

Mecanismul RPC

Trecut, prezent, viitor în realizarea de aplicații distribuite

– Sabin Corneliu Buraga

Vom ilustra în acest articol modul de utilizare a mecanismului RPC (**Remote Procedure Call**), folosit în construcția de aplicații distribuite pe sisteme eterogene. Vom începe prin a prezenta cum lucrează sistemul RPC.

Ca și alte paradigme de programare în rețea, RPC a apărut în mediul UNIX și permite unui client să execute proceduri pe alte calculatoare din rețea. Paradigma RPC face modelul client/server mai puternic și constituie un instrument de programare mai simplu decât interfața socket BSD. Dacă modelul client/server se axa mai mult pe partea de comunicație între procese aflate pe mașini diferite, RPC este mai aproape de proiectarea clasică a aplicațiilor, programatorul focalizându-se pe logica programului și abia la final divizând aplicația în componente și adăugând capabilități de comunicare în rețea.

Filosofia RPC

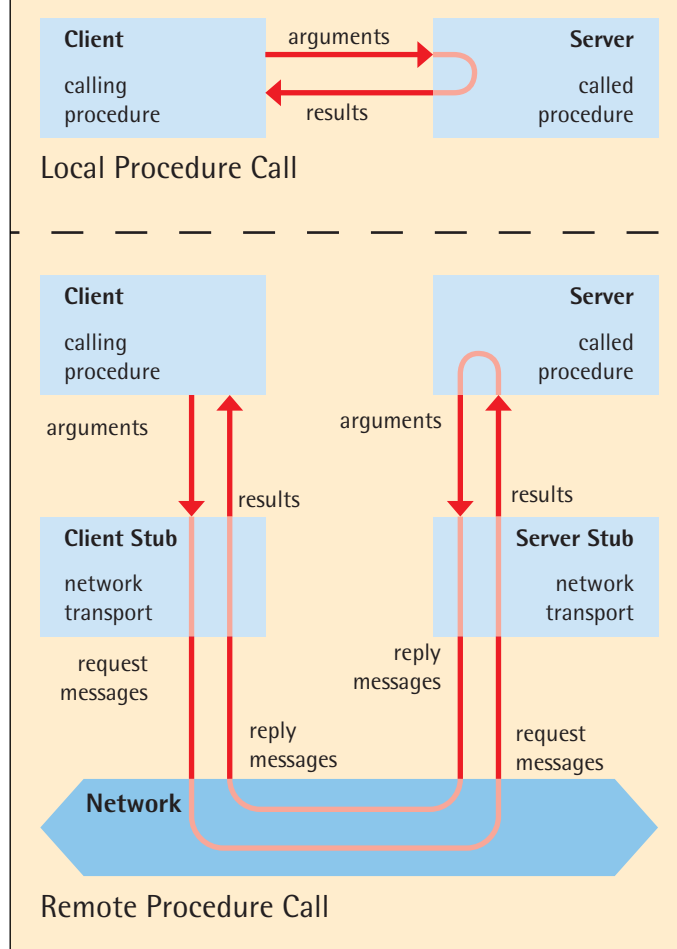
O aplicație RPC clasică va consta dintr-un client și un server, serverul fiind localizat pe mașina care execută procedura. Aplicația client comunică prin rețea cu procedura de pe calculatorul aflat la distanță transmițând argumentele și recepționând rezultatele. Clientul și serverul se execută ca două procese separate care pot rula pe calculatoare diferite din rețea. Biblioteca RPC realizează comunicarea dintre aceste două procese, utilizându-se socket-uri și stiva de protocoale TCP/IP (în mod uzual, prin protocolul de transport *UDP – User Datagram Protocol*). Procesele client și server comunică prin intermediul a două interfețe numite *stub* (*ciot*): vom avea deci un *stub* pentru client și altul pentru server.

Aceste interfețe implementează protocolul RPC. Acest protocol specifică modul cum se construiesc și cum se prelucrează mesajele emise între procesele client și server. În figura „Apelul local de procedură vs. apelul de procedură la distanță” se poate face o comparație între apelul local de procedură și apelul de procedură la distanță. Pentru diferite considerente referitoare la ce înseamnă un apel de procedură, cititorul poate consulta [4].

Stub-urile se generează de obicei cu ajutorul utilitarului *rpcgen*, după care se „leagă” de programele client și server. *Stub*-urile conțin funcții care translatează (de obicei, fără aportul programatorului) apelurile locale de procedură într-o secvență de apeluri de funcții RPC de rețea. Clientul apelează procedurile din *stub*-ul său prin care utilizează biblioteca RPC pentru a găsi procesul la distanță, pentru ca apoi să-i transmită cereri. Procesul la distanță „ascultă” rețeaua prin intermediul *stub*-ului său. *Stub*-ul serverului realizează invocarea rutinelor dorite cu ajutorul unei interfețe de apel de proceduri locale.

Clientul și serverul trebuie să comunice prin mesaje, utilizând o reprezentare a datelor independentă de calculator și de sistemul de operare. RPC utilizează un format propriu pentru reprezentarea datelor, cunoscut sub numele de *XDR* (*External Data Representation*). Componenta XDR de reprezentare a datelor este descrisă pe larg în documentul RFC 1014.

Apelul local de procedură vs. apelul de procedură la distanță



Tipurile standard suportate de XDR sunt cele uzuale din limbajul C (precum *int*, *unsigned int*, *float*, *double* sau *void*), plus altele, suplimentare (e.g. *string*, *fixed array*, *counted array*, *symbolic constant* etc.). Astfel, mecanismul XDR poate fi văzut drept o paradigmă puternică pentru transferul structurilor de date complexe între diverse aplicații Internet.

Stub-urile client și server sunt responsabile și cu traducerea în și din acest format. O bibliotecă XDR permite traducerea tipurilor predefinite din C, precum și a unor tipuri mai complexe cum ar fi vectorii de lungime variabilă.

Pentru conversia datelor din format intern în format XDR se pun la dispoziție funcțiile de mai jos, definite în antetul `rpc/xdr.h`:

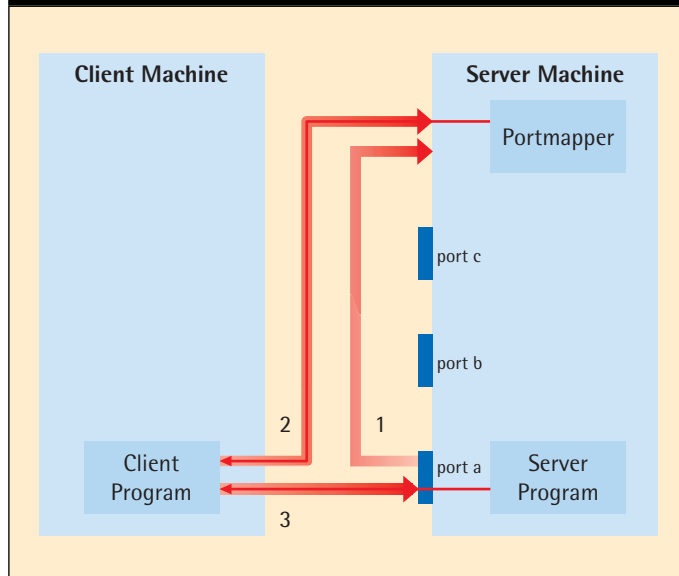
- `xdrmen_create()` – asociază unei zone de memorie obișnuite un flux de date XDR;
- `xdr_numetip()` – realizează conversia datelor, unde *numetip* se va înlocui cu unul dintre numele de tipuri definite de XDR.

Astfel, vom putea folosi funcția de conversie `xdr_int()` ca în exemplul următor:

```
#include <rpc/xdr.h>
#define BUFSIZE 400

/* lungimea zonei de memorie */
/* conversia unui intreg din format intern in format XDR */
...
XDR *xdrm;
/* zona de memorie XDR */
char buf[BUFSIZE];
int intreg;
...
xdr_mem_create(xdrm, buf, BUFSIZE, XDR_ENCODE);
...
intreg = 33;
xdr_int(xdrm, &intreg);
...
```

Pașii necesari pentru ca un client să poată apela un server



La celălalt capăt al comunicației (pe mașina aflată la distanță) vom înlocui constanta `XDR_ENCODE` cu `XDR_DECODE` pentru a realiza conversia în sens invers. Procesul de codificare a argumentelor transmise procedurilor la distanță se numește și *marshaling* sau *serializare*, iar decodificarea – *unmarshaling*.

Pentru mai multe detalii, cititorul interesat poate consulta man `xdr`.

O facilitate importantă oferită de RPC este ascunderea în totalitate a procedurilor de rețea în interiorul interfețelor *stub*. Acest lucru simplifică programele client și server, eliminând necesitatea de a controla detaliile legate de comunicarea în rețea. Ca urmare, RPC ușurează scrierea aplicațiilor distribuite. Din cauză că sistemul RPC încearcă să ascundă detalii legate de rețea, el include de obicei o specificație legată de schimbul de argumente și rezultate între client și server. Această specificație mărește portabilitatea aplicațiilor.

Cum funcționează mecanismul RPC?

Oferirea unui serviciu în rețea este diferită la sistemul RPC. Adresele clientului, serverului, numele serviciilor sunt păstrate la nivel simbolic. Un serviciu de rețea este identificat prin portul la care este oferit și unde există un *daemon* (proces care rulează în fundal) așteptând cererile de conectare. Un port în viziunea RPC reprezintă un canal logic de comunicare. *Portmapper*-ul reprezintă un serviciu de rețea care este responsabil cu asocierea de servicii la diferite porturi; acest serviciu de mapare (asociere) a porturilor este oferit la portul 111. Utilizând *portmapper*-ul, numerele de port pentru un anumit serviciu nu mai sunt fixe, ceea ce uneori prezintă un avantaj notabil. Figura „Pașii necesari pentru ca un client să poată apela un server” descrie cei trei pași necesari pentru ca un client să poată apela un server.

La pasul 1 se determină adresa la care serverul va oferi serviciul său. La inițializare, programul server stabilește și înregistrează, prin intermediul *portmapper*-ului, portul (adresa la nivelul transport) la care va oferi serviciul. În figura amintită este vorba despre portul *a*. Apoi clientul consultă *portmapper*-ul de pe mașina programului serverului pentru a identifica portul la care trebuie să trimită cererea RPC (pasul 2) – proces denumit și *binding*. Clientul și serverul pot comunica acum pentru a realiza execuția procedurii la distanță. Clientul trimite cereri, iar serverul răspunde acestor solicitări (pasul 3).

Secvența de evenimente inițiată de client printr-un apel de procedură la distanță (pasul 3 de mai sus) este descrisă de figura „Evenimentele inițiate de client printr-un apel de procedură la distanță”.

Clientul trimite o cerere în rețea cu ajutorul unui apel `callrpc()`. Programul server așteaptă mereu noi cereri, iar când o astfel de cerere este recepționată se invocă serviciul respectiv. O rutină *dispatcher* este de obicei folosită atunci când un server furnizează mai multe servicii; *dispatcher*-ul identifică cererile specifice și apelează procedura corespunzătoare. Se execută procedura și se returnează răspunsul care apoi este transmis prin rețea la client. Clientul, care în tot acest timp de după momentul emiterii cererii a așteptat inactiv, preia răspunsul și își continuă execuția.

Implementări RPC

În continuare vom descrie pe scurt implementarea Sun Microsystems a sistemului RPC, implementare numită *Open Network Computing RPC* (*ONC RPC*) – aceasta este de altfel cea mai răspândită implementare pentru UNIX/Linux. Specificația ei se găsește în RFC 1057.

În implementarea Sun, interfața RPC se structurează pe trei nivele:

- (1) *nivelul superior*: complet independent de sistemul de operare, hardware sau rețea (apelurile de proceduri la distanță sunt pentru programator simple apeluri de rutine de bibliotecă, folosindu-se biblioteca `rpcsvc`);
- (2) *nivelul intermediar*: face apel la funcțiile definite de biblioteca RPC, ca de exemplu:
 - `registerrpc()` înregistrează o procedură spre a putea fi executată la distanță,
 - `callrpc()` apelează o procedură la distanță (în prealabil înregistrată),
 - `svc_run()` rulează un serviciu RPC.
Acest nivel este utilizat de majoritatea aplicațiilor.
- (3) *nivelul inferior*: dă posibilitatea de a controla în detaliu mecanismele RPC (e.g. alegerea modului de transport al datelor prin utilizarea unuia dintre protocoalele UDP sau TCP, sincronizarea apelurilor etc.).

Procedurile la distanță se vor include într-un program la distanță.

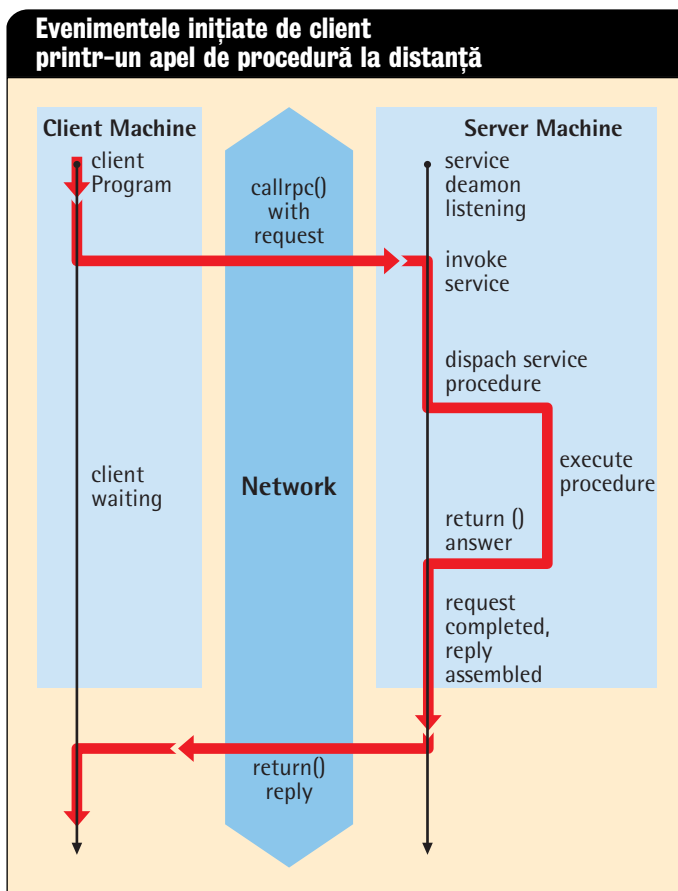
Un *program la distanță* reprezintă unitatea software care se va executa pe o mașină aflată la distanță. Fiecare program aflat la distanță corespunde unui server, putând conține un set de una sau mai multe proceduri la distanță ori date globale. Procedurile pot partaja date comune. De remarcat faptul că argumentele pasate procedurilor la distanță trebuie să fie încapsulate într-o structură (similară cu `struct`

din limbajul C) pentru a reduce numărul de argumente transmise procedurii.

Fiecărui program aflat la distanță i se va asigna un identificator unic pe 32 de biți, iar fiecare procedură componentă (care va fi executată în cadrul acelui program) este numerotată (indexată) secvențial de la 1 la n , unde n este numărul maxim de proceduri ale acelui program.

Identificatorii de program în implementarea Sun RPC au fost divizați astfel:

- 00 00 00 00 – 1F FF FF FF pentru aplicațiile RPC ale sistemului;
- 20 00 00 00 – 3F FF FF FF destinați programelor utilizatorilor;
- 40 00 00 00 – 5F FF FF FF reprezintă identificatori temporari;
- 60 00 00 00 – FF FF FF FF sunt valori rezervate.



Ca exemple de identificatori predefiniți se pot enumera:

- 10000 pentru programul (meta-serverul) *portmapper*;
- 10001 pentru programul *rstatd* care oferă informații despre sistemul aflat la distanță; se pot utiliza procedurile *rstat()* sau *perfmeter()*;
- 10002 pentru programul *rusersd* furnizând informații despre utilizatorii conectați pe mașina pe care se va executa procedura la distanță;
- 10003 pentru serverul *nfs* oferind acces la sistemul de fișiere în rețea NFS (Network File System);
- 10004 pentru serviciile *Yellow Pages*, acum regăsite sub denumirea *NIS* (Network Information Service).

Pentru fiecare program la distanță, se va include un număr întreg pozitiv desemnând versiunea. Prima versiune a unui program de obicei este 1. Următoarele versiuni vor fi identificate de alte numere, în mod unic. Numerele de versiuni oferă posibilitatea de a schimba detaliile de implementare sau de a extinde facilitățile aplicațiilor fără a asigna un alt identificator unui program.

Un program la distanță este, așadar, un 3-uplu format din (*identificator de program, versiune, index procedură*).

Unele implementări RPC permit folosirea unui utilitar (e.g. *uuidgen*) pentru generarea numărului universal de identificare a programului.

O componentă a implementării Sun pentru RPC este compilatorul (utilitarul) *rpcgen*. Acest compilator produce *stub*-urile client și server, produce o rutină *dispatch* capabilă să lucreze cu proceduri multiple și oferă flexibilitate în realizarea programelor server și client. Compilatorul *rpcgen* solicită la intrare un fișier de specificații RPC. Figura „Obținerea codului pentru serverul și clientul RPC” prezintă cum se obține codul pentru server și client RPC. Fișierul de specificații RPC este numit *q.x*. Utilizând compilatorul *rpcgen*, nu mai este necesar să realizați comunicarea RPC în codul serverului și clientului. Funcțiile respective sunt realizate de *stub*-ul server (*q_svc.c*) și *stub*-ul client (*q_clnt.c*) generate de compilator pornind de la fișier de specificații RPC. Aceste *stub*-uri utilizează apeluri RPC de nivel scăzut; acest lucru înseamnă că se complică puțin scrierea aplicației client și nu se mai folosește apelul *callrpc()*.

Compilatorul *rpcgen* generează și filtrele (funcțiile) de codificare și decodificare XDR utilizate de clientul și de serverul RPC. Aceste rutine se găsesc în fișierul *q_xdr.c*. Se mai generează și un fișier antet *q.h* care se include în toate celelalte trei fișiere generate (*q_svc.c*, *q_clnt.c*, *q_xdr.c*), dar și în programele scrise de programator pentru client și server (numite „aplicație client” și „aplicație server” în figură). Compilatorul *rpcgen* nu poate realiza totul, trebuind scrise programele C pentru server și client pe care le putem denumi *server.c* și *client.c*.

Pentru a obține serverul va fi necesară compilarea funcțiilor serverului, a *stub*-ului server și a rutinelor XDR prin comanda:

```
gcc server.c q_svc.c q_xdr.c -o server
```

Serverul RPC poate fi inițializat direct la încărcarea sistemului de operare prin intermediul *daemon*-ului *inetd*.

Pentru a genera codul executabil al clientului va fi necesară compilarea funcțiilor clientului din *client.c*, a *stub*-ului client și a rutinelor XDR prin:

```
gcc client.c q_clnt.c q_xdr.c -o client
```

Compilarea presupune existența bibliotecii *rpclib*. Pentru Linux, aceasta este inclusă, uzual, în bibliotecile standard.

Realizarea aplicațiilor client/server cu ajutorul mecanismului RPC implementat de Sun presupune așadar scrierea a trei elemente:

1. o specificație RPC într-un fișier *.x*;
2. un program *server.c* care să conțină procedurile invocate;
3. un program *client.c* care trebuie să realizeze apelurile corespunzătoare.

Ca fișier-specificație RPC putem da următorul exemplu (*trenuri.x*), folosit la dezvoltarea unei aplicații prin care se solicită informații despre mersul trenurilor, aplicație concepută de studentul *Manuel Șubredu* de la Facultatea de Informatică a Universității „A.I. Cuza” din Iași:

```
struct request
/* cererea adresata serverului */
{
    char tren_d[100];
/* descrierea trenului */
    char nr_tren[10];
/* numarul trenului */
    int optiuni;
/* PLECARI sau SOSIRI */
};

struct answer
```

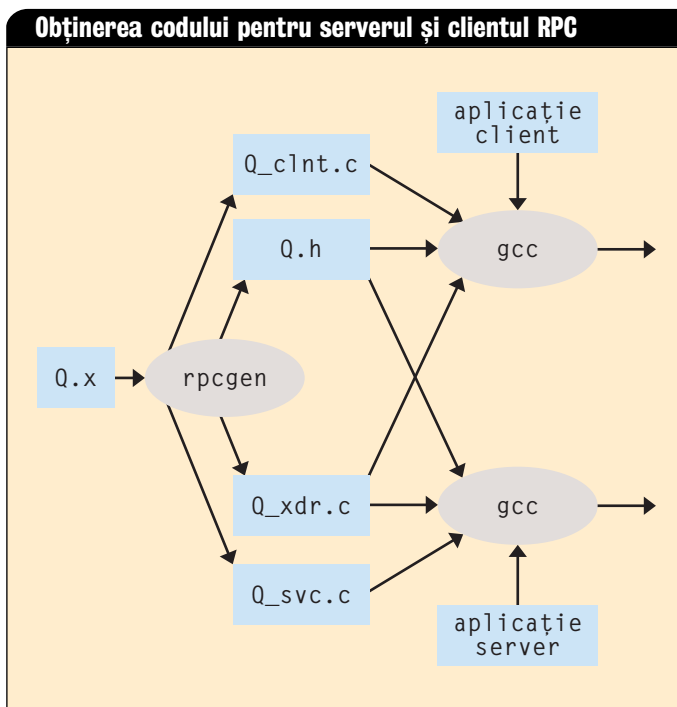
```

/* raspuns primit de client */
{
    char raspuns[4000];
};

program TRENURI
{
    version VERSIUNE
    {
/* procedura apelata la distanta */
        answer TREN (request) = 1;
    } = 1;
/* versiunea 1 (prima) */
    } = 0x200000f1;
/* identificatorul unic al programului */

```

Pentru alte implementări RPC, în locul specificației .x se folosește formatul IDL (*Interface Definition Language*), familiar programatorilor de aplicații CORBA. Mai multe detalii referitoare la conceperea de aplicații RPC se pot regăsi în lucrările [1] și [2].



Pentru a vedea ce programe RPC au fost înregistrate se poate utiliza comanda `rpcinfo`.

De asemenea, se pot realiza aplicații RPC asincrone, fără a apela la utilitarul `rpcgen`, programatorul trebuind să-și scrie propriul `dispatcher`.

Utilizări

Deși apărut relativ de mult timp, mecanismul RPC își găsește utilizări interesante, dintre care cea mai importantă o reprezintă accesul la sisteme de fișiere la distanță prin *NFS* (*Network File System*). NFS utilizează filosofia sistemului clasic de fișiere din UNIX, ierarhia de directoare NFS folosind terminologia UNIX. Astfel, NFS adoptă multe dintre detaliile sistemului de fișiere din UNIX (tipurile, modurile de acces, atributele fișierelor etc.). Operațiile care pot fi realizate asupra unui fișier aflat la distanță includ operații uzuale de citire/scriere, plus cele de consultare a intrărilor unui director, de creare, redenumire și ștergere a unui fișier, de obținere a unor informații despre un fișier etc.

În fapt, putem vedea NFS ca exemplu tipic de sistem distribuit de fișiere (*distributed file system*).

Pentru a fi operațional, NFS presupune existența unui server și a unui client NFS care vor comunica via RPC. Un sistem de fișiere la distanță va putea fi disponibil prin intermediul protocolului `mount`, familiar oricărui administrator de sistem. Folosind `mount` se realizează și autentificarea și autorizarea cererilor clienților NFS. Mecanismul de autentificare se bazează pe cel al sistemului UNIX. Pentru mai multe amănunte despre NFS se poate consulta RFC 1094.

O alternativă la mecanismul RPC de la Sun este *DCE* (*Distributed Computing Environment*) *RPC*, o parte dintre serviciile de rețea vitale ale Windows NT/2000/XP, mai ales privind replicarea datelor, bazându-se pe aceasta. La fel, Microsoft Message Queue Server utilizează intern DCE RPC.

În anii '90, mecanismul RPC a continuat să fie utilizat prin abordarea obiectuală *ORPC* (*Object RPC*), mesaje de cerere și răspuns de la distanță fiind încapsulate de obiecte. Ca descendenți direcți ai *ORPC* se pot enumera *DCOM* (*Distributed Common Object Model*, acum *COM+*) and *CORBA Internet Inter-ORB Protocol* (*IIOP*). Aceste modele folosesc pentru interschimbul de date protocoalele Common Data Representation (*CDR*) și, respectiv, Network Data Representation (*NDR*), inspirate de *XDR*.

În loc de concluzii (sau înapoi în viitor)

Deși aparent n-ar avea nici o legătură cu Web-ul, RPC a reintrat în atenția comunității programatorilor odată cu apariția conceptului de *serviciu Web* și ale sale protocoale de comunicație bazate pe XML, dintre care cel mai mediatizat este *SOAP* (*Simple Object Access Protocol*), văzut ca *sumum* dintre RPC, HTTP și XML. În loc de *XDR* (sau *NDR*) se folosește XML, iar mecanismul RPC se realizează prin intermediul protocolului HTTP.

Nimic nou sub Soare?...

Sabin-Corneliu Buraga este doctorand în Computer Science la Facultatea de Informatică, Universitatea „Al.I. Cuza” din Iași și poate fi contactat la busaco@infoiasi.ro sau <http://www.infoiasi.ro/~busaco/>. ■ 98

Referințe

- [1] J. Bloomer – „Power Programming with RPC”, O'Reilly, 1992
- [2] S. Buraga, G. Ciobanu – „Atelier de programare în rețele de calculatoare”, Editura Polirom, Iași, 2001
- [3] D. Comer, D. Stevens – „Networking with TCP/IP. Vol. III: Client-Server Programming and Applications”, Prentice Hall, New Jersey, 1993
- [4] B. Meek – “What is a Procedure Call?”, ACM Sigplan Notices, Vol. 30, No. 9, September 1995: <http://www.kcl.ac.uk/kis/support/cc/staff/brian/procsn.html>
- [5] A. Tanenbaum – „Modern Operating Systems”, Prentice Hall PTR, New Jersey, 2001
- [6] *** – „The NFS Distributed File Service”. Sun's NFS White Paper, 1995: <http://www.sun.com/software/white-papers/wp-nfs>
- [7] *** – „Using Remote Procedure Calls”, CISCO Technical Report, 1998
- [8] *** – „Web Services”, W3C: <http://www.w3.org/ws>