -c  rpcprog_clnt.c
gcc -c  rpcprog_xdr.c
gcc -o rpcprog rpcprog.o rpcprog_clnt.o rpcprog_xdr.c
gcc -c  rpcsvc.c
gcc -c  rpcprog_svc.c
gcc -o rpcsvc rpcsvc.o rpcprog_svc.o rpcprog_xdr.c
```

Ejemplo cliente

```
#include <stdio.h>
#include <utmp.h>
#include <rpc/rpc.h>
#include <rpcsvc/rusers.h>

main(int argc, char **argv)

{
   unsigned long nusers;
   enum clnt_stat cs;
   if (argc != 2) {
     fprintf(stderr, "usage: rusers hostname\n");
     exit(1);
    }

   if( cs = rpc_call(argv[1], RUSERSPROG,
          RUSERSVERS, RUSERSPROC_NUM, xdr_void,
          (char *)0, xdr_u_long, (char *)&nusers,
          "visible") != RPC_SUCCESS ) {
             clnt_perrno(cs);
             exit(1);
           }

   fprintf(stderr, "%d users on %s\n", nusers, argv[1] );
   exit(0);
}
```

Servidor RPC

- Registro de un servicio

```
rpc_reg(u_long prognum /* Server program number */,
        u_long versnum /* Server version number */,
        u_long procnum /* server procedure number */,
        char *procname /* Name of remote function */,
        xdrproc_t inproc /* Filter to encode arg */,
        xdrproc_t outproc /* Filter to decode result */,
        char *nettype /* For transport selection */);
```

RPC pros y contras

- Es un sistema potente diseñado para ser eficiente
- Es complicado de aplicar y muy dificil de depurar
- Varía entre versiones de linux, por lo que muchos programas han de reescribirse
- Solo permite C

DCOM

- Distributed Component Object Model
- Versión de Microsoft de RCP
- Añade comunicaciones a COM+
- Ha sido abandonado en favor de .net y webservices
- Alto acoplamiento, solo plataforma MS

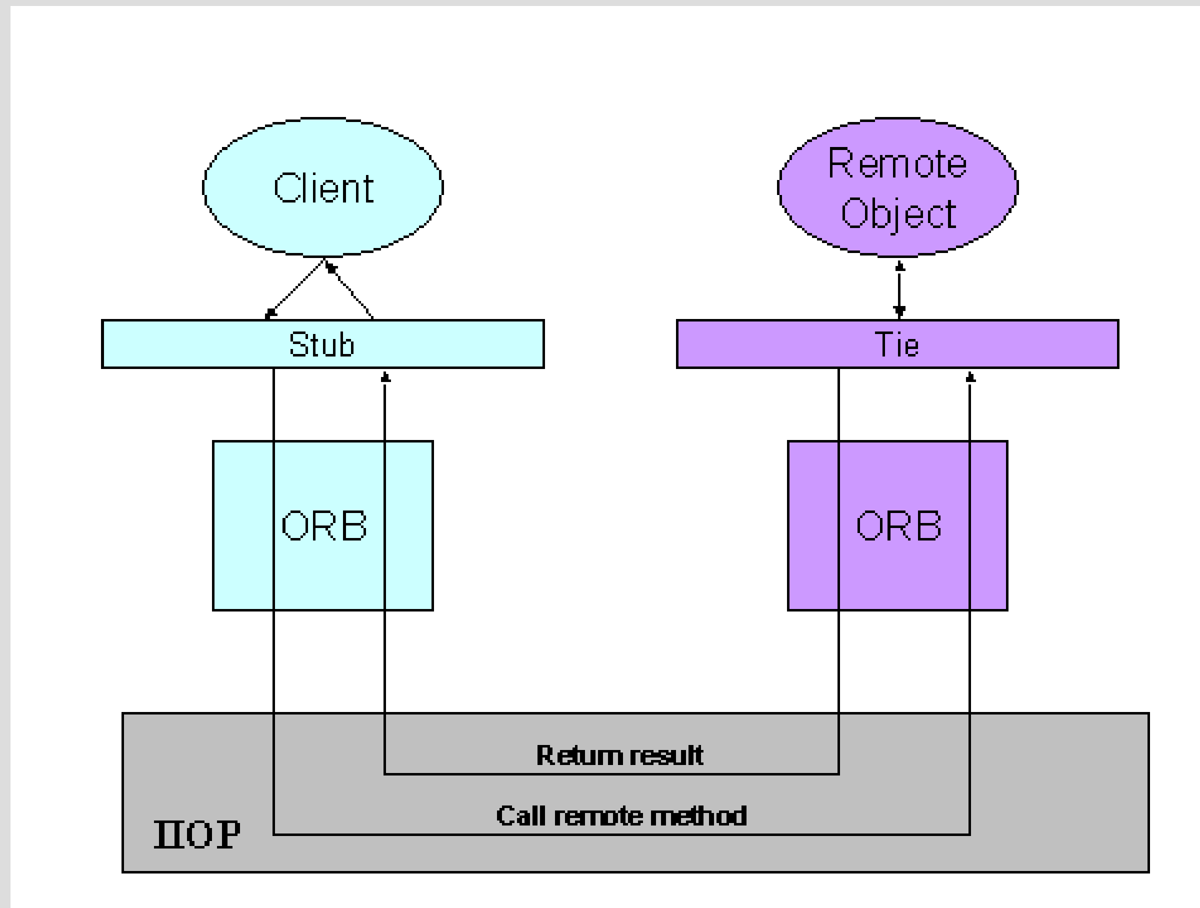
CORBA

- Common Object Broker Architecture
- Estandar para el desarrollo de aplicaciones distribuidas
- Definido y controlado por el Object Managemente Group (OMG)
- CORBA “envuelve” en una capa de comunicaciones el código

CORBA - historia

- 1991 – version 1.0
- 1992 – version 1.1
- 1993 – versión 1.2
- 1996 – versión 2.0
- 1997 – versión 2.1
- 1998 – version 2.2
- 1999 – v 2.3
- 2000 – v 2.4
- 2001 – v 2.5
- 2002 – v 3.0
- 2002 – v3.0.1 - 3.0.2

Funcionamiento CORBA



Programar en CORBA

- CORBA define uno formato de definición de interfaces (IDL)
- Una vez generado el IDL se utilizan herramientas del broker para generar:
  - stubs
  - skeletons

Ejemplo de IDL

```
interface Echo {
    string echoString(in string mesg);
};
```

- En el IDL se definen, de manera muy próxima al programador
  - interfaces
  - tipos de datos

Compilar idl

```
omniidl -bcxx echo.idl
```

- Compilador idl
- Genera archivos para stubs y skeletons
  - echo.h
  - echoSK.cc

Implementación del sirviente

```
class Echo_i : public POA_Echo,
        public PortableServer::RefCountServantBase
{
public:
  inline Echo_i() {}
  virtual ~Echo_i() {}
  virtual char* echoString(const char* mesg);
};

char* Echo_i::echoString(const char* mesg)
{
  return CORBA::string_dup(mesg);
}
```

Implantación del cliente

```
void
hello(CORBA::Object_ptr obj)
{
  Echo_var e = Echo::_narrow(obj);
  if (CORBA::is_nil(e)) {
    cerr << "cannot invoke on a nil object reference."
         << endl;
    return;
  }
  CORBA::String_var src = (const char*) "Hola!";
  CORBA::String_var dest;
  dest = e->echoString(src);
  cerr << "I said,\"" << src << "\"."
       << " The Object said,\"" << dest <<"\"" << endl;
}
```

Referencias

- Un cliente debe poseer la referencia del objeto remoto para poder llamarle
- Hay varias maneras de obtener la referencia
  - intercambio archivo IOR
  - servidor de nombres
- Obteniendo la referencia se puede acceder a cualquier elemento definido en el idl

Código cliente

```
int main(int argc, char** argv)
{
   CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

    if( argc != 2 ) {
      cerr << "uso:  cliente <ior>" << endl;
      return 1;
    }

    CORBA::Object_var obj = orb->string_to_object(argv[1]);
    Echo_var echoref = Echo::_narrow(obj);
    if( CORBA::is_nil(echoref) ) {
      cerr << "IOR mal." << endl;
      return 1;
    }
    for (CORBA::ULong count=0; count<10; count++)
      hello(echoref);

    orb->destroy();
  return 0;
}
```

Código servidor (ior)

```
int main(int argc, char** argv)
{
  CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

  CORBA::Object_var       obj = orb->resolve_initial_references("RootPOA");
  PortableServer::POA_var poa = PortableServer::POA::_narrow(obj);

  Echo_i* myecho = new Echo_i();

  PortableServer::ObjectId_var myechoid = poa->activate_object(myecho);

  obj = myecho->_this();
  CORBA::String_var sior(orb->object_to_string(obj));
  cerr << "'" << (char*)sior << "'" << endl;

  myecho->_remove_ref();

  PortableServer::POAManager_var pman = poa->the_POAManager();
  pman->activate();

  orb->run();
  orb->destroy();
  return 0;
}
```

Compilar

- `gcc echoSK.cc cliente.cc -I. -lomniORB4 -lomnithread -o cliente`
- `gcc echoSK.cc servidor.cc -I. -lomniORB4 -lomnithread -o servidor`

Ejecutar en distintas terminales

- `servidor` : copiar el IOR (sin las comillas)
- `cliente IOR` (IOR copiado de la salida del servidor)
- Comprobar la comunicación

Ejecutar en distintas maquinas

- `server` en maquina A
- copiar el IOR y pasarlo a otra máquina
- `cliente IOR` en maquina B

Multilenguaje

- Corba es multilenguaje, hay bindings para:
  - Ada
  - C / C++
  - Java
  - SmallTalk
  - Python
  - Perl
  - ...

Brokers / librerías

- omniORB
- orbit
- Mico
- Orbix
- Visibroker
- ... y cientos más

CORBA - Ventajas

- Independencia del lenguaje
- Independencia del sistema operativo
- Libre de tecnologías
- Fuerte tipado
- Posibilidad de ajuste de rendimiento
- Elimina los detalles de transmisión de datos

Desventajas

- Complicado de entender para novatos
- Problemas con el transporte de red
- Falta de apoyo de los desarrolladores y empresas
- Problemas en las implementaciones iniciales

Ejercicio

- Modificar el ejemplo anterior para que el cliente pida texto para mandar al servidor
- Hacer que el cliente recoja el IOR de un archivo y que el servidor lo escriba en un archivo (sin comillas).

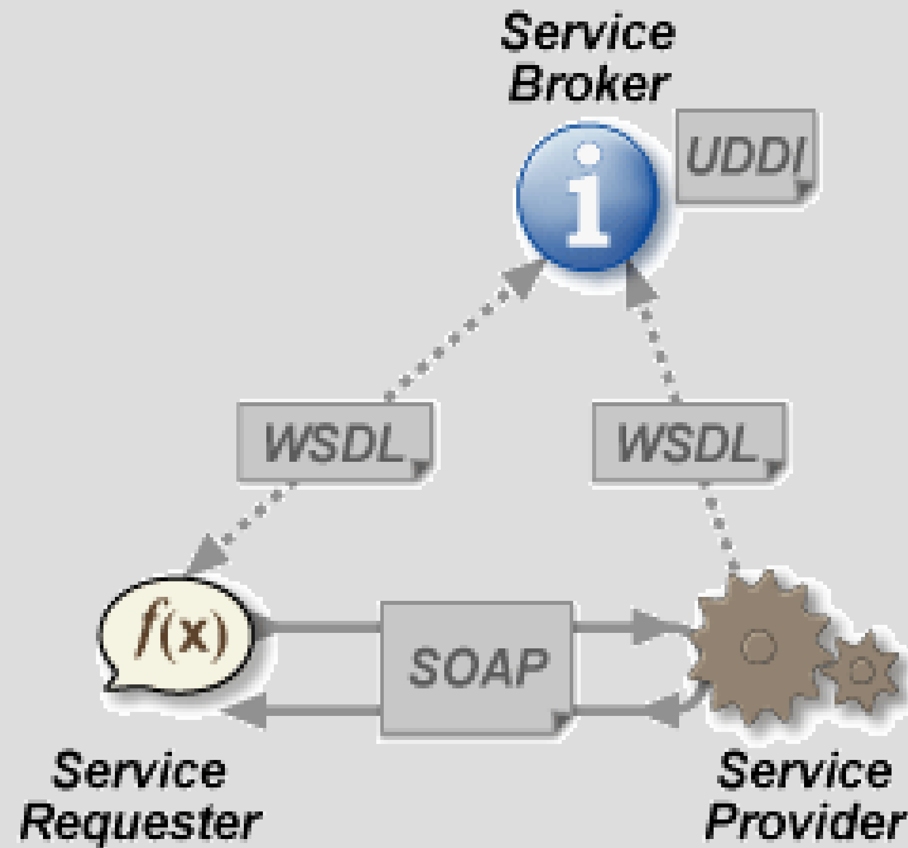
Web Services

- colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones
- Adopción (supuesta) de estándares abiertos

Estándares

- Web Services Protocol Stack
- XML
- SOAP o XML-RPC
- Otros protocolos: HTTP, FTP, o SMTP
- WSDL (Web Service Description Languaje)
- UDDI (Universal Description, Discovery and Integration)

Web Services - funcionamiento



Ventajas

- Interoperabilidad entre lenguajes
- Protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.
- Tolerantes a firewalls
- Combinación de servicios
- Uso de estándares

Inconventientes

- Pobre técnicamente
- Rendimiento bajo
- Dificil de asegurar al usar http
- Mala adopción de protocolos seguros

Técnicamente

- Definición de un WSDL para el interfaz común (debe ser conocido por cliente y servidor)
- Implantación de los parsers para el intercambio de archivos xml
- Implantación de servicio http para SOAP

WSDL (ejemplo)

Referencias

- www.cs.cf.ac.uk/Dave/C/node27.html
- www.linuxjournal.com/article/2204
- omniorb.sourceforge.net/docs.html
- msdn2.microsoft.com/es-es/webservices/default(en-us).aspx
- ws.apache.org/axis2