

Platformă standard pentru dezvoltarea aplicațiilor radio

Nevoia unei platforme standard specializată care să facă posibilă dezvoltarea sistematică a aplicațiilor radio a apărut în fața producătorilor de echipamente radio definite prin program sau virtuale odată cu creșterea integrării dispozitivelor radio și dispozitivelor de calcul. Multe noi protocoale de comunicații radio (de exemplu comunicațiile în rețele “trunking”) și tipuri de modulație (cum ar fi DRM și DAB) necesită o disponibilitate mărită de putere de calcul. În același timp, tot mai multă putere de calcul este utilizată pentru a adăuga dispozitivelor radio noi funcții și pentru înlocuirea panourilor cu elemente fizice de control prin panouri virtuale ce sînt ușor de personalizat în funcție de categoriile de utilizatori și permit accesul la funcțiile dispozitivului radio prin intermediul interfețelor standard ale calculatoarelor (tastatură, “mouse”).

Pentru a aprecia corect avantajul utilizării unei asemenea platforme standard pentru controlul dispozitivelor radio (receptoare și emițătoare) prin intermediul calculatoarelor personale, se prezintă standardul *XRS* (Extensible Radio Specification) ([1]) inițiat/propus și dezvoltat de firma WiNRADiO Communications.

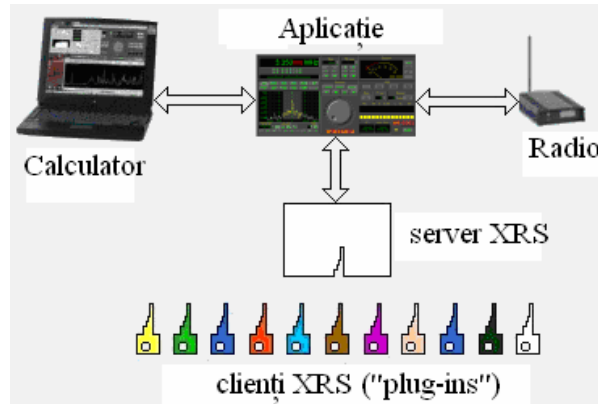


Figura 1

XRS definește interfața între aplicația informatică (programul) de bază care controlează dispozitivul radio (un “Server”) și un modul funcțional (“plug-in module”) opțional, suplimentar, atașabil ei (un “Client”). Specificația interfeței este suficient de flexibilă pentru a permite unei game largi de dispozitive radio să fie controlate printr-o gamă largă de asemenea module funcționale. Cu alte cuvinte, programul dezvoltat (scris) pentru un model particular de dispozitiv radio și modulele funcționale suplimentare vor putea lucra la fel de bine cu un alt dispozitiv, dacă sînt conforme specificației *XRS*.

Introducerea unei platforme standard, cum este *XRS*, este benefică pentru toate părțile:

- Pentru *utilizatorul final*, deoarece programele dezvoltate pe platforma *XRS*, achiziționate pentru un anumit model de dispozitiv radio care este conform cu aceste specificații, vor lucra la fel de bine cu oricare alt model. Cu cît numărul aplicațiilor dezvoltate pe platforma *XRS* va fi mai mare, cu atît mai mult va crește eficiența investiției în dispozitivele radio conforme *XRS*.

- Pentru *dezvoltatorii de aplicații*, deoarece odată scrisă, o aplicație *XRS* va lucra cu toate dispozitivele radio conforme *XRS*, nu numai cu modelul unui anumit producător. În felul acesta se economisește timp și crește piața produsului informatic,
- Pentru *producători*, întrucât aceștia pot valorifica aplicațiile *XRS* deja existente. Prin produse compatibile *XRS*, producătorii devin automat mai atractivi pe piață.

Principalele obiective ale platformei *XRS* au fost:

- Să permită noilor funcții radio să fie dezvoltate separat, într-o formă modulară, și să fie adăugate ușor pentru a mări funcționalitatea aplicațiilor de control al dispozitivelor radio.
- Se elimine incompatibilitățile dintre diferitele modele de dispozitive radio. Odată scrisă, o aplicație *XRS* va lucra cu oricare dispozitiv radio compatibil *XRS*.
- Să ofere o platformă deschisă dezvoltatorilor independenți de aplicații informatice, pentru o gamă largă de aplicații radio.
- Să ofere producătorilor de dispozitive radio posibilitatea folosirii avantajului oferit de aplicațiile *XRS* prezente și viitoare. Prin licențierea tehnologiei de server *XRS* în produsele lor, producătorii pot beneficia de efortul combinat al tuturor dezvoltatorilor independenți.
- Să asigure evoluția/extinderea standardului, pentru a fi capabil de acomodare cu evoluțiile din domeniul tehnologiei dispozitivelor radio și cele ale domeniului tehnicii de calcul.

Aplicațiile *XRS* pot fi create în mediile integrate de dezvoltare C/C++ și Delphi, rezultând implementări *XRS* specifice pentru fiecare sistem de operare (Windows, Macintosh sau Linux) și chiar generație de procesoare.

Atunci când se lansează în execuție un program (aplicație informatică ‘server *XRS*’) scris pentru un dispozitiv radio care se conformează specificației *XRS*, acesta caută fișiere cu extensia ‘.XRS’ în subdirectorul *plugins* din directorul aplicației și încearcă să le încarce și să le inițializeze.

“Viața” unui modul funcțional de la încărcare până la oprirea execuției aplicației principale parcurge următoarele etape:

- La pornirea aplicației principale (‘server *XRS*’), modulul funcțional este încărcat și este apelată funcția `xrsPluginInit` pentru a informa modulul că aplicația principală se conformează unei anumite versiuni a platformei *XRS*. Modulul funcțional răspunde prin numele său și informează aplicația principală dacă poate fi pornit sau nu imediat, precum și despre ce anume poate face.
- Atunci când modulul funcțional urmează să fie pornit (în mod automat la încărcare, la inițiativa utilizatorului sau de către un alt modul funcțional) este apelată funcția `xrsPluginStart`. Ca urmare a apelării acestei funcții se creează o nouă instanță a modulului funcțional iar aceasta întoarce un identificator propriu unic. Dacă aplicația principală controlează simultan mai multe dispozitive radio, fiecare dispozitiv poate cere deschiderea a câte unei instanțe a modulului funcțional pentru a fi deservit de aceasta. În această situație, fiecare instanță de dispozitiv radio transmite câte un indicator unic (propriu instanței dispozitivului) către instanța modulului funcțional aferentă, la pornirea acestuia.

- După ce modulul funcțional este pornit, el va primi notificări (sau evenimente) de la aplicația principală prin funcția `xrsPluginNotify`. Modulul poate de asemenea să controleze multe aspecte ale aplicației principale.
- Când modulul funcțional primește o notificare `PN_CLOSE`, el trebuie să se oprească și să distrugă instanța, nu înainte de a anunța aplicația principală că și-a încetat existența ca răspuns la comanda `PN_CLOSE`.
- Atunci când aplicația principală este oprită, ea apelează funcția `xrsPluginDone` pentru a informa modulul funcțional că urmează să fie șters din memorie.

Un modul funcțional este o librărie de cod nativ cu extensia de fișier '.XRS'. Intern, tipul fișierului depinde de sistemul de operare, astfel:

- *Windows*: librărie cu legătură dinamică ('.DLL')
- *Linux*: obiect partajat ('.SO' sau '.DSO')
- *Mac*: librărie partajată sau resursă de 68k

Limbajul de programare utilizat nu are importanță atât timp cât poate genera cel puțin unul dintre tipurile de fișiere de mai sus.

Deși este o librărie (*API* – Application Programming Interface) de cod nativ și de prin urmare este specifică unei platforme de sistem de operare și rulează dintr-o aplicație conformă *XRS*, *XRS API* este proiectată să furnizeze un grad maxim de flexibilitate și consistență funcțională peste toate platformele de operare.

De îndată ce se obține o licență de dezvoltator pentru '*XRS Client*', creerea unui modul funcțional *XRS* este un proces simplu care urmează următorii pași:

- Modulul funcțional se poate dezvolta pornind de la exemple furnizate în pagina de Web a fabricantului de dispozitive radio.
- Fiecare modul funcțional trebuie să implementeze patru funcții *API* esențiale, așa cum se va exemplifica mai departe. El trebuie să se poată închide atunci când programul principal (server-ul) îi cere asta.
- Odată scris codul, modulul funcțional trebuie compilat și instalat.
- Modulul funcțional se testează și eventual se corectează. Multe probleme pot fi preîntâmpinate prin testări dese, mai întâi ale funcționalității de bază, urmînd ca acele funcții care conferă unicitate modulului să fie introduse treptat.

Pentru a identifica un modul funcțional ("plug-in"), aplicațiile conforme *XRS* ('server'-e *XRS*) localizează mai întâi fișierul librăriei modulului avînd extensia '.XRS' și îl încarcă în memorie.

După încărcarea modulului funcțional, aplicația principală apelează funcția `xrsPluginInit` prin care informează modulul funcțional despre versiunea de platformă *XRS* suportată. Dacă modulul funcțional suportă versiunea *XRS* și în plus verifică validitatea server-ului *XRS* (aplicației principale), el răspunde server-ului prin numele modulului (cu o lungime de maximum 64 caractere) și categoria căreia îi aparține. Modulele funcționale trebuie să poarte nume diferite pentru a se elimina orice posibilitate de confuzie la încărcarea lor de către aplicația principală.

Valoarea de răspuns a modului funcțional specifică de asemenea gradul de control pe care acesta îl poate exercita asupra server-ului și modul în care server-ul trebuie să utilizeze modulul. De exemplu, modulul funcțional poate cere server-ului să fie pornit imediat (“plug-in” cu autostart, ce poate chiar “ascunde” serve-ul și prelua în acest fel interfața cu dispozitivul radio), să nu apară în meniul aplicației principale (atunci când modulul poate fi pornit numai de un alt modul funcțional), etc.

De îndată ce un modul funcțional este încărcat în memorie, aplicația principală apelează funcția `xrsPluginInit`, înainte de a fi creată prima lui instanță. Această funcție este folosită la alocarea de memorie și resurse partajate între toate instanțele unui modul funcțional.

C/C++:

```
int xrsPluginInit(int iXRSVer, PCHAR lpServerId, PCHAR lpName,  
int cbName);
```

Delphi:

```
Function xrsPluginInit(iXRSVer: Integer; lpServerId, lpName: Pchar;  
CbName: Integer): Integer;
```

La închiderea aplicației principale aceasta apelează funcția `xrsPluginDone`, care eliberează memoria sau resursele alocate prin `xrsPluginInit`.

Modulul funcțional trebuie să verifice informația despre versiunea platformei *XRS* conținută de parametrul `iXRSVer`, pentru a verifica că este compatibil cu capacitățile *API* furnizate de aplicația principală. Octetul inferior indică numărul versiunii minore iar octetul cel mai semnificativ indică numărul versiunii majore. De exemplu, pentru versiunea 1.0 `iXRSVer` va fi egal cu 0x0100.

Apoi, modulul funcțional trebuie să valideze identitatea (ID-ul) serverului (aplicației principale) prin apelarea funcției `xrsValidateServer`, iar dacă acesta este valid el răspunde cu valoarea TRUE. În cazul unui răspuns cu valoarea FALSE, modulul funcțional se închide singur întrucât inițializarea nu se poate realiza (eroare de alocare de resurse, versiune *XRS* nesuportată ori server *XRS* invalid).

Înainte de terminarea cu succes a procedurii de inițializare nu se poate întreprinde nici o acțiune pe baza funcțiilor *XRS API*.

Dacă inițializarea are succes, modulul trebuie să întoarcă o valoare pozitivă a cărui cel mai puțin semnificativ octet informează server-ul despre gradul de control pe care modulul îl permite.

Oricând după momentul inițializării, se poate crea sau porni o instanță a modului funcțional. Dacă aplicația principală poate opera simultan mai multe dispozitive radio de același tip, pot exista mai multe instanțe ale aceluiași modul funcțional.

O instanță a modului funcțional poate fi creată din meniu la inițiativa utilizatorului, poate fi pornită automat de aplicația principală sau de către un alt modul funcțional.

C/C++:

```
DWORD xrsPluginStart(HWND hAppWnd, LPRADIODEVCAPS  
lpRadioDevCaps, PLUGINPROC lpPluginProc,  
PDWORD lpFilterFlags);
```

Delphi:

```
function xrsPluginStart(hAppWnd: HWND; lpRadioDevCaps: PRadioDevCaps;  
lpPluginProc: TpluginProc; var lpFilterFlags: Longint):  
Integer;
```

Parametrul **hAppWnd** specifică indicativul ferestrei dispozitivului radio creată de aplicația principală. Aceasta poate fi chiar fereastra principală a aplicației sau o fereastră “child” (copil), dacă aplicația admite controlul simultan a mai multor dispozitive radio. În general, parametrul are rolul de a identifica un anumit dispozitiv radio într-o situație cu instanțe multiple.

Parametrul **lpRadioDevCaps** indică o structură **RADIODEVSCAP** care conține capabilitățile dispozitivului radio (care poate fi un receptor, un emițător sau un emițător-receptor). Modulul funcțional trebuie să verifice diversele câmpuri ale structurii pentru a se asigura că dispozitivul radio poate fi controlat complet și dacă este adecvat pentru respectivul dispozitiv.

Parametrul **lpPluginProc** este un pointer la o funcție de răspuns (“callback”) din cadrul aplicației principale, pe care modulul funcțional o poate utiliza pentru a controla funcționarea dispozitivului radio, sau pentru a informa aplicația principală de închiderea modului.

C/C++:

```
Typedef DWORD (CALLBACK* PLUGINPROC) (DWORD, int, DWORD,  
int, LPVOID);
```

Delphi:

```
type  
TpluginProc = function (hPlugin: Longint; uMsg: Integer; dwParam:  
Longint; cbData: Integer; lpData: Pointer):  
Longint;
```

După trimiterea unei comenzi **PM_CLOSED** prin funcția “callback”, modulul funcțional nu va mai putea primi nici o notificare și nu va mai putea genera nici o comandă.

Parametrul **lpFilterFlags** indică o variabilă de 32 biți care poate fi utilizată de modulul funcțional pentru a filtra oricare notificare pe care nu dorește să o primească. Toate notificările nedorite de modulul funcțional pot fi ignorate, cu excepția notificării **PN_CLOSE**.

Dacă instanța modulului funcțional pornește cu succes, ea trebuie să întoarcă o valoare pozitivă care este unică pentru această instanță. Aplicația principală nu va folosi această valoare decât ca un identificator în funcția de notificare.

Dacă instanța modulului funcțional nu pornește, funcția trebuie să întoarcă zero.

După instalarea unui modul funcțional, el poate fi pornit în orice moment. Când este pornit, i-se trece o structură cu datele dispozitivului radio (fie receptor, ori emițător). Aceste informații includ numele fabricantului și pe cel al produsului, domeniul de frecvențe în care lucrează, și multe altele. Toate informațiile sînt conținute în structura **RADIODEVCAPS**, a cărei formă pentru C/C++ este redată mai jos:

```
typedef struct _RADIODEVCAPS {
    int cbTotalSize;
    int cbFixedSize;
    int cbFreqRangeSize;
    /* --- Informații despre produs --- */
    CHAR szManufacturer[32];
    CHAR szProduct[32];
    CHAR szSerialNum[16];
    CHAR szUserDefName[64];
    DWORD dwAppVersion;
    int iDeviceNum;
    /* --- Informații globale --- */
    DWORD dwFreqRes;
    DWORD dwCalibrated;
    LPTONECAPS lpToneCaps;
    int iMinBpFreq;
    int iMaxBpFreq;
    LPPARAEQCAPS lpParaEqCaps;
    LPGRAPHEQCAPS lpGraphEqCaps;
    /* --- Informații despre receptor --- */
    DWORD dwRxFeatures;
    LPVOID lpRxExtraInfo;
    int iSquelchFeatures;
    int iMinSquelchLevel;
    int iMaxSquelchLevel;
    int iMinSquelchNoise;
    int iMaxSquelchNoise;
    int iNumRxFreqRanges;
    LPFREQRANGE lpRxFreqRanges;
    int iNumRxModes;
    int cbDemodDefSize;
    LPDEMODDEF lpRxModeDefs;
    int iNumRfInputs;
    int iMaxAtten;
    int iAttenStep;
    int iMaxPreamp;
    int iPreampStep;
    int iAgcSpeeds;
    LPAGCEXCAPS lpAgcExCaps;
    int iMinIfGain;
    int iMaxIfGain;
    int iMaxVolume;
    int iVolumeStep;
    int iBalanceRange;
    int iBalanceStep;
    int iRxAudioSources;
    int iMaxNbThreshold;
    int iMaxNotchFreq;
```

```

int iMaxNoiseReduction;
LPDSPCAPS lpRxDspCaps;
/* --- Informații despre emițător --- */
DWORD dwTxFeatures;
LPVOID lpTxExtraInfo;
int iNumTxFreqRanges;
LPFREQRANGE iTxFreqRanges;
int iNumTxModes;
int cbModDefSize;
LPMODDEF lpTxModeDefs;
int iTxModSources;
int iMaxTxPower;
int iMaxAntiVox;
int iAudioProcFlags;
int cbAudioProcSize;
LPTXAUDIOPROC lpAudioProcCaps;
int iTxSelCallTypes;
int iMaxToneLevel;
int iMaxToneDuration;
int iTxInitiators;
int iTxMaxReleaseDelay;
LPDSPCAPS lpTxDspCaps;
/* --- Memoria suport --- */
DWORD dwMemFeatures;
DWORD dwMaxRecords;
int iNumBanks;
} RADIODEVCAPS, FAR *LPRADIODEVCAPS;

```

Ori de câte ori o instanță a modulului funcțional este creată cu funcția **xrsPluginStart**, instanța poate fi închisă printr-o notificare cu **PN_CLOSE** primită într-o comandă **xrsPluginNotify**. Instanțele pot fi închise de ele înșiși, de utilizator, de un alt modul funcțional sau prin închiderea aplicației principale.

Când o instanță este închisă, trebuie să trimită aplicației comanda **PN_CLOSE**, după care instanța nu va mai putea primi noi notificări și nici nu va mai putea genera alte comenzi. Dacă modulul funcțional înscrie valoarea 1 în bit-ul numărul 16 din parametrul **dwParam** al comenzii **PN_CLOSE**, instanța aplicației dispozitivului radio se închide. Dacă aplicația principală suportă un singur dispozitiv radio, întreaga aplicație trebuie să se închidă.

Când o aplicație principală urmează să se închidă, ea va apela funcția **xrsPluginDone**. Această funcție dă modulului funcțional posibilitatea să șteargă toate datele și resursele alocate prin funcția **xrsPluginInit** care sînt partajate de toate instanțele. Totodată, modulul funcțional poate opri oricare flux Intrare/Ieșire semnificativ, firele de acțiune (dacă există), și poate elibera memoria blocată.

C/C++:

```
void xrsPluginDone(void);
```

Delphi:

```
procedure xrsPluginDone;
```


Următorul exemplu ilustrează cerințele minime pentru codul sursă în C al unui modul funcțional:

```

/* Variabile globale pentru stocarea adreselor callback */
PLUGINPROC PluginProc = NULL;
static char PluginName[] = "Exemplu Plug-in Minimal";
/* Funcția de inițializare */
int XRSAPI xrsPluginInit(int iXRSVer, PCHAR lpServerId, PCHAR lpName, int cbName)
{
    /* Validarea server-ului */
    if ( !xrsValidateServer( lpServerId )) return 0;
    /* Se comunică aplicației numele plug-in-ului */
    strncpy( lpName, PluginName, cbName );
    /* Plug-in inițializat corect, și de tip simplu */
    return 1;
}
#define INSTANCE_HANDLE 1
/* Funcția de creare a instanței */
DWORD XRSAPI xrsPluginStart(HWND hAppWnd, LPRADIODEVCAPS
lpRadioDevCaps, PLUGINPROC lpPluginProc, PDWORD lpFilterFlags)
{
    /* Dacă PluginProc este definită, atunci există deja o instanță a modului */
    if ( PluginProc ) return 0;
    /* Se salvează pointer-ul la funcția callback */
    PluginProc = lpPluginProc;
    /* Nu se primesc notificări */
    *lpFilterFlags = PNF_ALL;
    /* Întrucât este o singură instanță a plug-in-ului, nu se întoarce nimic */
    return INSTANCE_HANDLE;
}
void XRSAPI xrsPluginNotify(HWND hAppWnd, int uMsg, DWORD dwData, int cbData,
LPVOID lpData)
{
    /* Trebuie tratat mesajul de închidere a plug-in-ului */
    if ( uMsg == PN_CLOSE )
    {
        /* Se informează aplicația că plug-in-ul s-a închis */
        PluginProc( INSTANCE_HANDLE, PM_CLOSED, 0, sizeof(PluginName), PluginName );
        /* Se permite pornirea altei instanțe */
        PluginProc = NULL;
    }
    /* Se ignoră toate notificările care mai ar mai sosi */
}
/* Funcția de închidere */
void XRSAPI xrsPluginDone(void)
{
    /* Nu trebuie făcut nimic în această situație! */
}

```

În timpul și după apelarea funcției **xrsPluginStart**, modulul funcțional poate iniția apeluri către aplicația principală. În general, la momentul pornirii, instanța modului funcțional poate obține orice informație de stare și parametrii care îi vor fi utili pe durata existenței și poate face inițializări ale unor controale. El poate pune dispozitivul radio într-o stare particulară convenabilă pentru funcțiile modului funcțional.

Sînt implementate următoarele categorii de notificări:

- Notificări ale aplicației principale: Prin acestea se informează modulele funcționale despre schimbări survenite în interfața dispozitivului radio cu utilizatorul. Aici intră activările și dezactivările diferitelor controale și funcții, maximizarea sau minimizarea interfeței, etc.
- Notificări ale dispozitivelor de recepție: Ele informează modulul funcțional despre toate modificările survenite în starea unui dispozitiv radio receptor, ca de exemplu: frecvența de acord, tipul de demodulator activat la un moment dat, nivelul semnalului recepționat, funcționarea blocului de frecvență intermediară (controlul automat sau manual al amplificării, banda filtrului), funcționarea blocului de audiofrecvență (volumul, filtrare suplimentară), etc.
- Notificări ale dispozitivelor de emisie: Prin ele se informează modulul funcțional despre toate modificările survenite în starea unui dispozitiv radio emițător, ca de exemplu: frecvența purtătoarei, parametrii de modulație, puterea de ieșire, etc.
- Notificări din partea modulului funcțional: Prin ele, instanța modulului funcțional informează alte module funcționale că a fost pornit sau că urmează să își oprească activitatea.
- Notificări de memorie: Prin această categorie de notificări, modulul funcțional este informat atunci cînd au loc schimbări în zonele de memorie aferente dispozitivului radio, cum ar fi: memorii de frecvențe discrete sau benzi de frecvențe, căile către fișiere conexe bazelor de date, etc.
- Notificări din partea blocurilor DSP: Platforma XRS suportă funcții DSP, incluzînd zona de conversie analog-numerică și numeric-analogică, atît pentru dispozitive radio receptoare cît și emițătoare. Această funcționalitate avansată include procesele de modulare și demodulare, codare și decodare, înregistrare și redare (atît în banda de bază a semnalelor cît și în frecvență intermediară), goniometrare, etc. Notificările include cereri și răspunsuri din partea DSP, transferuri de date (singulare sau în blocuri) și schimbări de regimuri și funcții ale blocului DSP.

Într-un mod asemănător cu notificările, comenzile se împart în aceleași categorii:

- Comenzi către aplicația principală: Comenzi care trimise aplicației principale controlează comportamentul interfeței cu dispozitivul radio. Pe lîngă modificările de dimensiuni și vizibilitate ale interfeței se pot introduce restricții temporare asupra acțiunilor pe care le poate exercita utilizatorul (inhibări de controale, mesaje de avertizare, etc), pentru ca acestea să nu interfere cu funcționalitatea modulului.
- Comenzi pentru dispozitivele de recepție: Comenzile pot viza oricare parametru de stare a dispozitivului receptor, pe care acesta (în mod direct sau prin aplicația principală) îl poate modifica. Pentru toate notificările pe care le poate primi modulul funcțional, există o comandă echivalentă, excepția făcînd evident cele legate de nivelul semnalului recepționat.
- Comenzi pentru dispozitivele de emisie: Comenzi ce pot fi trimise aplicației principale pentru a controla toate aspectele funcționării emițătoarelor radio.

- Comenzi pentru module funcționale : Comenzi ce pot fi trimise aplicației principale pentru a obține o listă a tuturor modulelor funcționale instalate și pentru a afla care dintre acestea se află în execuție. Un modul funcțional poate comanda lansarea în execuție sau oprirea execuției altor module funcționale.
- Comenzi de memorie: Un modul funcțional poate trimite comenzi de modificare, adăugare sau ștergere a unor zone din memoriile de frecvențe ale dispozitivelor radio. De asemenea, se pot schimba căile de acces spre fișierele componente ale bazelor de date și comanda preluarea din ele a informațiilor necesare schimbării stării dispozitivului.
- Comenzi pentru blocurile DSP: Platforma *XRS* suportă comenzi separate pentru blocurile DSP ale dispozitivelor radio receptoare sau emițătoare, inclusiv pentru funcțiile de conversie analog-numerică și numeric-analogică. Singura condiție este ca aceste funcții să fie accesibile din exteriorul dispozitivului.

BIBLIOGRAFIE

[1] ----- “*XRS Extensible Radio Specification – Version 1.2*”, WiNRADiO Communications, Melbourne, Australia (www.winradio.com);