Mecanismes d'invocació

Leandro Navarro Moldes
Pedro A. García López

P07/11068/00082

UOC
www.uoc.edu

Índex

Introducció

En aquest mòdul didàctic veurem maneres diferents d'invocar els serveis distribuïts. Descriurem la idea bàsica d'invocació remota, que, d'una banda, comporta la problemàtica de com representar o codificar les dades per ser enviades en sèrie i, de l'altra, el protocol necessari per a tractar la seqüenciació d'accions necessària per a la invocació remota.

Distingirem entre els mecanismes que usen codificacions binàries com Corba, Java-RMI, DCOM, ONC-RPC i els mecanismes recents d'invocació remota fonamentats en codificacions textuals basades en l'XML i el transport HTTP: XML-RPC i SOAP.

S'introdueix també el concepte d'arquitectures orientades a serveis i s'insisteix en els serveis o mecanismes de localització remota. Convé tenir en compte, en totes aquestes tecnologies, aspectes com eficiència o economia en el volum de dades que cal transferir i interoperabilitat.

Per tal d'adquirir els coneixements, cal fer els petits experiments que trobareu en l'apartat d'activitats i en la web de l'assignatura i que ajuden tant a concretar les idees centrals, com a tenir experiències pròpies i personals dels fenòmens, tècniques i eines que es descriuen.

Objectius

Els objectius d'aquest mòdul didàctic són els següents:

1. Conèixer les diverses maneres que hi ha per a la invocació d'operacions entre processos d'un mateix context, o en un de diferent, l'organització, etc.

2. Conèixer les característiques i el funcionament de cada mecanisme.

3. Poder triar la millor opció en cada situació i valorar les implicacions del muntatge que caldrà fer.

1. El mecanisme d'invocació remota

El mecanisme bàsic per a la construcció de programes no trivials es basa en la invocació entre les parts d'un programa: les crides de funció, procediment o mètode d'un objecte. Al seu torn es basa en el mecanisme de crida de subrutina: el pas de paràmetres per una zona de memòria anomenada *pila* i la invocació d'una instrucció de llenguatge màquina de crida de subrutina. A causa del mecanisme que s'utilitza esdevé un mecanisme d'invocació local: només té sentit en un context de memòria-processador.

És còmode i temptador estendre aquest model per a la invocació al codi situat a les màquines separades d'una xarxa. La primera impressió és que programes, funcions o serveis d'una mateixa màquina es podrien trobar en màquines allunyades, o bé poder invocar funcions o serveis situats en altres màquines d'Internet.

El mecanisme d'invocació remota (en anglès, *remote procedure call* o RPC) és una extensió del mecanisme d'invocació de funció que permet d'invocar codis remots *com si fossin locals*. El procés client invoca una funció que sembla ser el codi servidor (*stub* client) però només recull i envia les dades d'entrada per xarxa a un altre codi del servidor (*stub* servidor) que invoca la funció sol·licitada. Els resultats de la invocació segueixen el camí invers.

Seria ideal que la separació no es notés (en diríem un *mecanisme transparent*, en el sentit que no es veu), però no és fàcil, ja que una xarxa com Internet té un comportament més complex que el bus intern d'un PC: es perden, desordenen i dupliquen paquets, i de vegades la xarxa falla.

Per a fer "transparent" la crida (invisible), fa falta introduir nous mecanismes per transportar la crida per la xarxa. Aquesta tasca la fan uns nous components anomenats *stub*.

Aquests estubs constitueixen el codi intermedi encarregat de gestionar connexions remotes entre el client i el servidor, i establir protocols de codificació i pas de paràmetres i resultats de la invocació remota. Els estubs són, així, els artífexs de la transparència d'accés i localització en el programari intermediari o *middleware* d'invocació remota de procediments.

Per a la generació d'aquests estubs, és necessari comprendre el paper de les interfícies (*interfaces*) remotes. En local, els llenguatges de programació organitzen el programa en mòduls que es comuniquen entre ells mitjançant trucades a procediments. Per a controlar les interaccions entre mòduls, la interfície de

cada mòdul defineix la signatura dels procediments o funcions que poden ser invocats.

En invocació remota, és imprescindible definir una interfície dels procediments o serveis que s'invocaran remotament. A partir de la informació de signatures i paràmetres d'aquesta interfície, les eines de generació de estubs podran crear el codi intermedi necessari per a fer aquestes trucades remotes transparentes al programador.

Segueixen aquest model d'invocació els mecanismes d'invocació d'ONC-RPC i RMI, que operen en un entorn homogeni: el mateix llenguatge de programació i, en el cas de Java-RMI, la mateixa màquina (la màquina virtual de Java).

Aquesta "separació" entre la màquina procés que sol·licita un servei i qui el du a terme possibilita la introducció de variants que permeten, en general, la interacció entre sistemes heterogenis en el següent:

- Arquitectura (format: mida i organització de dades),
- Llenguatge de programació (afecta la forma d'invocar i traslladar arguments),
- Entorn d'execució (sistema operatiu).

La publicació fundacional...

... d'invocació remota és l'article "Implementing Remote Procedure Calls" de Birrell, Nelson, etc. en el qual es descriu la primera realització d'un sistema d'invocació remota transparent. Cal destacar-ne les comparacions d'eficiència.

El suport a l'heterogeneïtat provoca una complexitat més gran de l'entorn. Es poden "controlar" més paràmetres, però el programador està obligat a tractar-hi: la invocació no és transparent, sinó que el programador ha d'escriure "força" (decidir diversos aspectes) per fer qualsevol invocació remota.

D'aquesta manera, en els mecanismes d'invocació remota que suporten l'heterogeneïtat, és usual la utilització d'un llenguatge de definició d'interfícies (IDL - *interface definition language*). Aquest IDL és independent del llenguatge de programació i a partir d'ell es podran crear generadors d'estubs per a diferents llenguatges de programació.

L'avantatge fonamental del suport a l'heterogeneïtat resideix en el fet que es podran comunicar programes escrits en diferents llenguatges de programació i que s'executaran en diferents sistemes operatius. Segueixen aquest model d'invocació en entorn heterogeni els mecanismes d'invocació de Corba i SOAP.

Es troben, doncs, dues classes de mecanismes d'invocació remota: els de programació senzilla (que persegueixen ocultar certs aspectes) per a sistemes distribuïts homogenis (casos: ONC-RPC per a llenguatge C, RMI per a Java), i els de programació complexa (ofereixen control de certs detalls), aptes per a sistemes distribuïts heterogenis (casos: Corba, DCOM, SOAP).

En un sentit ampli, el protocol HTTP que s'utilitza en la web també és una RPC: és un protocol de petició/resposta. Tots els navegadors invoquen les mateixes operacions als servidors: **doc=get(uri)**, **put(uri, doc)**, **post(uri, text)**, i les operacions d'edició que usen dades codificades en XML: **res=propfind(uri, args)**, **res=proppatch(uri, args)**, etc. Una generalització d'aquests mecanismes per a l'RPC és l'adoptat en l'XMLRPC o SOAP, per exemple.

Quins temes es poden destacar de la invocació remota?

Aspectes principals:

- representació de les dades (forma seqüencial o sèrie),
- mecanisme d'invocació (*stub* client i servidor).

Aspectes secundaris:

- Forma d'especificar la invocació remota (IDL estàtic o dinàmic, llenguatge),
- Serveis associats: noms, transaccions, etc.

Problemàtica:

- Pas de paràmetres (in, out, inout) i *stubs* (estàtics o dinàmics),
- Invocació de mètode: problemàtica de referències i pas d'objectes, gestió de memòria dinàmica ("recol·lecció de desferres")*, pas per objecte per valor o referència a l'objecte,
- Localització del servei (noms),
- Complexitat del model: interoperabilitat, heterogeneïtat i serveis.

*** En anglès:** ***garbage collection.***

La resta del mòdul s'estructura en dues parts: la manera com es poden preparar o codificar les dades per a la transferència, i els protocols d'invocació remota que permeten d'invocar una operació a distància.

En la codificació de dades s'examina el problema de com representar dades en forma de seqüència que es puguin enviar còmodament per la xarxa i que les pugui entendre qualsevol altra màquina de la xarxa. Hi ha diverses propostes amb objectius, compromisos, optimitzacions, complexitat, interoperabilitat diferents. Es tracta de conèixer les característiques, poder distingir un mecanisme d'un altre i entendre la utilitat de cada un, per poder triar en un cas real.

En els protocols d'invocació remota es vol automatitzar i coordinar el procés d'invocació d'operacions que està relacionat, com altres protocols de transport, a ordenar i controlar l'ordre i vigilar que ni la xarxa ni qualsevol de les dues màquines tinguin problemes de sobrecàrrega o que es cansin d'esperar.

Cada forma d'invocació remota té alhora un format de codificació de dades per a l'intercanvi i un protocol d'invocació.

En primer lloc, examinarem com es poden representar estructures de dades en seqüències per a ser enviades per una xarxa.

2. Codificació de dades per a l'intercanvi

Qualsevol procediment d'invocació remota necessita traspassar dades per la xarxa en forma de seqüència d'octets (bytes).

Marshalling* i *un-marshalling

El mecanisme de seriació i desseriació es denomina *marshalling* i *un-marshalling* en honor al general Marshall, un militar que va organitzar el desembarcament de Normandia de la Segona Guerra Mundial: les tropes i efectius militars es van disposar amb molta cura sobre la platja de Dover. Un mecanisme curós i ordenat de transport en sèrie per vaixell, va acabar reproduint la mateixa organització sobre la platja de Normandia.

Un tanc seria una estructura de dades en les nostres aplicacions, un vaixell seria un paquet IP, un soldat seria un enter, etc.

Aquest intercanvi és possible si hi ha un acord de representació i d'interpretació de les dades entre ambdues parts: les dades que es reben han de tenir sentit.

El sentit de les dades significa que s'han d'acordar dos aspectes:

- Mecanisme de codificació (i descodificació) dels arguments i els resultats: un procés que transforma la informació, expressada en forma d'estructures de dades, en una seqüència d'octets. Aquest procés rep el nom de *conversió*, *seriació* o *marshalling*.

- Interpretació: dependent de l'entorn de treball i que al seu torn està condicionat per l'arquitectura del computador, el sistema operatiu i el llenguatge de programació. Es complica quan les màquines que es comuniquen tenen característiques diferents.

Aquests aspectes formen un protocol pel que fa al transport que posa en comunicació dos agents situats als extrems de la xarxa, amb el mecanisme d'invocació (RPC) en lloc del model de "tub fiable" que ofereix TCP.

2.1. Problemàtica

És necessari seleccionar o convenir un format de representació de les dades que seran usades com a arguments i resultats de la invocació d'operacions remotes. Això inclou definir quins tipus s'utilitzaran: caràcter, enter, real, etc., l'*endianness* o l'ordre en què s'envien els bytes, quin repertori o joc de caràcters s'utilitzarà, i quines característiques tindrà el nostre llenguatge d'especificació de dades. Per exemple, si es podran definir tipus de dades que puguin usar punters i com fer correspondre els tipus de dades i les maneres de construir tipus de dades complexes amb el format escollit, amb els tipus de dades dels llenguatges de programació habituals.

Per exemple, en una aplicació per a la reserva de places en un avió podria fer falta representar tipus de dades com les següents:

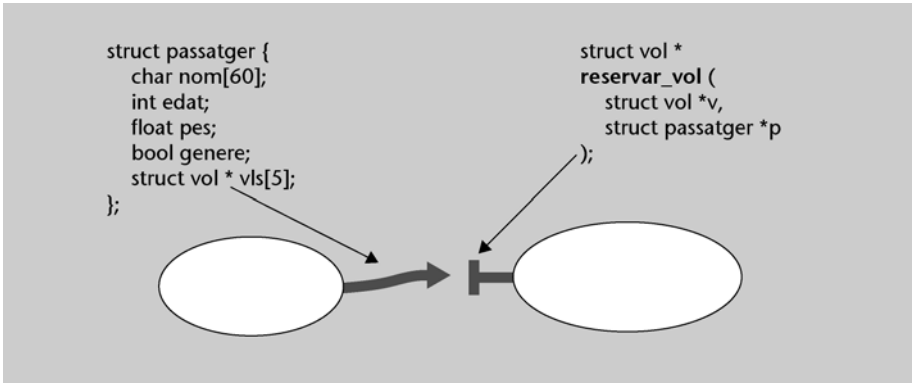


Figura 1. Tipus de dades necessàries per a una reserva de vol, expressat en llenguatge C

L'operació reservar_vol podria funcionar de manera natural en el llenguatge C amb tipus de dades que usin punters a zones de memòria de longitud variable. Aquestes dades presenten dificultats per a ser representades en "sèrie" i el llenguatge d'especificació de tipus de dades milloraria si obligués a especificar millor la mida exacta de les dades i així evitar ambigüitats o una tramesa de dades innecessària.

2.2. Representació externa

La representació externa és el conveni que se segueix per a representar estructures de dades durant el transport per la xarxa: per a convertir o seriar arguments i resultats.

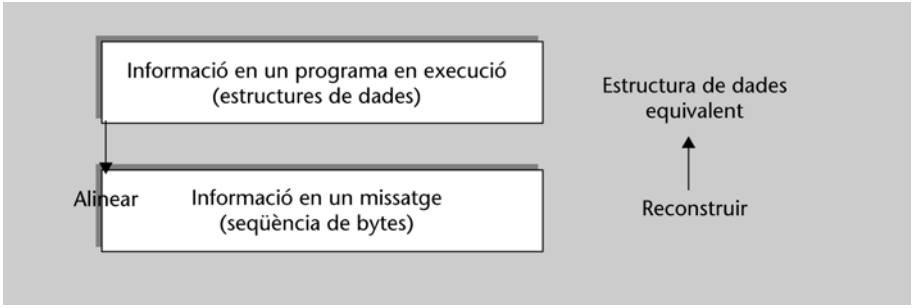


Figura 2. Processos de conversió de forma sèrie a representació en memòria

Els tipus de dades més habituals es corresponen amb els tipus de dades primitius dels processadors i dels llenguatges de programació més populars com el C o el C++.

- Tipus base: total, real, caràcter, enumerat,
- Tipus plans: agregacions, estructures, vectors (farciment),
- Tipus complexos: amb punters, per exemple, arbre (que s'ha "d'aplanar", etc.).

Hi ha diverses estratègies possibles de conversió de les dades representades en cada computador, del format intern a un format de sèrie, per a ser transferides. L'objectiu és de reduir la complexitat i el cost de conversió de dades:

- Enviar en el format intern de l'emissor.

- Transformar les dades per enviar-les en el format intern del receptor. En tots dos casos el problema és complex, ja que un dels extrems no ha de fer res mentre que l'altre hauria de saber com representar les dades per a qualsevol possible representació (arquitectura). No sembla una solució viable.

- Enviar en una forma canònica intermèdia (per exemple, com es representen les capçaleres IP que són *big-endian*). Cada computador només ha de saber com ha de fer la conversió entre el seu format intern i el format canònic.

- No conversió, si ambdós computadors són similars. En el cas anterior pot passar que dues màquines amb la mateixa arquitectura interna hagin de fer dues conversions innecessàries perquè les dades viatgin en el format canònic quan es podrien haver comunicat sense fer cap conversió. Si es pogués conèixer el format del computador de l'altre extrem, es podria optimitzar la transferència estalviant conversions supèrflues.

- En el format de l'emissor incloent una indicació de format, el receptor converteix. D'aquesta manera, l'emissor no ha ni de conèixer el receptor ni ha de fer mai cap conversió. La feina recau sobre el receptor que ha d'aplicar la transformació necessària per a cada format possible. En alguns casos es fixen els formats i les mides de les dades, i tot el que ha de fer el receptor és canviar l'ordre dels bytes en algunes dades (*endianness*).

2.3. *Big-endian* o *little-endian*

Endian: conveni sobre l'ordre de representació de dades de diversos bytes de longitud.

El terme s'utilitza per analogia amb la història del llibre *Els viatges de Gulliver* de Jonathan Swift, en què va imaginar una lluita interminable entre els regnes de Blefuscu, on trencaven els ous durs per l'extrem més ample –*big-endians*–, i el regne de Lilliput, en el qual estaven obligats per llei a trencar els ous durs per l'extrem més estret.

L'analogia en computació és si en els nombres s'ha de desar primer el byte més significatiu o el menys significatiu.

Com a la novel·la de Swift, no cal deixar-se endur pels fanatismes, són dues alternatives i el que és important és acordar d'usar-ne una.

- ***Little-endian***: el byte menys significatiu es guarda a l'adreça de memòria menor, que serà l'adreça de la dada. Són *little-endian* els processadors Intel 80X86, Pentium, Alfa, els sistemes operatius Windows, OSF/1 i diversos formats d'arxius.

- ***Big-endian***: el byte més significatiu es guarda a l'adreça de memòria menor, que serà l'adreça de la dada. Són *big-endians* els processadors Motorola 680x0, el PA-RISC de Hewlett-Packard i el SuperSPARC de Sun. Els processadors MIPS de Silicon Graphics i el PowerPC d'IBM/Motorola poden treballar com a *little-endian* i *big-endian* (*bi-endian*). El sistema operatiu d'Apple també és *big-endian*, i també la màquina virtual de Java. Internet també és *big-endian*: en TCP i en IP es transmet primer el byte més significatiu.

Quin és millor? Tots dos tenen avantatges i desavantatges:

– En la forma *little-endian* el byte de menys pes d'una dada d'1, 2, 4 o més bytes és sempre el primer (desplaçament 0), motiu pel qual les operacions matemàtiques són més fàcils de fer: en ordre creixent es van trobant els bytes per operar.

– En la forma *big-endian* el byte de més pes és sempre el primer, pel que es pot saber si el nombre és positiu o negatiu mirant el primer byte: es guarda tal com s'escriu, es van trobant bytes de pes menor.

Un arxiu o fitxer també és una representació externa de dades:

Per exemple, els fitxers amb extensió .bmp es van dissenyar en una arquitectura *little-endian* (Intel), per tant, en escriure un fitxer .bmp en una màquina amb enters *big-endian*, s'ha d'intercanviar l'ordre dels bytes, fet que significa una mica més de feina. En canvi, el fitxer es pot enviar a qualsevol altra màquina i es podrà obrir amb independència del tipus de màquina en què fou creat.

- Alguns formats de fitxers i el seu ordre:

Nom del format	*Little-endian*	*Big-endian*
Adobe Photoshop		✓
BMP (Bitmaps Windows i OS/2)	✓	
DXF (AutoCad): variable	✓	✓
GIF	✓	
JPEG		✓
FLI (Autodesk Animator)	✓	
MacPaint		✓
PCX (PC Paintbrush)	✓	
PostScript	No aplica: és text	

Nom del format	*Little-endian*	*Big-endian*
QTM (Pel·lícula Quicktime): en un MAC...	✓	
Microsoft RIFF (.WAV & .AVI)	✓	✓
Microsoft RTF (*rich text format*)	✓	
Sun Raster		✓
TIFF: identificació d'ordre al fitxer	✓	✓
WPG (WordPerfect Graphics Metafile): en un PC		✓
XWD (imatge de X Windows): identificació d'ordre al fitxer	✓	✓

2.4. Etiquetes

Cada valor d'un tipus de dades o una seqüència pot estar precedit d'unes etiquetes que aporten informació per a delimitar i identificar el valor i així ajudar a ser reconstruït: aquestes etiquetes poden indicar el tipus de valor, la longitud, o l'arquitectura (opció *d'endian*). Això permet d'evitar ambigüitats en la representació i poder enviar dades que no encaixen exactament en una plantilla prefixada.

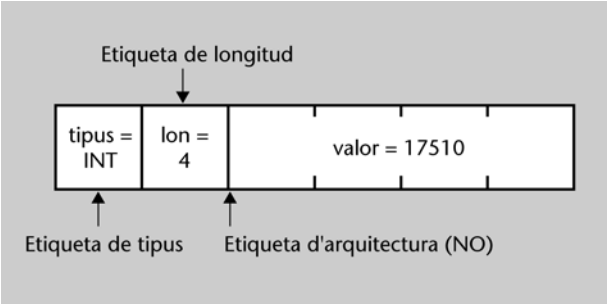


Figura 3. El valor sencer 17510 codificat amb 4 bytes.

2.5. *Stubs*

Els *stubs* són trossos de codi que s'encarreguen de seriar i desseriar dades. Aquest codi pot ser:

- estàtic: fix i compilat a partir d'una especificació de tipus de dades concreta (per exemple, un fitxer d'especificació d'interfície en un *interface description language* o IDL), o

- dinàmic: a partir de les dades que rep i la informació que porten inclosa (en forma d'etiquetes) converteix una seqüència en una estructura de dades i viceversa, sense necessitat de conèixer les dades prèviament. Aquest codi pot ser menys eficient, però també evita la dependència d'una especificació de tipus de dades prefixada. Aquests tipus de *stubs* s'usen per a les interfícies d'invocació dinàmica.

2.6. Altres usos de la representació externa

La representació externa de dades, en què les dades es representen en un format independent de la representació en memòria, té altres usos per a l'emmagatzematge: per a preservar la informació en un format permanent (guardar en un arxiu de disc, emmagatzemar en una base de dades, guardar en un disquet) i que també pugui ser transportat a una altra màquina.

2.7. Codificació textual o binària

De fet, el tipus universal de dades a Internet és el joc de caràcters ASCII, les lletres que s'usen en la llengua anglesa, nombres i alguns símbols de puntuació i de control. Molts protocols representen qualsevol informació en format textual.

El nombre π o una variable que té un valor "concret" es pot expressar així en un format textual (cada casella és un caràcter):

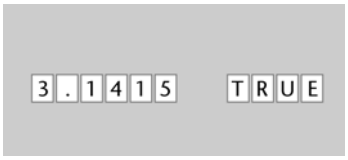


Figura 4

Codificació textual = tot s'ha de convertir a text, fins i tot les dades estructurades. És l'opció que usen protocols d'Internet com l'SMTP, el POP, el IMAP, l'HTTP (0.9; 1.0; 1.1).

És el més llegible i es prefereix per la seva facilitat a l'hora d'analitzar què passa quan alguna cosa falla, però té l'inconvenient que és molt prolífic (s'usen molts bits per a expressar una informació que moltes vegades es podria dir amb molts menys).

La codificació binària no busca la llegibilitat sinó l'eficiència: minimitzar el nombre de bits que ocupa la informació i/o minimitzar el temps de conversió de les dades entre el format intern (privat) i l'extern (convingut pels participants en la comunicació).

Per a codificar dades binàries en text...

... (passar de 8 → 7 bits), s'usa l'algorisme **base64** que selecciona grups de 6 bits i els assigna una lletra d'un conjunt de 64 caràcters (a-zA-Z0-9+/): per cada 3 bytes genera 4 lletres.

És l'opció que usen protocols d'Internet com el DNS, l'LDAP, l'SNMP, l'NFS, l'H.323.

En totes dues codificacions sorgeix el problema de la codificació de l'estructura de la informació:

- Delimitació d'elements: com marcar el final o l'inici d'un element en una seqüència.

- Tipus d'informació (i interpretació): com expressar quina dada és cada component d'una col·lecció: sobretot si n'ometem alguna en enviar-les o el seu ordre és variable.

- Estructuració (pertinença a un conjunt): com indicar que certa dada pertany a una agrupació.

2.8. Delimitació dels elements

Qualsevol codificació ha de determinar una manera de separar les parts que componen una seqüència de dades. És a dir, com es determina la longitud d'una dada?

Algunes opcions habituals en protocols d'Internet són les següents:

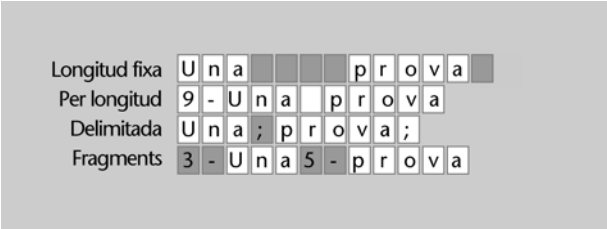


Figura 5. Formes de delimitar dades

- Longitud fixa: compacta però inflexible. Precisament, l'efecte 2000 és una conseqüència d'usar dades de longitud fixa. Només sembla recomanable per a dades de longitud sempre fixa, ja que en dades de longitud variable, o es desaprofita molt d'espai o es corre el risc de no poder expressar alguna dada, i no hi hauria manera d'arreglar-ho en ometre la longitud: és un conveni entre els participants en una comunicació: tots haurien d'acordar el canvi per a fer ajustos.

- "Per longitud": millora l'anterior, però s'ha de saber la longitud de bon principi. Això pot ser un inconvenient per al productor de la informació, ja que podria ser necessari haver de generar totes les dades per a poder calcular la longitud i enviar aquesta primera dada. En canvi, el receptor sap primer quant li ocuparà la dada i després pot llegir exactament els bytes indicats sense dificultat.

- "Per longitud a fragments": s'usa quan la longitud total de la dada s'ignora des del començament. Permet d'anar enviant trossos de dades tal com es van produint, fet que permet que les dades es comencin a emetre abans, i que el productor pugui estalviar memòria dedicada a mantenir dades preparades per a la tramesa.

- "Delimitada": En lloc de calcular la longitud, es reserva un símbol per separar dades. Si el delimitador apareix entre les dades, s'ha de definir una manera d'"ignorar-lo". Per exemple:

Caràcter reservat d'un llenguatge de programació: \ "	AB\CD; → AB\\CD\; I "F → E""F
Correu electrònic: capçalera MIME	aquí→=?iso-8859-1?Q?aqu=ED?=
MIME: *Quoded Printable*	Ä → =C4
URL de web	Ä → %C4
Caràcter reservat HTML o XML	<a> → <a>
Entitat caràcter d'HTML o XML	Ä → Ä Ä → Ä

En general, d'una dada se'n pot expressar: una etiqueta o identificador que distingeix la seva presència de la resta de dades, la longitud de la dada si no s'usen delimitadors, el valor que té.

Per a minimitzar la informació que s'envia per la xarxa, una o diverses d'aquestes dades es podrien ometre:

- Si totes les dades d'un conjunt apareixen sempre i en el mateix ordre, l'etiqueta es pot ometre.

- Si els participants en la comunicació coneixen la longitud per endavant, es pot ometre en la informació que s'envia.

- Si els participants coneixen el valor prèviament, és a dir, pren el valor per defecte, es pot ometre.

La taula següent mostra com es codificaria una estructura de dades que tingués informació de persones:

```
typedef struct persona {
    int edat;
    float pes;
    struct {
        char *nom;
        char *cognom;
    } nom_complet;
};
```

La informació corresponent de l'etiqueta-longitud-valor seria:

Element	Part	Octets	Codificació
Edat	Etiqueta	1	El valor "0" representa "Edat" en aquest protocol
	Longitud	1	Sempre 1
	Valor	1	Valor binari
Pes	Etiqueta	1	El valor "1" representa "Pes" en aquest protocol
	Longitud	1	Sempre 1
	Valor	4	Primer octet exponent base 10,3 octets amb mantissa, tots dos en format binari

Element		Part	Octets	Codificació
Nom complet		Etiqueta	1	El valor "2" representa "Nom complet" en aquest protocol
		Longitud	1	La longitud total dels components
Components del nom	Nom	Etiqueta	1	El valor "3" representa "Nom" en aquest protocol
		Longitud	1	La longitud del text
		Valor	segons el text	ISO 8859-1
	Cognom	Etiqueta	1	El valor "4" representa "Cognom" en aquest protocol
		Longitud	1	La longitud del text
		Valor	segons el text	ISO 8859-1

La informació precedent es podria haver expressat de manera textual (tal com es codifiquen els missatges de correu electrònic):

Edat: 32
Pes: 64
Nom: Perez, Joan

3. Formats de codificació de dades

Hi ha molts formats de dades per al transport, també anomenats *wire formats*. Aquests formats que s'han especificat i s'utilitzen àmpliament permeten de superar la diversitat de llenguatges, sistemes operatius i màquines. Els formats més rellevants són els següents:

- Textual:
  - XML: s'usa en l'XML-RPC, el SOAP.
  - JSON: s'usa en aplicacions JavaScript que s'executen en un navegador.

- Binaris:
  - *External data representation* (XDR): s'usa en l'RPC (v2; ONC-RPC).
  - *Abstract syntax notation* (ASN.1): s'usa en diversos protocols com, per exemple, X.509, l'SNMP.
  - *Network data representation* (NDR): s'usa en el DCOM, el DCE-RPC.
  - *Common data representation* (CDR): s'usa en el CORBA (GIOP, IIOP).
  - *Java remote messaging protocol* (JRMP): s'usa per a comunicar màquines virtuals Java (Java-RMI).

3.1. Codificació textual amb documents XML

El llenguatge XML (*extensible markup language*, 'llenguatge extensible de marques') és, segons el Consorci Web, el format universal per a documents estructurats i dades al Web. Per això, s'ha convertit en un estàndard massivament utilitzat per a la codificació de dades interoperables a Internet.

Diversos protocols usen l'XML per a codificar (seriar) les dades que s'intercanvien: per exemple, WEBDAV, que és una extensió del protocol HTTP per a tractar un servidor web com si fos un servidor de fitxers en xarxa, o el SOAP, que és un mecanisme d'invocació remota d'operacions (també conegut com a *serveis web*).

Tanmateix, l'XML té diferents restriccions:

- És textual i les dades binàries s'han d'enviar codificades en format Base64 o s'han d'enviar a banda del document XML (usant un enllaç, com fa l'HTML per a les imatges).

- Pot requerir bastant text, encara que després es pot comprimir usant un compressor general com gzip o compress, o amb un compressor específic per a l'XML com el que estudia el grup de treball sobre caracterització binària de l'XML.

En l'exemple següent podeu veure un document XML que es transporta en el protocol HTTP per a invocar una operació d'un servei web en el protocol XML-RPC.

```
<?xml  version="1.0"?>
<methodCall>
    <methodName >buscar</methodName>
    <params>
        <param>
         <value>
            <struct>
                <member> <name >nom</name > < value >Joan</value > </member>
                <member> <name>edat</name > <value><i4> 42 </i4></value>  </member>
            </struct>
         </value>
        </param>
    </params>
</methodCall >
```

Podeu veure que no és una representació compacta però sí que és molt informativa i fàcil d'entendre. En el cas de l'XMLRPC, els tipus dels paràmetres s'especifiquen com a etiquetes XML, com la i4 per a tipus enters que observem en l'exemple anterior. En canvi, altres mecanismes d'invocació més complexos com el SOAP s'aprofiten més de les possibilitats de l'XML.

Més concretament, un dels punts a destacar dels documents XML és la **validació** o restricció de documents sobre la base de contractes establerts. En general, podem dir que un document que compleix la sintaxi de l'XML es diu que està "ben format". Si a més el document compleix un conjunt de restriccions addicionals especificades per cert contracte, es pot dir que és "vàlid".

Les característiques del llenguatge del contracte, la seva gramàtica (elements d'una llengua i les seves combinacions), es poden expressar en una secció del document, o en un document a banda, de dues maneres:

1) DTD o definició del tipus de document, que és l'únic format que existia al començament de l'XML. La sintaxi d'un DTD no és XML, i manca de tipus de dades per a restringir els valors.

2) Esquema XML. És un format més recent, basat en l'XML, amb tipus de dades i capacitat d'expressar més restriccions. És preferible davant els DTD i ens hi centrarem a partir d'ara.

L'esquema XML és una eina molt potent que permet definir tipus de dades i imposar una sèrie de restriccions a les dades del document. Així, l'XML Schema permet les restriccions següents:

- Control sobre les dades: longitud *–length–* (nombre de caràcters de la cadena, binari), *maxlength* (màxima longitud de la cadena, binari), *lexical representation* (representacions possibles, per exemple, DD-MM-AAAA),

enumeration, *maxInclusive* (màxim possible), *maxExclusive* (màxim exclusiu), *minInclusive* (mínim possible), *minExclusive* (mínim exclusiu), *precision* (nombre de dígits), *scale* (dígits amb part decimal), *encoding* (binari).

- Control sobre el refinament: mecanismes d'herència i extensió.

- Control sobre l'extensibilitat: oberta, refinable, tancada.

A més, l'XML Schema suporta els tipus de dades següents:

String (seqüència de caràcter, iso 10646, unicode), boolean, real, decimal (real, no Exp), integer [–T,0,T], non-negative integer[0,T], positive integer [1,T], non-positive integer [-T,0], negative integer [-T,-1], date-Time (iso8601), date (dateTime), time (dateTime), timePeriod (dateTime), binary (datos+codif hex/base64), uri (rfc2396), language (es, ca, en, rfc1766), a més de permetre els tipus definits per l'usuari.

Aquestes característiques de l'XML i el seu XML Schema el fan idoni com a llenguatge de codificació de dades interoperable. En aquesta línia, s'han creat molts contractes basats en esquemes XML com el MathML, BioML, Bizztalk o el SOAP entre d'altres.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <SOAP-ENV:Body>
            <ns1:echoString xmlns:ns1="http://soapinterop.org/">
                    <arg0 xsi:type="xsd:string">Hello!</arg0>
            </ns1:echoString>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Com veiem en aquest exemple d'invocació remota utilitzant el SOAP, els paràmetres de la invocació utilitzen tipus de l'esquema XML (xsd:string). Cada vegada que es produeix una invocació remota amb el SOAP, s'ha de generar un document XML que compleixi l'esquema i on es produeixi el mapatge de tipus entre el llenguatge de programació i la definició XML. El procés de codificació SOAP (*SOAP encoding*) defineix bàsicament aquest mecanisme de conversió entre tipus del llenguatge de programació i tipus XML, incloent-hi tipus bàsics, compostos, *arrays* i tipus del llenguatge com referències o classes.

3.2. Codificació textual JSON (*JavaScript object notation*)

Encara que de molta menys importància que l'XML com a llenguatge d'intercanvi de dades, el JSON ha adquirit certa rellevància com a format lleuger per a l'intercanvi de dades en entorns AJAX. El JSON és un subconjunt de la nota-

ció literal d'objectes del Javascript, però no està restringit a aquest llenguatge, sinó que és usat per multitud d'entorns de programació.

Un dels avantatges del JSON sobre l'XML com a format d'intercanvi de dades és que escriure un analitzador semàntic del JSON és molt més senzill. En el Javascript, el JSON es pot analitzar trivialment usant el procediment `eval` i per això s'ha popularitzat en entorns web d'invocació remota com l'AJAX. També s'ha utilitzat com a llenguatge de codificació de serveis REST (*representational state transfer*) i en mecanismes d'invocació com ara el JSON-RPC.

De tota manera, el JSON no especifica tipus de dades ni permet restriccions com l'XML i el seu XML Schema. És simplement un format textual per a agrupar i estructurar continguts. Tanmateix, la seva simplicitat i rapidesa fan que l'utilitzin servidors de Google o Yahoo, on la mida del flux de dades entre client i servidor és molt important.

--> {"method": "postMessage", "params": ["Hello all!"], "id": 99}
<-- {"result": 1, "error": null, "id": 99}

Exemple del JSON-RPC:

AJAX SideBar

Asynchronous JavaScript and XML: és una tècnica de desenvolupament d'aplicacions web interactives. Mitjançant l'ús de comunicació asíncrona entre el client i el servidor, és possible modificar la pàgina web sense recarregar-la i rebre esdeveniments del servidor. En molts casos la comunicació client-servidor usa mecanismes d'invocació remota basats en l'XMLRPC o el JSON.

3.3. *External data representation*

XDR és el format d'ONC-RPC o Sun-RPC

Característiques:

- Suporta tots els tipus del llenguatge C excepte el punter a funció.
- Defineix la forma canònica intermèdia.
- No usa etiquetes (excepte per a longitud de vectors).
- Usa *stubs* compilats.
- Enter XDR: 32 bits en complement a 2, MSB primer (*big-endian*), signed int, unsigned int.
- Vector de longitud variable: nombre d'elements: 4 bytes, elements.
- Farcit fins a 4 bytes: excepte caràcters (1 byte).
- *Stubs* generats amb interfície (.idl) + RPCGEN o manualment.

Exemple de codificació d'una estructura:

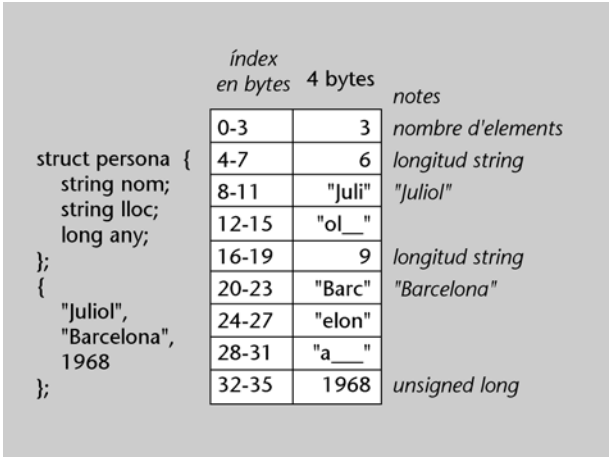


Figura 6. Representació de dades en format XDR.

3.4. *Abstract syntax notation 1*

ASN.1 és un llenguatge de descripció d'estructures, de representació comú de dades a la xarxa (codificació).

Característiques:

- Format: { etiqueta, longitud, valor }
- Usa etiquetes de tipus, 8 bits, multibyte
- Longitud en bytes del valor: si<127 ocupa 1 byte
- Valor: per exemple, enter (*integer*) complement a 2, *big-endian*

Regles de codificació:

- BER: *basic* ...; no gaire eficients, molta redundància, suport extensió
- DER: *distinguished* …; no opcions (per seguretat); longitud codificació definida
- CER: *canonical* …; no opcions (per seguretat); longitud codificació no definida
- PER: *packet* …; molt compactes, poc extensible
- LWER: *light weight*; gairebé estructura interna, codificació/descodificació ràpida

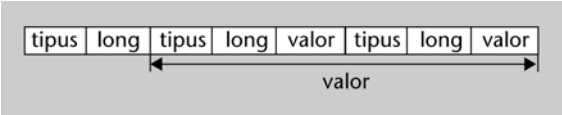
Usen ASN.1: LDAP, SMIME, SNMP, Kerberos, SSL

Permet objectes de longitud molt gran d'una manera compacta:

- Longitud: 0-127 bytes: 

- Si longitud > 127: 

- Tipus compost:



Té un rendiment menor que l'XDR, ja que és més complex, l'alineament de valors en paraules és pitjor, la (des)seriació és una mica més complexa (per exemple, per a tractar el camp longitud).

3.5. Seriació d'objectes Java

Específic per al llenguatge Java. Es poden enviar dades primitives i objectes com a arguments, o resultats, o desar-se.

Característiques:

- Classe indica que implementa *interfície seriable* (java.io).
- Si implementa *interfície remota*, es poden passar referències a l'objecte (en lloc de "passar" l'objecte).
- Si la variable es declara transitòria (*transient*), no se seria.
- Reflexió: durant l'execució és possible esbrinar les propietats d'una classe. Això implica que permet una (des)seriació genèrica: no fa falta tenir funcions de *marshalling* per a cada tipus, com a Corba.
- Referències a un objecte remot: poden aparèixer en un argument o resultat, en la invocació indica un objecte de destinació. Han de ser úniques i no reutilitzar-se.

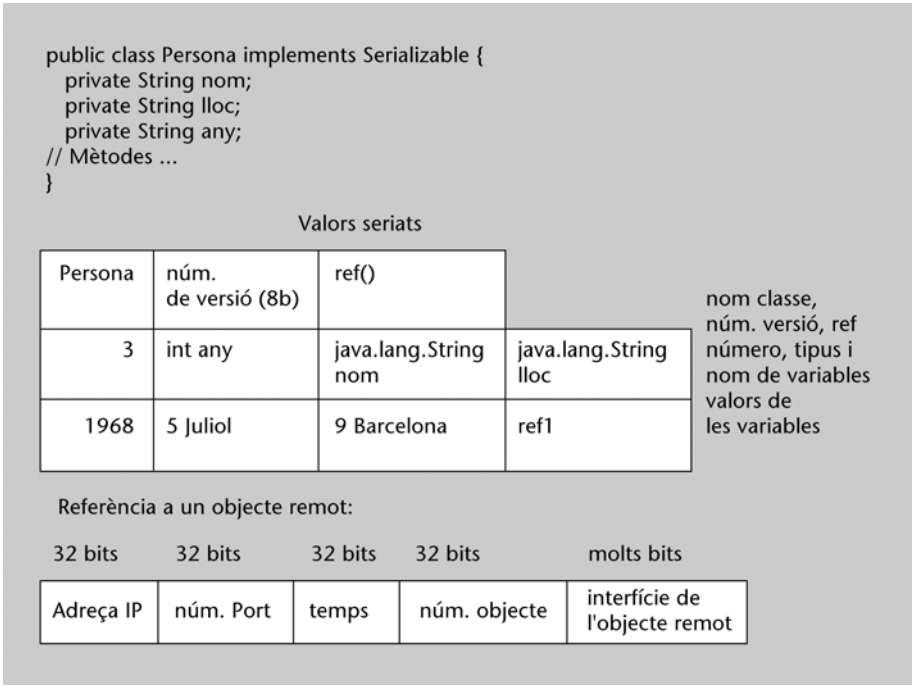


Figura 7. Representació de dades en format Java.

3.6. Característiques i comparació

L'avantatge principal de l'XDR és la seva simplicitat. Fa servir enters de longitud fixa, i totes les dades ocupen múltiples de 4 bytes; els més petits s'arrodoneixen amb bits a **0**, excepte els caràcters que es codifiquen amb un byte.

L'ASN.1-DER potser és més ineficient en espai i codificació, i el mecanisme d'empaquetament de tipus i longituds fa que requereixi moltes més instruccions per a processar les dades. Hi ha tipus de dades potser innecessaris: ***octet string***, ***IA5string*** (ASCII), ***printablestring*** (usa un subconjunt d'ASCII), i ***T61string*** (ASCII estès).

La codificació textual amb l'XML ocupa més espai, ja que ocupa bytes per a expressar el nom de cada element, no expressa la longitud de cada dada, sinó que les delimita per (") i el codi per a codificar i descodificar és significativament més complex i, per tant, més lent. La seva forma textual fa que sigui, en canvi, molt fàcil de depurar, fet que facilita la interoperabilitat.

La codificació textual JSON-RPC és més senzilla i compacta que l'XML, adaptada al seu ús amb el JavaScript i navegadors web en aplicacions interactives.

4. Invocació d'operacions remotes (RPC)

La invocació d'operacions remotes exigeix un "protocol" ordenat de diàleg entre les dues parts que participen en la invocació. Es tracta de conèixer com s'esdevé aquest procés en general per a després entrar a comentar els detalls que diferencien els mecanismes més habituals: l'ONC-RPC (proposada per Sun i que s'utilitza en diversos protocols d'Internet), l'RPC de Corba, l'RPC de Microsoft DCOM, RMI (l'RPC de Java), i el SOAP (una RPC que usa documents XML i s'utilitza per a serveis web).

En primer lloc, es caracteritzarà què és un mecanisme d'invocació remota i quines característiques principals té.

El mecanisme d'invocació d'una operació remota consisteix a

1) *recollir* (el procés client invoca una funció que en realitat és l'*stub* client. Aquest recull i seria les dades de la petició),

2) *enviar* les dades per la xarxa,

3) *seleccionar* el procés (segons com sigui l'operació es passa al procés amb l'*stub* corresponent),

4) *invocar* la petició (l'*stub* servidor desseria les dades, invoca l'operació localment), i viceversa per a *retornar els resultats*.

La invocació remota és més complexa que la invocació local per diverses raons:

- Xarxa més complexa que el bus local d'un computador.
- La xarxa pot limitar la mida dels missatges, tendència a pèrdua, desordre.
- Computadors diferents en cada procés.
- Arquitectura i format diferent de la representació de dades.

La comunicació es podria fer còmodament sobre el TCP, que proporciona un servei de transport fiable, però s'estaria perdent eficiència a causa dels mecanismes que incorpora el TCP: més temps per a obrir una connexió (mecanisme de *three-way handshake*), sota el rendiment per a transferències breus (mecanisme de *slow start*).

Si s'usa l'UDP en lloc seu, el protocol de l'RPC haurà d'introduir mecanismes específics per a superar les limitacions de la xarxa. En ser específics per a una RPC, poden donar millor rendiment que amb el TCP.

Els components d'un mecanisme d'RPC són els següents:

1) Protocol entre client i servidor per a tractar d'ocultar els efectes de les propietats no desitjables de la xarxa.
2) Suport en el llenguatge de programació i compilador per a recollir i enviar dades (compilador RPC: *stubs*).

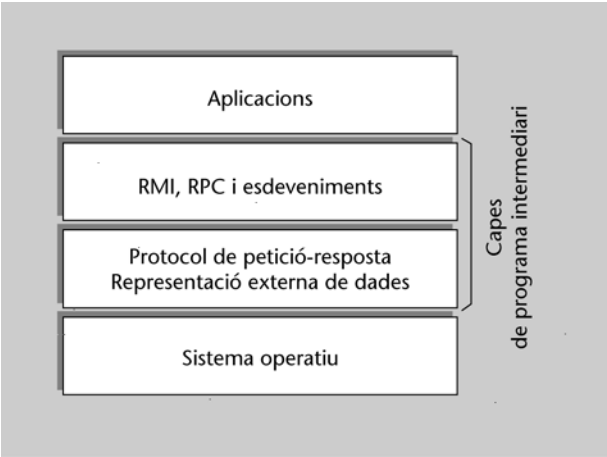


Figura 8. Situació del programari intermediari d'invocació remota en l'arquitectura d'un sistema.

En convertir una trucada local en remota, sorgeixen alguns aspectes que cal tenir en compte:

- Interfícies i generació d'estubs.
- Localització de serveis.
- Gestió de memòria i referències.
- Models de comunicació.
- Tractament d'errors.
- Garanties d'execució d'una invocació.

4.1. Interfícies i generació d'estubs

Un aspecte clau de la conversió d'una invocació local en una invocació remota resideix en el procés de generació d'estubs. Aquest procés està íntimament lligat a les interfícies dels serveis als quals s'ha d'accedir remotament.

Tant si és en un llenguatge procedimental com el C o en un d'orientat a objectes com el Java, el llenguatge de programació defineix interfícies on s'especifica la signatura de les operacions que es poden invocar i parametritzar en un programa. Així, quan un mòdul o classe vol invocar les operacions d'un altre, ha d'incloure o importar aquestes interfícies (*include*, *import*) al seu codi.

Quan es tracta d'invocació remota, si els programes que s'han de comunicar parlen el mateix llenguatge de programació (entorn homogeni), es poden utilitzar, per a generar estubs, les mateixes interfícies ofertes pel llenguatge. Així, en el Java RMI, per exemple, la interfície d'un servei remot és una interfície Java (implementant java.rmi.Remote).

En canvi, si la tecnologia d'invocació remota ha de comunicar llenguatges diferents (entorn heterogeni), és necessària la utilització d'un llenguatge de definició d'interfícies (IDL - *interface definition language*). Aquest IDL no estarà lligat a cap llenguatge en concret (neutral) i a partir d'ell es podran generar estubs en diferents llenguatges de programació. Aquest és el cas del SOAP amb l'IDL WSDL, o el CORBA amb el CORBA IDL.

Finalment, es pot destacar que el procés de generació d'estubs pot ser estàtic o dinàmic:

1) En la generació estàtica d'estubs, hi ha eines especials que a partir d'una interfície són capaces de generar els estubs. Així en el Java RMI trobem l'eina rmic, que genera estubs Java. En canvi, en el SOAP tindrem eines generadores d'estubs per a cada llenguatge com el WSDL2Java, WSDL2C#, WSDL2C++ i d'altres.

2) En la generació dinàmica d'estubs, en temps d'execució, el codi intermedi obté la interfície i genera les trucades necessàries a la Xarxa i així aconsegueix la transparència d'accés. En aquest cas no és necessari generar aquests estubs prèviament en temps de compilació.

4.2. Localització de serveis

Un altre aspecte que s'ha de tenir en compte és com trobar el servei distribuït que s'està executant en una altra màquina a la Xarxa. La manera més immediata és conèixer l'adreça IP i el port on el servei s'està executant. Tanmateix, això va en contra del principi de transparència d'accés essencial en qualsevol tipus de programari intermediari. Si el servei remot canviés de localització a la Xarxa, tots els clients n'haurien de canviar l'adreça.

Per a aconseguir aquesta transparència s'utilitzen els denominats *serveis de noms* (*naming systems*). Un servei de noms manté una llista d'associacions entre noms (identificadors simbòlics) i valors (adreces de xarxa). Per exemple, el DNS manté, en ASCII, les associacions entre noms de domini i IP associades. D'aquesta manera, ja no hem de conèixer la IP de la pàgina web que volem sol·licitar i a més s'aconsegueix transparència d'accés. És a dir, pot canviar la IP del servidor de manera transparent als clients.

En el cas de la invocació remota de procediments, els serveis de noms emmagatzemen l'identificador simbòlic del servei ("echoService") i la seva IP i port

associats. Els clients fan servir l'identificador simbòlic per a obtenir l'adreça real del servei i connectar-s'hi. És el cas de l'RMIRegistry en el Java RMI o el servei de nom del CORBA (CORBA Naming Service).

Finalment, es pot destacar que hi ha serveis de noms més sofisticats, com UDDI o LDAP, que permeten cerques sofisticades d'informació associades al servei. D'altra banda, en el context dels serveis web, és molt comú localitzar serveis directament usant l'URL.

4.3. Gestió de memòria i referències

En general, tot el que té relació amb la gestió de memòria és problemàtic: apuntadors. Els apuntadors són referències a espais de memòria local que de vegades no tenen una longitud definida, que poden ser gestionats automàticament (*garbage collection*), que acostumen a fer-se passar com a arguments d'entrada, sortida o entrada/sortida de funcions. L'accés remot a la memòria normalment no és viable i, encara que hi ha alguns sistemes d'accés a la memòria distribuïda, no és eficient.

La gestió de memòria (manual o automàtica) demana de comptabilitzar el nombre de referències internes però també des d'altres màquines a un espai de memòria (per exemple, ocupat per un objecte). Aquest és un dels problemes d'observar l'estat d'un sistema distribuït que en el mòdul 2 es va comprovar que era molt complex.

4.4. Models de comunicació

S'ha de diferenciar clarament entre el model de comunicació síncron i l'asíncron. En el model síncron, el client es bloqueja fins que rep la resposta del servidor. En l'asíncron, el client fa la invocació remota i es desconnecta, sense esperar el resultat del servidor.

Algunes vegades, és interessant que el servidor invoqui alguna operació en el client posteriorment. Per exemple, hi ha models d'invocació asíncrona que proporcionen al servidor un *callback* o invocador perquè aquest els avisi del resultat més tard. Tanmateix, generalment es requerirà que la tecnologia d'invocació remota suporta el pas per referència per a poder invocar aquest invocador posteriorment.

Una altra distinció en els models de comunicació és entre els models PULL i PUSH. En el model PULL ('llançar' en anglès), el client llança o consulta periòdicament el servidor per sol·licitar-li informació actualitzada. En el model PUSH ('empènyer' en anglès), el servidor avisa els clients de canvis que s'hagin pogut produir.

El model PULL és un model típic d'arquitectures desacoblades en què el servidor no ha de conèixer ni tenir cap referència als clients. El model PUSH implica que el servidor té una referència als clients, per la qual cosa és capaç d'avisarlos o invocar peticions sobre ells.

En moltes situacions, és possible substituir el model PUSH per un model PULL periòdic en arquitectures desacoblades on és complicat que el servidor conegui i notifiqui als clients. Per exemple, les invocacions asíncrones en la tecnologia AJAX es basen en una arquitectura PULL pel caràcter desconnectat del protocol HTTP.

4.5. Tractament d'errors

En una invocació remota es poden produir errors parcials: sense que el client s'equivoqui, pot fallar la xarxa o el servidor. El client ha de poder i saber reaccionar quan durant el camí d'anada o tornada cap al servidor, o durant l'execució, es produeixin situacions excepcionals en el servidor. Pot ser que sigui un error temporal en què venç un temporitzador i el client ha de reintentar o no l'operació, o pot ser que sigui un error de la màquina servidor.

En alguns llenguatges, el tractament d'aquestes situacions es fa amb construccions per al tractament d'excepcions, com les que ofereix Java o C++.

4.6. Garanties d'execució d'una invocació

Quan s'invoca una operació i es produeix un error, el resultat és diferent si falla durant l'anada –l'operació no s'arriba a iniciar–, durant la invocació –l'operació no es finalitza i el resultat pot ser inesperat–, durant la tornada del resultat –l'operació ha finalitzat completament a l'altre costat. Per a evitar la incertesa, el mecanisme de l'RPC pot donar garanties de lliurament, també anomenat *semàntica d'invocació*: zero-o-més execucions i idempotència.

El mecanisme de l'RPC podria reintentar la petició fins a rebre resposta o saber que s'ha produït un error remot. El resultat és que les operacions s'executarien 0 vegades en cas de fallada i entre 1 o més vegades si falla alguna i l'RPC ho ha de reintentar.

El servidor es pot protegir d'invocacions repetides degudes a retransmissions detectant i eliminant les peticions que li arriben duplicades. Per a fer-ho, hauria de guardar la història de les peticions i dels resultats però s'estalviaria execucions.

El client podria definir les operacions com a idempotents: que es poden repetir diverses vegades sense que canviï el resultat. Per exemple, afegir la lletra *a* al

final d'un fitxer no és una operació idempotent, pero sí que ho és col·locar la lletra *a* en la posició 1547.

Mesures de tolerància d'errors			Semàntica d'invocació
Retransmetre el missatge de petició	**Filtrar duplicats**	**Reexecutar proc. o retransmetre resp.**	
No	No s'aplica	No s'aplica	Potser
Sí	No	Reexecutar	Com a mínim una vegada
Sí	Sí	Retransmetre resp.	Com a màxim una vegada

Mesures de tolerància a fallades que caldria aplicar per a proporcionar certes garanties o semàntica d'invocació.

5. Tipus de protocols RPC

A continuació es descriuen les característiques dels principals mecanismes de l'RPC:

- Propi, TCP o UDP: ONC-RPC (Sun-RPC), DCE (DCOM), Corba (IIOP), Java RMI
- RPC sobre HTTP: XML-RPC, SOAP

5.1. ONC-RPC

Conegut com a ONC-RPC o Sun-RPC, basat en els tipus de dades del llenguatge C. S'ha popularitzat pel seu ús en el *network file system* o NFS.

Format d'una petició ONC-RPC

ID transacc.
Msg Type=CALL
RPCVersion=2
Programa
Versió
Procediment
Credencials
Verificador
Dades
...

El generador d'estubs (*RPC language compiler*) es denomina *rpcgen*. A partir de la interfície C del servei genera un fitxer *include* i *stubs* de client i servidor.

Característiques:

- Utilitza l'XDR com a llenguatge de codificació.
- No garanteix la semàntica "com a màxim una vegada".
- Funciona sobre l'UDP, permet la difusió (*broadcast*).
- Selecció: l'UDP selecciona el procés, l'RPC, el procediment.
- Portmapper (port 111): núm. de programa→port.
- Missatges limitats per una mida de com a màxim IP (32 kb).
- Autenticació en cada petició.
- Servidor sense estat.
- Si la retransmissió arriba al servidor mentre la resposta viatja cap al client, no nota el duplicat; sí que el nota "durant procés", i l'elimina. Aquesta memòria de curt termini no és problema en una xarxa local.

5.2. *Remote method invocation* o RMI

El mecanisme d'invocació remota de Java (RMI) serveix tant per a invocar mètodes d'altres objectes tant dins la mateixa màquina virtual Java (JVM), com d'una altra màquina virtual, a la mateixa o diferent màquina física (invocació local o remota).

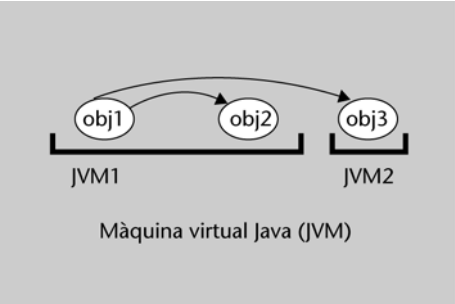


Figura 9. Invocació amb Java RMI a la mateixa o a diferents JVM.

L'RMI està pensat per a la invocació entre objectes Java, això el fa més senzill que altres mecanismes com el CORBA, que permeten la comunicació entre llenguatges diferents:

Corba/DCOM	Java RMI
Heterogeneïtat	Homogeneïtat (tot escrit en Java/JVM)
stub-skeleton+(un)marshalling	intern (trans. d'objectes, càrrega de codi dinàmica, *Java object serialization*)
IDL (neutral)	Java: interfície que estén java.rmi.Remote
Llenguatges heterogenis	Java
Tipus de dades bàsiques (comunes/representables)	Java
Correspondències complexes	=
Vida: complexa o comptar referències	=
Info. × xarxa prefixada	*Java object serialization*
Pas per una referència	Pas per referència (passar codi *stub*), pel valor (passar objecte (i codi))

En general, una invocació remota implicarà els següents passos:

En el servidor:

- Definir la interfície del servei que serà accessible remotament. En l'RMI, per a fer que un objecte sigui **remot** s'ha de declarar que implementa la interfície **remote**. Com que la invocació remota pot fallar, cada mètode de la interfície declara també l'excepció **java.rmi.RemoteException** a la secció **throws**.

- Implementar el codi del servei complint aquesta interfície i generar els estubs corresponents. En l'RMI, el compilador o generador d'estubs es denomina *rmic* i és l'encarregat de generar codi intermedi. En les últimes versions (a partir del Java 1.2), ja no és necessari generar estubs de servidor (*skeletons*) en temps de compilació. L'entorn RMI usa reflexió en temps d'execució per a fer les invocacions.

- Finalment, s'instancia l'objecte servidor i es publica o registra en un servei de noms per a fer-lo accessible als clients. En l'RMI el servei de noms es denomina *RMIRegistry*, accessible a través de l'API java.rmi.Naming.

En el client:

- A partir de la interfície del servei remot, generar els estubs de client. Això es pot realitzar en temps de compilació o execució. En compilació, rmic genera els estubs (per defecte en el protocol JRMP) necessaris per a la invocació remota. Tanmateix, no és necessari amb els *proxies* dinàmics (*dynamic proxies*), permet obviar l'ús d'*rmic*. Es produeix així una generació d'estubs dinàmicament en temps d'execució.

- El client pot ara localitzar el servidor mitjançant el servei de noms i invocar les operacions remotament. En aquesta fase, també es poden obtenir estubs de client dinàmicament baixant-los d'un servidor en localitzar el servei.

Finalment, cal destacar que el Java RMI permet el pas per referència (objectes "remots" els mètodes dels quals es poden invocar des de "lluny": altres màquines virtuals) o per valor (l'objecte "migra" completament per la Xarxa: resulta un objecte local i els seus mètodes s'invoquen localment).

Per a invocar els mètodes d'un objecte remot, en lloc d'una còpia de l'objecte la màquina on hi ha l'objecte remot envia un objecte *stub* (el seu servidor intermediari –*proxy*– o representant que actua en referència a l'objecte remot), que es construeix dinàmicament i es carrega durant l'execució, quan és necessari.

Reflexió en Java

És la possibilitat que un programa pugui esbrinar les propietats d'un objecte, com la llista de mètodes, els arguments i els tipus.

Permet de construir en temps d'execució, d'una manera dinàmica, un stub per a qualsevol objecte.

Per a fer que un objecte pugui viatjar s'ha de seriar (*Java object serialization*). Les dades i el codi de la implementació s'han de traslladar i crear una còpia al lloc remot. Això implica que qualsevol classe Java que sigui seriable es pot passar per paràmetre entre objectes remots RMI.

A continuació, un exemple d'una miniaplicació Java que invoca un mètode remot.

L'objecte Java **HelloImpl.java** conté la implementació de l'objecte remot (el servidor):

```
public class HelloImpl extends UnicastRemoteObject implements Hello {
   public HelloImpl() throws RemoteException {
     super();
   }
   public String sayHello() {
     return "Hola amics!";
   }
   public static void main(String args[]) {
     if (System.getSecurityManager() == null) { // Crear i instal·lar un gestor de seguretat{
        System.setSecurityManager(new RMISecurityManager());
     }
     try {
       HelloImpl obj = new HelloImpl();
           Naming.rebind("//myhost/HelloServer", obj); / Associar aquesta instància al nom
"HelloServer"
       System.out.println("HelloServer associat al registre");
     } catch (Exception i) {}{
        System.out.println("Error" + e.getMessage());
     }
   }
 }
```

Un client que localitza l'objecte servidor, obté en canvi un *stub* dinàmic, i invoca el mètode remot **sayHello()**:

```
try {
    obj = (Hello)Naming.lookup("/server/HelloServer");
   message = obj.sayHello();
  } catch (Exception i) { }{
    System.out.println("Excepció:" + e.getMessage());
  }
```

5.3. Invocació sobre web (HTTP)

El protocol de transferència de la web, l'HTTP, va ser concebut com un mecanisme de petició/resposta en què s'intercanvien missatges i s'invoquen mètodes predefinits (**GET**, **HEAD**, **PUT**, **POST**, **DELETE**, **PROPFIND**, **PROPPATCH**).

Amb l'aparició del servidor HTTPD d'NCSA es va definir un mecanisme simple (interfície comuna de passarel·la, *common gateway interface* o CGI) per a invocar comandaments a l'estil dels filtres Unix: el servidor web invoca un comandament amb entrada estàndard i arguments proporcionats en la petició HTTP i amb sortida estàndard de tornada a la resposta HTTP.

Es tracta de serveis que reben d'un navegador el contingut d'un formulari HTML codificat en un format simple i tornen un codi HTML perquè un navegador el presenti. Serveixen per a les persones que utilitzen un navegador puguin accedir a serveis, però no perquè un procés pugui invocar operacions.

Tanmateix, i malgrat l'orientació del protocol HTTP a l'obtenció i accés a documents remots, diversos mecanismes d'invocació remota l'han utilitzat com a protocol de comunicació entre client i servidor. Això es deu en primer lloc al fet que és un protocol obert i flexible utilitzat massivament a Internet, però també a raons de seguretat. La majoria d'organitzacions permet el trànsit HTTP extern però limita amb tallafocs les comunicacions TCP o UDP no conegudes. Per això, el CORBA o l'RMI funcionant sobre TCP o UDP poden trobar problemes de comunicació entre punts remots de la xarxa Internet.

Un dels primers mecanismes d'invocació remota a adoptar l'HTTP com a protocol de comunicació és el denominat *XML-RPC*. Com el seu nom indica, aquest protocol ha escollit l'XML com a llenguatge de codificació de dades entre client i servidor. La utilització de l'XML i l'HTTP fa que l'XML-RPC sigui un protocol idoni per a intercomunicar serveis heterogenis en diferents llenguatges i plataformes.

Invocació d'una operació (en llenguatge Python):

```
→ x.buscar({'nom':'joan','edat':42})
← {'id': 123456}
```

La invocació de l'operació anterior desencadena els següents intercanvis d'objectes en l'HTTP:

Invocació:

```
<?xml  version="1.0"?>
<methodCall>
    <methodName >buscar</methodName>
    <params>
        <param>
            <value>
                <struct>
                    <member>
                        <name >nom</name>
                        <value >joan</value>
                    </member>
                    <member>
                        <name>edat</name>
                        <value><i4> 42 </i4><value>
                    </member>
                </struct>
            </value>
        </param>
    </params>
</methodCall>
```

Per a construir un servei web, són necessaris diferents components.

– Un protocol de petició/resposta que es pugui comunicar amb un procés en un servidor: l'HTTP.

– Un format de representació de les dades que s'intercanvien en una invocació: l'XML pot ser la base per a construir-lo, encara que d'altres com el JSON també es poden utilitzar.

– Una interfície de programació per a invocar serveis de les aplicacions web.

Com veiem al lateral, l'XMLRPC defineix un conjunt de tipus limitat al fet que cada llenguatge haurà de mapar els seus tipus propis. A més, els tipus són etiquetes XML i no tipus predefinits en l'XML Schema. És una aproximació molt senzilla per a afavorir que la seva implementació sigui molt senzilla. Com a conseqüència d'això, s'han implementat llibreries XMLRPC en gran quantitat dels llenguatges de programació existents.

XML-RPC data types

<i4> or <int>
<boolean>
<string>
<double>
<dateTime.iso8601>
<struct>
<array>
<base64>

Resposta:

```
 <?xml  version='1.0'?>
<methodResponse>
    <params>
        <param>
            <value>
                <struct>
                    <member>
                        <name>id</name>
                         <value><int> 123456</int></value>
                    </member>
                    <member>
                        <name>nom_complet</name>
                        <value>joan perez</value>
                    </member>
                </struct>
            </value>
        </param>
    </params>
</methodResponse>
```

S'ha de tenir en compte que l'XMLRPC s'ha dissenyat per a ser molt lleuger i molt simple. Per això, no és comparable en funcionalitats a sistemes com el CORBA o l'RMI.

Així, l'XMLRPC no defineix interfícies remotes, no té compilador o generador d'estubs, no disposa de serveis de noms, no té excepcions i no ofereix pas per referència.

Està dissenyat per a serveis sense estat amb una interfície molt simple i en què per a localitzar el servei s'utilitza l'URL estàndard del servidor. A més, el client ha de conèixer els paràmetres i tipus del servei prèviament per a poder-lo invocar; tota la responsabilitat queda en el client. Per això no és necessària la generació d'estubs.

5.4. Serveis web

La invocació d'objectes distribuïts mitjançant tecnologies RMI, CORBA o DCOM té un problema fonamental d'interoperabilitat entre sistemes heterogenis. En conseqüència, ha estat necessari utilitzar complexos ponts i passarel·les (*bridges*) que permetessin la traducció de protocols i la seva comunicació. D'aquesta manera, sistemes desenvolupats en tecnologies Microsoft (DCOM) trobaven dificultats a comunicar-se amb sistemes basats en el CORBA. A més, l'ús de ports propietaris i codificació binària ha ocasionat també problemes a causa de tallafocs i xarxes corporatives protegides.

Per a resoldre aquests problemes, la tendència actual és apostar per protocols oberts i interoperables com l'HTTP com a canal de transport i l'XML com a llenguatge de codificació d'informació. En aquesta línia, la companyia Userland va començar definint el protocol XMLRPC com a protocol senzill d'invocació remota basat en l'HTTP i l'XML. Més tard, Userland i Microsoft van desenvolupar un protocol d'invocació remota més complet i complex denominat *SOAP*.

A partir d'aquí, els serveis web basats en el SOAP han tingut una gran acceptació com a mecanisme interoperable de comunicació a través d'Internet. L'especificació SOAP ha passat a ser un estàndard del W3C (World Wide Web Consortium) liderada per l'XML Protocol Working Group. En aquest moment ja està disponible l'especificació SOAP 1.2, tot i que la versió 1.1 s'utilitza més. Tots els grans fabricants de programari com Microsoft, IBM, ORACLE o Sun han apostat per ella com a mecanisme comú d'interoperabilitat. Si bé encara és possible trobar diferències entre diferents implementacions del protocol, les grans companyies estan treballant per aconseguir una interoperabilitat total i una vertadera maduresa del protocol.

Els avantatges més grans del SOAP són els següents:

- Interoperabilitat: l'ús d'estàndards oberts afavoreix la comunicació de sistemes heterogenis.

- Seguretat: l'ús de l'HTTP assegura la comunicació a través de tallafocs.

- Desacoblament: clients i servidors no s'han de conèixer entre ells *a priori* per comunicar-se. Un servei web està sempre disponible a Internet i pot ser invocat per diferents clients en moments diferents. D'aquesta manera el servei està sempre actiu dins del contenidor web. A més, l'HTTP no manté una connexió permanent entre client i servidor, per la qual cosa es produeix la comunicació només quan és necessària.

D'altra banda, el SOAP té com a inconvenients més grans:

- Eficiència: l'ús de l'XML és més ineficient que la codificació binària perquè requereix l'anàlisi de l'XML.

- Manteniment de sessions: com que l'HTTP no manté la connexió, es requereixen mecanismes de manteniment de la sessió quan el servei ho necessita. Tanmateix, la majoria dels serveis web trobats són sense estat (*stateless*) i no requereixen manteniment de sessions.

- Interoperabilitat: el SOAP encara està madurant i es poden trobar conflictes entre diverses implementacions. També és difícil la comunicació d'estructures de dades complexes entre implementacions diferents.

Per això, els serveis web SOAP no són la solució a tots els problemes. Per exemple, en àmbits d'intranet on l'eficiència és clau, seria recomanable utilitzar seriació binària. En canvi, si la comunicació es produeix entre sistemes heterogenis en punts remots d'Internet, llavors el SOAP és una solució adequada.

5.5. *Simple object access protocol* o SOAP

Un altre model de l'RPC amb l'XML sobre l'HTTP que ha sorgit posteriorment és el SOAP. En el SOAP, el mecanisme d'invocació continua essent senzill, però es poden intercanviar objectes amb informació més complexa.

No s'ha de confondre un mecanisme d'invocació remota de serveis web amb un sistema que es comunica amb un altre intercanviant documents XML: intercanviar documents XML de forma explícita i en un format estipulat no significa invocació, significa estipular els formularis, tràmits, els protocols particulars d'una "burocràcia". De fet, SOAP es pot utilitzar tant per a intercanviar formularis –el SOAP "estil documents"– com per a invocació remota –SOAP "estil RPC".

Història del SOAP

Versió 1.0: 2000
Versió 1.1: 2001
Versió 1.2: 2002

Les versions 1.1 i 1.2 són molt semblants. Es poden distingir per l'espai de noms que usen. SOAP 1.1 usa xmlns=http://schemas.xmlsoap.org/soap/envelope/, mentre que SOAP 1.2 usa xmlns=http://www.w3.org/2001/06/soap-envelope.

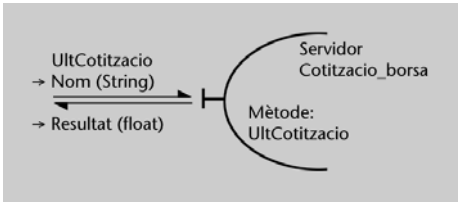
Més informació a: http://www.w3.org/2002/ws/

Per exemple, la indústria de l'automòbil pot acordar que tots els seus intercanvis es facin amb documents que continguin certa informació en format XML. En aquest cas, l'èmfasi no rau a proporcionar un mecanisme d'invocació gairebé automàtic que automàticament "seria" les dades i invoca una operació remota, sinó precisament a assegurar que el format de les dades que s'intercanvien sigui un de concret sense la preocupació per haver d'oferir un mecanisme de programació senzill i automàtic. En l'XML-RPC es pretén simplificar la tasca de programació deixant a un mecanisme genèric d'invocació la responsabilitat d'organitzar les dades en format XML.

Ni l'XML-RPC ni el SOAP pretenen proporcionar una solució completa. Per exemple, no es considera:

- Collita de memòria no usada: *garbage collection* distribuïda o DGC.
- Descripció de metainformació o gestió de versions.
- Comunicació HTTP bidireccional (poder fer peticions des del servidor cap al client).
- *Pipelining* de missatges: enviar o adjuntar diverses invocacions seguides.
- Pas d'objectes per referència (necessita DGC + HTTP bidireccional).
- Activació (reclama objectes per referència).

En la figura següent podem observar un exemple d'invocació_resposta SOAP en què el client demana la cotització (mètode **VeureUltCotitzacio**) en borsa d'una empresa (*string*) a un servidor de **cotitzacio_Borsa**, que li torna un valor (*float*):




```
POST/Cotitzacio_Borsa  HTTP/1.1
Host:www.borsa.com
Content-Type:text/xml
Content-Length:417
SOAPAction: "urn:Cotitzacio_Borsa#VeureUltCotitzacio"

<s:Envelope
  xmlns="http:// www.w3.org /2001/06/soap-envelope">
    <s:Header>
      <m:transaction  xmlns:m="soap-transaction"
                          s:mustUnderstand="true">
        <transactionID>10</transactionID>
      </m:transaction>
    </s:Header>
    <s:Body>
      <n:VeureUltCotitzacio xmlns="urn:ServeiBorsa">
        <empresa>IBM</empresa>
      </n:VeureUltCotitzacio>
    </s:Body>
</s:Envelope>
```

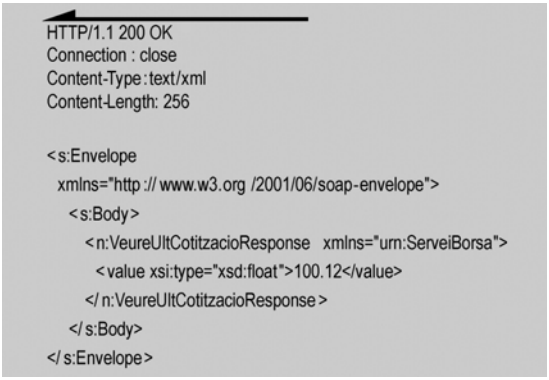


Figura 10. Interaccions en una invocació_resposta del SOAP versió 1.2.

El protocol no necessita canvis en l'HTTP i està pensat per coexistir amb tallafocs (*firewalls*) i servidors intermediaris (*proxies*). A la capçalera es distingeix una interacció en SOAP per a definir el camp: **Content-Type: text/xml**. Apareixen camps addicionals en la capçalera per a facilitar el filtratge per un tallafoc i pel mateix servidor.

Al cos apareix la crida al mètode i una resposta opcional codificades en l'XML amb el vocabulari del SOAP. La invocació i la resposta contenen un element d'arrel <Envelope> que conté la capçalera i el cos de l'objecte en SOAP (semblant a altres protocols que també distingeixen entre capçalera i cos).

Es pot veure en l'exemple anterior que la petició conté la informació següent:

- URI de l'objecte de destinació (dir servidor),
- nom de la interfície (opcional),
- nom del mètode,
- firma del mètode (opcional),
- paràmetres del mètode,
- nom, tipus (opcional), valors,
- dades capçalera (opcional).

En l'exemple, el client decideix passar una capçalera:

```
<s:Header>
    <m:transaction xmlns:m="soap-transaction" s:mustUnderstand="true">
       <transactionID>10</transactionID>
   </m:transaction>
</s:Header>
```

Aquesta capçalera opcional la pot generar el client. En aquest exemple, podria servir per a proporcionar al servidor un identificador de transacció necessari per a l'operació en el <body>. L'atribut de SOAP mustUnderstand="true" indica que el receptor ha d'entendre aquesta capçalera i, si no, ha de rebutjar tot el missatge. Si el valor fos "false", aleshores el receptor podria ignorar sense problema aquesta capçalera si no l'entén.

Després de rebre la crida a un mètode es poden donar les situacions següents:

a) No pot seguir (resultat = fallada).

b) El SOAP rep, descodifica, passa al servidor i invoca la funció corresponent a interfície + mètode (i torna el resultat).

c) El SOAP rep la petició i després falla en algun pas (torna un missatge de fallada).

Si hi ha errors, es manifesten en una resposta SOAP d'error o en una resposta HTTP d'error, per exemple:

```
HTTP/1.1 401 Unauthorized
Connection: close
```

Si no hi ha errors, es torna un missatge en SOAP (**Content-Type: text/xml**):

```
<s:Envelope
 xmlns="http://www.w3.org/2001/06/soap-envelope">
   <s:Body>
     <n:VeureUltCotitzacioResponse xmlns="urn:ServeiBorsa">
       <value xsi:type="xsd:float">100.12</value>
     </n:VeureUltCotitzacioResponse>
   </s:Body>
</s:Envelope>
```

5.5.1. Interfícies SOAP: llenguatge de descripció de serveis web o *web service description language* (WSDL)

Com en qualsevol tecnologia d'invocació remota, és necessari establir un contracte que defineixi el servei ofert. Aquest contracte definirà el nom dels mètodes i els tipus dels paràmetres i el resultat d'aquests.

El WSDL és un llenguatge XML que defineix el nom dels mètodes i els tipus de paràmetres que accepten. A més, existiran en cada llenguatge eines que generin estubs a partir del contracte (WSDL2Java, WSDL2C#, WSDL2Python...). A diferència del Java RMI, tant els clients com els servidors SOAP es poden implementar en qualsevol llenguatge de programació (amb llibreries SOAP, és clar). WSDL és també un estàndard del W3C.

Vegem un exemple de WSDL per a la següent classe Java:

```
public class Calculator {
   public int add(int i1, int i2) {
      return i1 + i2;
   }
   public int subtract(int i1, int i2) {
       return i1 - i2;
   }
}
```

Figura 11. Servei Java

```
<wsdl:definitions targetNamespace="http://localhost:8080/axis/Calculator.jws">
   <!--
   WSDL created by Apache Axis version: 1.2RC3
   Built on Feb 28, 2005 (10:15:14 EST)
   -->
   <wsdl:message name="subtractRequest">
         <wsdl:part name="i1" type="xsd:int"/>
         <wsdl:part name="i2" type="xsd:int"/>
   </wsdl:message>

   <wsdl:message name="subtractResponse">
         <wsdl:part name="subtractReturn" type="xsd:int"/>
   </wsdl:message>

   <wsdl:message name="addResponse">
         <wsdl:part name="addReturn" type="xsd:int"/>
   </wsdl:message>

   <wsdl:message name="addRequest">
         <wsdl:part name="i1" type="xsd:int"/>
         <wsdl:part name="i2" type="xsd:int"/>
   </wsdl:message>
   <wsdl:portType name="Calculator">
         <wsdl:operation name="add" parameterOrder="i1 i2">
               <wsdl:input message="impl:addRequest" name="addRequest"/>
               <wsdl:output message="impl:addResponse" name="addResponse"/>
         </wsdl:operation>

         <wsdl:operation name="subtract" parameterOrder="i1 i2">
               <wsdl:input message="impl:subtractRequest" name="subtractRequest"/>
               <wsdl:output message="impl:subtractResponse" name="subtractResponse"/>
         </wsdl:operation>
   </wsdl:portType>

   <wsdl:binding name="CalculatorSoapBinding" type="impl:Calculator">

   <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>


   <wsdl:operation name="add">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="addRequest">
               <wsdlsoap:body
               encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
               namespace="http://DefaultNamespace" use="encoded"/>
         </wsdl:input>

         <wsdl:output name="addResponse">
               <wsdlsoap:body
               encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
               namespace="http://localhost:8080/axis/Calculator.jws" use="encoded"/>
         </wsdl:output>
   </wsdl:operation>

   <wsdl:operation name="subtract">
         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="subtractRequest">
               <wsdlsoap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="http://DefaultNamespace" use="encoded"/>
         </wsdl:input>

         <wsdl:output name="subtractResponse">
               <wsdlsoap:body
               encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
               namespace="http://localhost:8080/axis/Calculator.jws" use="encoded"/>
         </wsdl:output>
   </wsdl:operation>
   </wsdl:binding>
```

```
    <wsdl:service name="CalculatorService">
          <wsdl:port binding="impl:CalculatorSoapBinding" name="Calculator">
          <wsdlsoap:address location="http://localhost:8080/axis/Calculator.jws"/>
    </wsdl:port
    </wsdl:service>
 </wsdl:definitions>
```

Figura 12. WSDL d'exemple

Podem observar que en aquest fitxer trobem una definició de les operacions ofertes pel servei (portType, operation), dels missatges de sol·licitud i resposta intercanviats (message), incloent-hi el tipus de paràmetres (type) i dels protocols de comunicació (bindings).

En general, no serà necessari que el programador conegui el format del llenguatge WSDL. Hi ha eines automatitzades que generen el WSDL a partir de codi en cada llenguatge de programació.

Tanmateix, només els tipus bàsics estan estandarditzats en el SOAP. Així, per exemple, la correspondència estàndard per a Java és la següent:

```
xsd:base64Binary byte[]
xsd:boolean boolean
xsd:byte byte
xsd:dateTime java.util.Calendar
xsd:decimal java.math.BigDecimal
xsd:double double
xsd:float float
xsd:hexBinary byte[]
xsd:int int
xsd:integer java.math.BigInteger
xsd:long long
xsd:QName javax.xml.namespace.QName
xsd:short short
xsd:string java.lang.String
```

Figura 13. Mapatge SOAP –> Java

Observem en la figura anterior que el SOAP sí que utilitza en la seva codificació tipus estàndard XML de l'XML Schema com xsd:string o xsd int entre d'altres. En realitat, es poden usar altres esquemes XML de codificació. El que es descriu d'aquesta manera segueix l'esquema referenciat amb l'URL **http://schemas.xmlsoap.org/soap/encoding/**. Alguns exemples:

```
<cost xsi:type="xsd:float">29.95</cost> <SOAP-ENC:int>30</SOAP-ENC:int>

<salutacio>Hola</salutacio>

<salutacio id="Texto-0">Hola</salutacio>
<salutacio href="#Texto-0"/>

<dada_binaria xsi:type="SOAP-ENC:base64"> aG93IG5vdjb3cNCg== </dada_binaria>

<Book> <author>H. Ford</author> <preface>Elmeullibre</preface><intro>Este...</intro> </Book>

<Book> <title>My Life and Work</title> <authorhref="#Prs-1"/> </Book>
<Person id="Prs-1"> <name>Henry Ford</name> <address href="#Dir-2"/> </Person>
<Address id="Dir-2"><email>henry@x.com</email><web>www.henry.com</web> </Address>

<Array type="int[2]"><item>3</item><item>4</item></Array>
<Array id="array-1" type="string[10,10]"><item position="[2,2]">elem 2</item>
    <item position="[7,2]">elem 7</item></Array>
```

Si volem utilitzar altres tipus complexos ja depèn de cada llenguatge i entorn. Algunes vegades se n'ha d'implementar la codificació (o usar alguna eina de l'entorn). En general els *arrays* són acceptats en la majoria d'implementacions com a estructura de dades composta. Tanmateix, col·leccions complexes poden no ser reconegudes per altres implementacions del SOAP en llenguatges diferents.

5.5.2. Localització de serveis

Per a invocar serveis remots des d'un client, primer cal localitzar-los mitjançant algun mètode. Així, és possible usar localitzacions directes al servei (incloent-hi el servidor i port) o bé mitjançant algun servei de noms remot. En el CORBA, per exemple, els localitzadors directes es denominen *IOR* (*interoperable object reference*) i el servei de nom és el *CORBA naming service*. En l'RMI podem usar l'RMIRegistry com a servei de noms per a localitzar objectes.

Per a localitzar serveis SOAP podem utilitzar l'URL del seu WSDL com a localitzador directe, o bé usar el servei denominat *UDDI* (*universal description, discovery and integration*). De tota manera, per la simplicitat de la localització directa per l'URL, els desenvolupadors acostumen a evitar l'ús d'UDDI a favor del mètode directe. En aquesta línia, els serveis web a Internet publiquen l'URL del seu WSDL en pàgines web perquè els clients els puguin invocar. També existeixen pàgines web especialitzades com www.xmethods.com amb llistes de serveis públics que inclouen l'URL de cada un d'ells.

L'ús d'UDDI s'associa més a models empresarials en què, de manera similar a les pàgines grogues, els clients puguin descobrir els serveis web empresarials que necessiten. En aquesta línia, Microsoft ofereix la seva Microsoft UDDI Business Registry com a eina per a accedir-hi. IBM i Microsoft ofereixen també directoris UDDI de registre de serveis web (http://uddi.microsoft.com/ i http://www-3.ibm.com/services/uddi/).

El nivell de descobriment (*discovery*) permet així que un consumidor de serveis pugui obtenir descripcions de serveis. Els dos mecanismes disponibles són *universal description, discovery and integration* (UDDI) i el llenguatge d'inspecció de serveis web (*WS-inspection*).

Això permet de tancar el cicle d'un servei web:

– Un ***proveïdor de servei*** que publica la seva *oferta de servei* en un ***registre de serveis***,

– Un ***registre de serveis*** que recull ofertes i pot ser examinat per un ***consumidor de serveis***.

– Un ***consumidor de serveis*** que *busca* cert servei en un ***registre de serveis*** i quan el troba connecta (bind) amb el ***proveïdor de servei*** seleccionat per invocar-li operacions amb SOAP.

Al seu torn, sobre aquests nivells es construeixen sistemes més complexos com .NET de Microsoft o ebXML de les Nacions Unides, per donar suport a serveis d'integració entre empreses.

Arquitectura orientada a serveis (SOA)

Les denominades *arquitectures orientades a serveis* (*service oriented architectures*) són una tendència actual en arquitectures de programari i interconnexió de sistemes heterogenis. Aquestes arquitectures estan formades per serveis d'aplicació dèbilment acoblats i altament interoperables. Per a comunicar-se entre ells, aquests serveis es basen en una definició formal independent de la plataforma subjacent i del llenguatge de programació (per exemple, WSDL).

Amb aquesta arquitectura, es pretén que els components de programari desenvolupats siguin molt reutilitzables i els serveis es puguin utilitzar una i una altra vegada en diferents contextos. És de destacar que els serveis solen no tenir estat persistent (*stateless*) i se solen basar en tecnologies web obertes (SOAP i WSDL), encara que no és obligatori.

Resum

En aquest mòdul s'han presentat les múltiples formes que el mecanisme d'invocació remota pot tenir en un sistema distribuït.

El mecanisme d'invocació remota és una extensió del mecanisme fonamental d'invocació local per a construir programes a base de combinar i cridar altres parts de programa.

El problema és que les dades han de "sortir fora" i, per tant, s'han de representar fora de la memòria, s'ha de canviar el format de les dades per adaptar-les a altres màquines diferents, tractar errors parcials (que falli la xarxa o la màquina remota), pèrdues o repeticions de missatges, i problemes de gestió de memòria distribuïda, entre altres problemes.

Tot s'ha de fer facilitant la interacció entre màquines diverses, respectant la seguretat de màquines, xarxes i organitzacions, minimitzant la despesa de recursos de xarxa i computació, simplificant el mecanisme perquè gairebé sembli una petició local.

En primer lloc, s'ha d'analitzar la codificació de dades per a l'intercanvi o representació externa. Hi ha diverses decisions a prendre pel que fa a la informació que s'inclou en la representació externa i aquella que els extrems coneixen però no intercanvien, els tipus de dades del format, el suport a diverses convencions de representació (*endianness*), la delimitació dels elements i la generació estàtica (amb fitxers IDL) o dinàmica (interfície d'invocació dinàmica a Corba o sempre a Java RMI) dels *stubs*: el codi que alinea i reconstrueix les dades en cada màquina que participa en una RPC.

Les codificacions poden ser textuals, simples com les que usen molts protocols d'Internet (per exemple, l'SMTP, l'HTTP), o més complexes com les basades en l'XML: poc compactes i una mica costoses de processar, però molt expressives i fàcils de depurar.

Les codificacions també poden ser binàries, simples com l'XDR o més complexes (com ASN.1), o la que utilitza Java per a seriar objectes: molt compactes i eficients, però una mica complexes i difícils de depurar.

Els protocols d'invocació remota s'encarreguen de transportar les dades codificades necessàries per a invocar una operació seguint un protocol que ofereixi certes garanties d'execució (potser, com-a-mínim-1-vegada, com-a-màxim-1-vegada), que tracti les fallades que s'esdevinguin a la xarxa, reenviï dades i vigili temporitzadors i gestioni la memòria distribuïda. És a dir, ha d'actuar com un protocol de transport per invocació.

Els protocols de l'RPC poden ser binaris, més aviat estàtics i adequats a situacions amb diversitat, com l'ONC-RPC o l'RMI, que està dissenyat específicament per al llenguatge Java.

Els protocols de l'RPC també poden ser textuals: l'HTTP + XML poden formar un protocol d'invocació acceptable: són exemples l'XML-RPC o el SOAP, que en general es poden utilitzar sobre altres protocols d'Internet com a transport per a invocar accions amb mètodes associats.

Finalment, en els mecanismes d'invocació remota és clau entendre el paper de les interfícies públiques (WSDL o IDL), la generació d'estubs a partir de les interfícies i dels mecanismes de localització de serveis remots.

Exercicis d'autoavaluació

1. Després de veure l'explicació sobre *little-endian* i *big-endian*, quina opció penseu que és la millor en un món dominat per la família de processadors 386 d'Intel?

2. Escriviu una funció, en el llenguatge de programació que us resulti més còmode, que inverteixi l'ordre *endian* d'un enter de diversos bytes (32 bits = 4 bytes).

3. Feu una taula comparant les principals diferències entre una aplicació escrita invocant operacions amb 1) el SOAP o l'XML-RPC, 2) l'RMI, 3) intercanviant documents XML. (Comenteu la facilitat/dificultat de programació, control del format de transferència, eficiència, complexitat, interoperabilitat, etc.).

4. Indiqueu les tres conseqüències més importants que tindria codificar les operacions de WebDAV amb un format de codificació binari (per exemple, amb l'XDR, l'ASN.1) en lloc de l'XML.

5. En una empresa es plantegen fer aplicacions distribuïdes i avaluen el Java XMI/RMI i el SOAP.
a) Si es tractés d'una aplicació interna en una organització, quins arguments utilitzaríeu per a triar la millor opció?
b) Si es tractés d'una aplicació per a establir una comunicació amb una altra organització (per mitjà d'una connexió d'Internet filtrada: tallafoc), quins arguments utilitzaríeu per a triar la millor opció?

6. Justifiqueu a quin model de comunicació (PULL, PUSH) pertanyen la sindicació RSS o l'IRC (*Internet relay chat*).
a) Quin model de comunicacions és més eficient pel que fa a comunicacions? Per què?
b) Per què s'utilitza el model menys eficient quant a comunicacions en un dels exemples d'aquest apartat?

Solucionari

1. Es perdria certa diversitat. És una resposta personal.

2. Una sola funció pot servir per a canviar d'un ordre a l'altre en qualsevol adreça. Una versió senzilla però no gaire eficient podria ser:

```
long reverse (long n) {
  byte b0, b1, b2, b3;

  b0 = n % 256; n = n / 256; /* n >> 8 */
  b1 = n % 256; n = n / 256;
  b2 = n % 256; n = n / 256;
  b3 = n % 256;

  return (((b0 * 256 + b1) * 256 + b2) * 256 + b3) ;
}
```

3.

Aspectes/mecanismes	RMI	Corba	SOAP o XML-RPC	Doc. XML
Facilitat/dificultat de programació	Mínima	Alta	Mitjana	Molt alta
Control del format de transferència	Mínim	Mínim	Mínim	Màxim
Eficiència	Alta	Mitjana	Baixa	Baixa
Complexitat	Mitjana	Alta	Mitjana	Baixa
Interoperabilitat	Java	Alta	Alta	Alta

4. Algunes de les conseqüències serien:

- Codificació més compacta.
- Poder enviar dades amb tipus independentment de l'arquitectura.
- Poder enviar objectes binaris d'una manera eficient.
- S'haurien de modificar els clients i servidors per a poder usar aquesta codificació.

5. Els arguments per a tots dos casos podrien ser:

- Java implica que en totes les màquines, els processos estan fets en Java i hi ha una JVM.
- SOAP: adequat per a aplicacions interorganitzatives, per això les organitzacions ofereixen i utilitzen serveis d'altres organitzacions, i ho fan únicament per mitjà d'HTTP, amb un protocol dissenyat per a facilitar el filtratge d'operacions (camps addicionals de la capçalera HTTP que informen de l'operació continguda), basat en l'XML, fet que facilita la interoperabilitat.

6. La sindicació RSS segueix un model PULL en què els clients pregunten periòdicament al servidor si té una notícia nova. L'IRC segueix un model PUSH en què el servidor avisa a través d'un sòcol (*socket*) TCP els clients de missatges enviats per d'altres a un canal de xat.

a) El model PUSH és més eficient pel que fa a comunicacions, perquè quan es produeix un canvi en el servidor, es comunica amb cada un dels clients per a avisar-los d'això. Es produeix així una única comunicació per client i canvi produït. En canvi, en el model PULL, els clients es connecten periòdicament al servidor en busca de notícies, amb la qual cosa es generen peticions i trànsit innecessari.
b) La sindicació RSS usa el model PULL per dues raons:

- Desacoblament: el servidor no ha de conèixer els clients. Com que l'HTTP és un protocol desconnectat (no es manté una connexió permanent entre client i servidor), el servidor no té manera de conèixer l'adreça del client per a avisar-lo. En IRC, en canvi, es manté obert un canal TCP per client que permet les notificacions.
- Seguretat: molts clients es troben en xarxes protegides per tallafocs o NAT, per la qual cosa no poden rebre connexions des de l'exterior. Així, és millor connectar-se periòdicament al servidor per preguntar-li si hi ha hagut canvis, ja que el servidor no pot obrir una connexió amb ells.

Glossari

common object request broker architecture *f* Arquitectura que permet que els objectes es puguin comunicar entre si amb independència del llenguatge de programació en què van ser escrits, o el sistema operatiu sobre el qual circulen. CORBA ha estat desenvolupat per un consorci industrial conegut com l'Object Management Group o OMG.
sigla: **CORBA**

CORBA *f* Vegeu ***common object request broker architecture***.

invocació d'operacions remotes *f* Protocol que permet a un programa en un computador executar un altre programa en un altre ordinador enviant dades en sèrie per la xarxa. La situació remota pot estar gairebé oculta pel programador. Encara que ha d'atendre diversos aspectes que no ocorren en una invocació local, el protocol s'encarrega de fer arribar les dades, invocar l'operació i tornar els resultats al programa que ha invocat l'operació.
en remote procedure call
sigla: **RPC**

***marshalling*/serialització/alineació** Procés de recollir dades i transformar-les en un format estàndard que es transmet per la xarxa perquè pugui viatjar a altres computadors. Quan les dades es transmeten, el computador receptor les ha de convertir de nou en un objecte equivalent.

protocol senzill d'accés a objectes *m* Protocol que permet que les aplicacions es comuniquin entre si per Internet amb independència de les particularitats locals de la xarxa o les màquines, utilitzant un vocabulari XML per a codificar les dades i operacions. És un estàndard de l'IETF i el Consorci Web.
en simple object access protocol
sigla: **SOAP**

remote procedure call *m* Vegeu **invocació d'operacions remotes**.

RPC *f* Vegeu **invocació d'operacions remotes**.

simple object access protocol *m* Vegeu **protocol senzill d'accés a objectes**.

SOAP *m* Vegeu **protocol senzill d'accés a objectes**.

Bibliografia

Coulouris, G.; Dollimore, J.; Kindberg, T. (2005). *Distributed Systems: Concepts and Design.* Londres: Addison-Wesley (trad. cast. *Sistemas Distribuidos: Conceptos y Diseño*, Pearson, 2001).

És un llibre que tracta els principis i el disseny dels sistemes distribuïts, incloent-hi els sistemes operatius distribuïts. En aquest mòdul ens interessen els capítols 4: "Comunicació entre processos"; 5: "Objectes distribuïts i invocació remota"; 20: "Estudi de Corba".

Peterson, L.; Davie, B. (2003). *Computer Networks: a system approach* (3a. ed.). EUA: Morgan-Kauffman.

És un llibre de text amb una perspectiva d'enginyeria de sistemes, amb èmfasi en l'anàlisi empírica, en el rendiment, amb referències no solament als algorismes sinó al codi real, partint dels conceptes bàsics. En aquest capítol interessen els capítols 5.3, "Protocols extrem a extrem" –tracta dels detalls de funcionament dels protocols de l'RPC–; i 7, que tracta d'alguns conceptes de representació i compressió de dades.

Bibliografia complementària

McLaughlin, B. (2001). *Solutions to Real-World Problems* (2a. ed.). Sebastopol, CA, EUA: O'Reilly.

És un llibre sobre el Java i l'XML. El capítol 12 tracta de l'XML-RPC i el SOAP per a programadors en Java.

Hi ha un gran nombre de llibres específics i detallats sobre cada tecnologia explicada: sobre les especificacions, sobre com usar-les, sobre com instal·lar-les i administrar-les, referències breus, etc. Entre d'altres, l'editorial O'Reilly té molts llibres bastant bons i molt actualitzats sobre diversos temes de l'assignatura. Són molt detallats, molt més del que es pretén en l'assignatura, però a la vida professional poden ser de gran utilitat per a aprofundir els aspectes pràctics d'un tema (http://www.ora.com).

Tanenbaum, A.; Van Steen, M. (2005). *Distributed Systems, Principles and Paradigms*. EUA: Prentice Hall.

És un llibre de text sobre principis i paradigmes de sistemes distribuïts que presenta adequadament molts dels conceptes d'aquest mòdul. Concretament, són interessants per a aquest mòdul les seccions de *communication* (RPC, RMI), *naming* i *distributed object based systems*. També és adequat per a ampliar conceptes en general de diversos sistemes distribuïts.