Using DLL as Interface between API an VC#.NET Applications

Conf.dr. Marian DÂRDALĂ, lect. Adriana REVEIU, prof.dr. Ion SMEUREANU
Catedra de Informatică Economică, A.S.E. Bucureşti

This paper presents a solution for using complex Win32API data structures and functions in Visual C# .NET applications. We built DLL (Dynamic Link Library) to manage the API functions and data structures and we used DLL modules in a C# application. This is an easier working way compared with the traditional way of importing and managing API's functions in C# programs.

Keywords: *DLL, API (Application Programming Interface), MCI (Media Control Interface), exportable function, internal function, unmanaged code.*

1 Introducere

Interfața de programare a aplicațiilor Windows (Win32 API) este formată dintr-un set consistent de funcții care oferă programelor de aplicații diferite tipuri de servicii.

Dezvoltatorii de software în limbaje precum: C, C++, Pascal etc. folosesc mai mult sau mai puțin, interfețe API în vederea accesării unor resurse sau dispozitive din aplicații Windows. Este cunoscut faptul că structurile şi funcțiile aparținând API conțin membri respectiv parametri, variabile de tip pointer spre diferite tipuri de date. Limbajul C#.NET este un limbaj care nu utilizează pointerii decât în modul *unsafe* iar folosirea funcțiilor, structurilor şi constantelor simbolice implică redefinirea lor folosind tipuri sinonime, adecvate din C#. Acest procedeu de lucru este facil când se apelează funcții izolate din API, dar dacă complexitatea operațiilor de efectuat necesită multe apeluri de funcții şi utilizarea de numeroase variabile de tip structură predefinite pentru API, atunci efortul de programare devine considerabil.

Luând în considerare toate aceste aspecte este utilă definirea de DLL-uri de utilizator, ce exportă funcții, cu rol de interfață, între API şi aplicațiile C#.NET. Astfel, funcțiile exportabile sunt mai uşor de folosit din aplicațiile C#.NET.

2. Structura bibliotecii Win32 API

Biblioteca Win32 API poate fi împărțită, în funcție de serviciile pe care le oferă, în următoarele grupuri:

- **Servicii** Windows **de bază** – conțin servicii pentru depanare, manipularea erorilor, pentru procese, *thread*-uri, fişiere, comunicații interprocese, monitorizarea performanțelor, securitate etc.,
- **Servicii pentru interfața cu utilizatorul**, numite şi servicii utilizator - se ocupă de gestiunea cozii de mesaje, a controalelor, a resurselor, a intrărilor la nivelul utilizatorului,
- **Servicii pentru grafică şi multimedia** – conțin funcții pentru gestiunea culorilor, a GDI (*Graphical Device Interface*), a funcțiilor multimedia, funcții pentru gestiunea secvențelor video, a imaginilor statice, funcții OpenGL, funcții Windows Media,
- **Servicii pentru baze de date şi mesagerie** – conțin funcții pentru gestiunea DAO (*Data Access Objects*), SQL server, MAPI (*Messaging API*),
- **Servicii pentru rețea şi sisteme distribuite** – conțin funcții pentru gestiunea cozii de mesaje, a rețelei, pentru apelul procedurilor aflate la distanță (*Remote Procedure Call*), funcții de *rout*-are şi accesul la distanță, gestiunea sincronizărilor, TAPI (*Telephony API*),
- **Servicii pentru Internet, Intranet şi Extranet** – conțin funcții pentru indexare, funcții pentru manipularea Internet Explorer-ului, serverul web, *NetShow*,
- **Servicii de gestiune a sistemului** – conține servicii de configurare, de gestiune sistemului şi funcții setup.

Fiecare grup de servicii este susținut de un set de componente ale sistemului de operare: subsistemul de DLL-uri al mediului Win32, *drivere*-le, serviciile sistemului de operare. Interfața API este implementată ca un set de

biblioteci cu legare dinamică (DLL-uri), astfel că orice aplicație care utilizează funcții din API realizează legături dinamice.

3. Utilizarea DLL-urilor de sistem în aplicații C#.NET

Bibliotecile cu legare dinamică (DLL) sunt module, în format executabil, care conțin funcții şi date. Un DLL este încărcat la momentul execuției prin referirea modulului. Când un DLL este încărcat, el se mapează în spațiul de adrese al procesului care-l apelează.

Exemplificăm utilizarea bibliotecilor cu legare dinamică pentru a manipula tipul media în aplicații C#.NET, deoarece .NET Framework nu are clase specializate în acest sens. Există totuşi posibilitatea de a utiliza tipul media în C# prin folosirea bibliotecii DirectX dar ea trebuie instalată în prealabil, deci nu oferă o independență totală a programului de aplicație în raport de maşina pe care el rulează. În plus, bibliotecile DirectX actule, implementează un set limitat de funcții multimedia. De aceea, utilizarea interfețelor API şi MCI rămîn soluții viabile pentru programarea multimedia în C# sub Windows.

Apelul funcțiilor din DLL-uri implică utilizarea *namespace*-ului *InteropServices*, în forma:

using System.Runtime.InteropServices;

Prin această declarație se permite utilizarea clasei *DllImportAttribute* prin care se indică funcțiile exportabile care vor fi apelate dintr-o bibliotecă dinamică de tip *unmanaged*.

De exemplu, dacă se doreşte derularea unei secvențe audio, încărcate în memorie, cu funcția *sndPlaySound* având prototipul:

BOOL sndPlaySound(LPCSTR *lpszSound*,
UINT *fuSound*);

atunci ea trebuie redefinită în C# conform tipurilor echivalente şi precizând numele funcției aşa cum ea a fost definită în DLL-ul corespunzător (*winmm.dll*). Fiind vorba de sunetul existent în memorie, parametrul *lpszSound* indică adresa zonei de memorie unde sunetul a fost încărcat, deci nu indică numele fişierului de sunet dat printr-un şir de caractere. Pornind de la aceste ipoteze, prototipul funcției va avea forma:

[DllImport("winmm.dll")] public static extern bool sndPlaySoundA(byte[] buf, uint flag);

unde:

buf – masiv de baiți care conține fişierul de sunet în memorie;

flag – indicator al modului de derulare a secvenței audio;

sndPlaySoundA – numele funcției din DLL;

Funcția returnează o valoare booleană care indică modul în care a decurs apelul funcției (cu succes sau nu). Flagurile se furnizează prin constante simbolice dar în C# acestea trebuie redefinite în forma: public const int SND_MEMORY=0x0004;

Funcțiile importate dintr-un DLL *unmanaged* se includ într-o aplicație C# ca fiind metode statice externe.

In secvența următoare se construieşte un obiect de tip *StreamReader* asociat fişierului care conține secvența de sunet; în vectorul de baiți *buf* se citeşte conținutul fişierului, după care se apelează funcția *sndPlaySoundA* pentru a derula secvența de sunet.

```
        StreamReader strsunet = new
StreamReader("sunet.wav");
                byte[] buf = new
byte[strsunet.BaseStream.Length];
        strsunet.BaseStream.Read(buf,0,(int)st
rsunet.BaseStream.Length);
        sndPlaySoundA(buf,SND_MEMORY);
```

Lucrurile devin complicate când multiple funcții API trebuie apelate, când acestea lucrează cu parametri de tip structură sau cu multe constante simbolice care impun redefinirea lor în C#. Structurile care au membri de tip pointer sunt mai greu de gestionat în C# mai ales dacă se impune alocarea dinamică a zonelor de memorie.

O potențială soluție constă în a crea DLL-uri de utilizator care să redefinească interfața astfel încât să fie cât mai facil de utilizat în C#, după cum se va prezenta în continuare.

4. Construirea DLL-urilor de utilizator ca interfață între API şi aplicații C#.NET

DLL-urile care exportă funcții pot conține două tipuri de funcții: exportabile şi interne. Funcțiile exportabile pot fi apelate din alte module, în timp ce, funcțiile interne pot fi apelate doar în DLL-ul în care ele au fost definite. Datele dintr-un DLL pot fi accesate prin funcții exportabile similare funcțiilor de acces din cadrul claselor.

Bibliotecile cu legare dinamică oferă avanta-

jul modularizării aplicațiilor astfel că funcționalitatea lor poate fi uşor modificată sau îmbunătățită. Un alt avantaj al utilizării DLL-urilor se referă la reducerea consumului de memorie prin faptul că mai multe aplicații care utilizează aceeaşi bibliotecă, în acelaşi timp, au seturi de date distincte dar partajează acelaşi cod executabil.

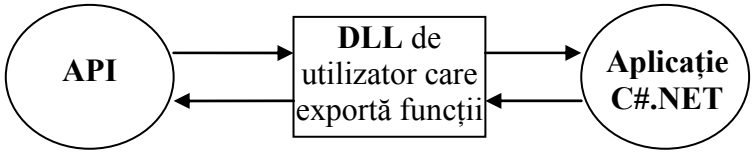

Fig. 1. DLL de utilizator cu rol de interfață

Controlul unei date multimedia de tip video, prin funcții API, presupune parcurgerea unui anumit scenariu. Astfel, se accesează stream-urile, după care ele sunt prelucrate cu funcții specifice în funcție de tipul stream-ului (video, sunet etc.). Pentru exemplificare, se va construi o bibliotecă dinamică, având funcții exportabile, care să permită accesul, la nivel de cadru, al fluxului de imagini pentru o dată video. DLL-ul s-a construit în limbajul C şi s-a folosit biblioteca *vfw32.lib* care s-a adăugat la proiect (*Properties/Linker/Additional Dependencies*).

```
#include <windows.h>
#include <vfw.h>
int cderr=0;
 AVIFILEINFO aviinf;
 PAVIFILE pfile;
 PAVISTREAM avs;
 AVISTREAMINFO avisinf;
 PGETFRAME fr;
 BITMAPFILEHEADER hdr;
char *data;
int sfv;
// functia care deschide fisierul avi ce contine data multimedia
__declspec(dllexport) void deschfv(char *numef)
{
        AVIFileInit();
        LONG hr=AVIFileOpen(&pfile,(LPCTSTR)numef,OF_READ,0L);
        if(hr) { cderr=1; return; }
        AVIFileInfo(pfile,&aviinf,sizeof(aviinf));
        AVIFileGetStream(pfile,&avs,streamtypeVIDEO,0L);
        AVIStreamInfo(avs,&avisinf,sizeof(avisinf));
        fr=AVIStreamGetFrameOpen(avs,NULL);
        BITMAPINFOHEADER *pbim=\
                        (BITMAPINFOHEADER *)AVIStreamGetFrame(fr,AVIStreamStart(avs));
        sfv=pbim->biSize+pbim->biClrUsed*sizeof(RGBQUAD)+pbim->biSizeImage;
        data = new char[sfv+sizeof(BITMAPFILEHEADER)];
        hdr.bfType = 0x4d42;
        hdr.bfSize = (DWORD)(sizeof(BITMAPFILEHEADER) + pbim->biSize +
                                    pbim->biClrUsed * sizeof(RGBQUAD));
        hdr.bfReserved1 = hdr.bfReserved2 = 0;
        hdr.bfOffBits = (DWORD)sizeof(BITMAPFILEHEADER) + pbim->biSize +
                                            pbim->biClrUsed * sizeof(RGBQUAD);
        memcpy(data,&hdr,sizeof(BITMAPFILEHEADER));
        memcpy(data+sizeof(BITMAPFILEHEADER),pbim,sfv);
}
// functii pentru obtinerea latimii / inaltimii unui cadru
__declspec(dllexport) int get_latime()  {    return aviinf.dwWidth;   }
__declspec(dllexport) int get_inaltime()  { return aviinf.dwHeight;  }
// functie care returneaza numarul de cadre al fluxului de imagini
__declspec(dllexport) int get_nrf() {  return AVIStreamLength(avs);  }
// functie care returneaza un cadru, identificat prin numarul lui
__declspec(dllexport) char *get_frame(int nrf)
{
        memcpy(data,&hdr,sizeof(BITMAPFILEHEADER));
        memcpy(data+sizeof(BITMAPFILEHEADER),AVIStreamGetFrame(fr,nrf),sfv);
        return data;
}
// functie care returneaza marimea, in baiti, a unui cadru
__declspec(dllexport) int get_sizeframe() {  return sfv+sizeof(BITMAPFILEHEADER);  }
// functie care inchide sesiunea de lucru cu data video
__declspec(dllexport) void inchide()
```

```
{
        delete []data;
        AVIStreamGetFrameClose(fr);
        AVIStreamRelease(avs);
        AVIFileRelease(pfile);
        AVIFileExit();
}
```

Se observă că funcțiile exportabile care constituie interfața cu aplicațiile ce vor folosi această bibliotecă au parametri şi returnează valori de tipuri fundamentale, deci folosirea DLL-ului din aplicații C# nu necesită redefiniri complexe de tipuri de date. In plus, toate tipurile definite pe baza structurilor (*AVIFILEINFO*, *BITMAPINFOHEADER*, *AVISTREAMINFO* etc) sunt *ascunse* utilizatorului la fel ca şi funcțiile care primesc pointeri la structuri complexe ori lucrează cu constante simbolice (*AVIFileOpen*, *AVIFileGetStream*, *AVIStreamInfo*, *AVIStreamGetFrameOpen* etc).

Utilizarea bibliotecii create, într-o aplicație C#.NET, necesită redeclararea funcțiilor exportabile după cum urmează:

```
[DllImport("test_dll.dll")] public static extern void deschfv(string numef);
[DllImport("test_dll.dll")] public static extern int get_nrf();
[DllImport("test_dll.dll")] public static extern int get_inaltime();
[DllImport("test_dll.dll")] public static extern int get_latime();
[DllImport("test_dll.dll")] public static extern IntPtr get_frame(int nrf);
[DllImport("test_dll.dll")] public static extern int get_sizeframe();
[DllImport("test_dll.dll")] public static extern void inchide();
```

Se observă că funcția *get_frame* care returnează cadrul propriu-zis, printr-un vector de baiți a fost redeclarată cu ajutorul tipului *IntPtr*, folosit în C# pentru compatibilitate cu tipul pointer sau handle din C.

Scenariul care trebuie urmat, constă din succesiunea operațiilor:

⇒ inițializare:

- se declară două variabile, una de tip întreg şi un vector de baiți

```
int szf;
byte [] pz;
```

- se deschide fişierul care conține video-ul

```
deschfv("d:\\mar\\dll\\f2.avi");
```

- se obține dimensiunea în baiți a unui cadru (imagine fixa)

```
szf=get_sizeframe();
```

- se crează zona de memorie în care se va copia cadrul

```
pz=new byte[szf];
```

⇒ prelucrare (afişarea cadrului):

- obținerea cadrului *nrc* din fluxul de imagini al filmului sub forma unui pointer

```
IntPtr sfr=get_frame(nrc);
```

- copierea în vectorul de baiți, identificat prin *pz*, a cadrului referit de variabila *sfr*

```
Marshal.Copy(sfr,pz,0,szf);
```

Clasa *Marshal* conține metode pentru alocarea zonelor de memorie *unmanaged,* pentru conversia din tipuri de date *managed* în tipuri *unmanaged* şi pentru interacțiunea cu codul *unmanaged*.

- crearea unui stream de date, în memorie, din vectorul *pz*

```
MemoryStream ms=new MemoryStream(pz);
```

- crearea unui obiect Bitmap din streamul *ms*

```
Bitmap bmp = new Bitmap(ms);
```

- afişarea imaginii într-un *PictureBox* (*pb*) prin setarea proprietății *Image* cu bitmap-ul *bmp*

```
pb.Image=bmp;
```

⇒ terminare:

- se finalizează lucrul cu data video

```
inchide();
```

5. Concluzii

Avantajele oferite de acest mod de lucru nu se referă doar la manipularea tipului media. Există programatori care au creat aplicații în tehnologia SDK şi preferă să reutilizeze codul deja scris pentru a realiza anumite operații, chiar dacă acum dezvoltă aplicațiile în C#.NET.

Bibliografie

⇒ * * *, *Microsoft Developer Network Library,* Microsoft Press, 2005;

⇒ Rimmer, S., *Multimedia Programing for Windows*, McGraw-Hill, 1994;

⇒ Smeureanu, I., Dârdală, M., *Multimedia Programming Objects,* Al cincilea Simpozion de Informatică Economică, A.S.E., Bucureşti, 2001;

⇒ Yuan, F., *Window Graphics Programming Win32 GDI and DirectDraw*, Prentice Hall, 2000