5. Laborator

5.1. Scopul laboratorului

Acest laborator are următoarele scopuri:

1. Înțelegerea conceptelor ce stau în spatele arhitecturii de tip Document/View;
2. Încărcarea unei imagini drept resursă în executabilul ce va fi generat;
3. Lucrul cu elemente grafice (imaginii și culori);
4. Cum se utilizează meniurile;
5. Crearea de ferestre de dialog și asocierea acestora cu o clasă corespondentă.

6.2. Cerințe laborator

Dezvoltați un program de tip ***"Single document interface"*** în mediul Visual Studio care:

1. În momentul pornirii să afișeze o imagine, ca în Figura 5.1. Imaginea afișată va fi încărcată anterior drept resursă a programului. *Background*-ul foii documentului va fi de culoar neagră iar afișarea imaginii și colorarea *background*-ului se va realiza în cadrul unei funcții ce va trata mesajul WM_ERASEBKGND.
2. Va avea un meniu capabil să lanseze în execuție o fereastră similară cu cea din figura de mai jos. Butonul start va porni un element de tip ***"Progress Control"***. Butoanele **OK** și **Cancel** vor avea funcționalitatea clasică.

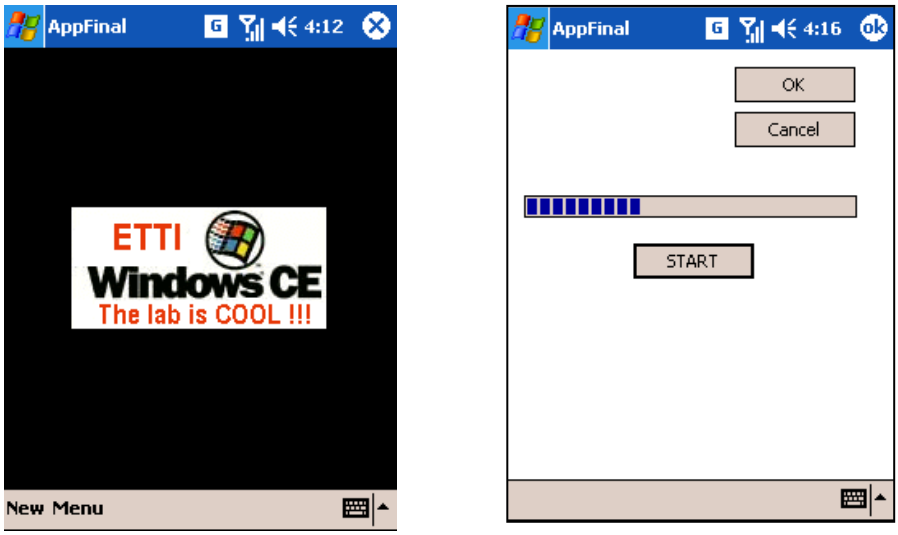

Figura 5.1. Interfețele cu utilizatorul

5.3. Arhitectura programelor "Single document interface"

Deoarece se va crea un program de tip ***"Single document interface"*** (conceptual aceste programe sunt similare ca funcționalitate programului *Notpad*) trebuie urmăriți în mod generic toți pașii prezentați în cadrul Laboratorului 2 cu câteva excepții.

Astfel atunci când se ajunge la fereastra care ne permite să selectăm tipul aplicației (Application Type) se alege opțiune "*Single Document*" și nu "*Dialog Base*" cum realizam anterior. În cadrul ferestrei următoare "*Document Template Strings*" se dă *Next* iar în cadrul ferestrei "*User Interface Features*" se lasă opțiunea implicită "*Menus only*". În ultima fereastră "*Generated Classes*" se observă că *Wizard*-ul va genera 4 clase: C...App, C...View, C...Doc și C...Frame. Aceste clase definesc arhitectura de tip Document/View și sunt derivate din alte clase mai generice și au următoarele roluri:

1. O clasă document, care este derivată din clasa CDocument, și care este responsabilă de datele programului;
2. O clasă de vizualizare derivată din CView. Această clasă este interfața dintre datele programului gestionate de clasa C...Doc (CDocument) și utilizatorul care le manipulează;
3. O clasă de tip *frame* care conține partea de vizualizare, C...View, și orice alt element de interfațare cu utilizatorul – precum meniuri. Această clasă este derivată din clasa CFrameWnd și
4. Clasa aplicației globale (C...App), derivată din CWinApp, ce este responsabilă cu lansarea programului și managementul anumitor interacțiuni cu sistemul de operare.

În Figura 5.2 se prezintă legăturile existente între diferitele clase ce compun un program de tip ***"Single document interface"***.

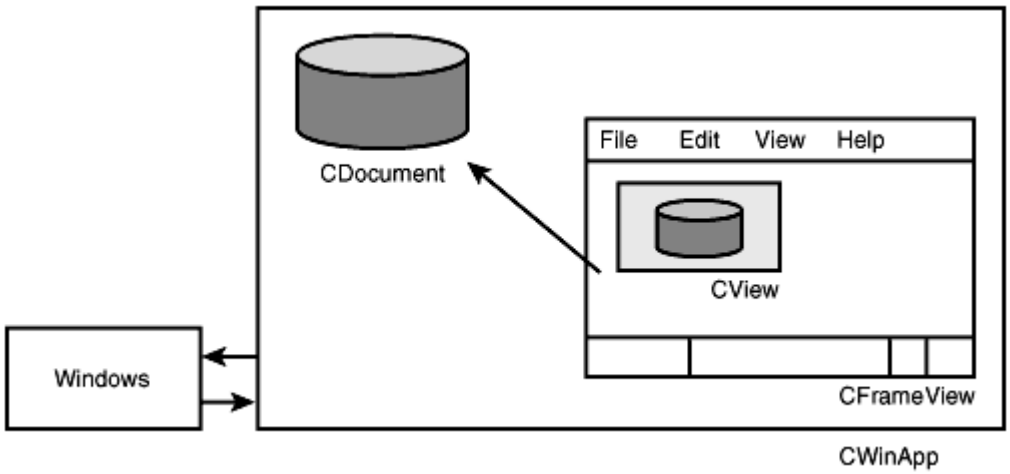

Figura 5.2. Arhitectura de tip Document/View ce se află la baza programelor de tip "*Single document interface*"

Pentru a verifica corectitudinea funcționării programului, până în acest moment, în cadrul funcției OnDraw inserați codul de mai jos și compilați programul. Funcția OnDraw este apelată atunci când se afișează o anumită informație pe ecran, se tipărește la imprimantă o anumită informație sau doar se realizează un *preview* a informației ce se dorește a fi tipărită – mai general, această funcție este chemată atunci când se dorește ca un document să fie afișat. La fiecare apelare specifică funcția primește adresa altui *device context* (DC).

```
void CLabDlg_GrafView::OnDraw(CDC* pDC)
{
	CLabDlg_GrafDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	// TODO: add draw code for native data here
	RECT TextArea = {1, 1, 400, 100};
	pDC->DrawText(L"Merge !!!", &TextArea, DT_CENTER | DT_SINGLELINE | DT_VCENTER);
}
```

5.4. Elementele de grafică

Afișarea imagini și schimbarea culorii *background*-ului o vom realiza în cadrul funcției asociate cu mesajul WM_ERASEBKGND. Această funcție va fi internă clasei „C…View" (în cadrul aplicației dvs. punctele, "…", sunt înlocuite de un nume specific, nume furnizat în momentul inițial când construiți aplicația). Pentru aceasta urmați pașii:

1. Se selectează clasa „C…View" (dând click pe ea – click stânga);
2. Din fereastra *Properties* se apasă butonul *Messages*;
3. Din lista de mesaje, apărută, se selectează mesajul WM_ERASEBKGND și se asociază o funcție acestuia, selectându-se funcția implicită: OnEraseBkgnd.

După cum s-a prezentat în laboratorul precedent acest mod de tratare a unui mesaj se poate generaliza și aplica în tratarea oricărui alt mesaj al sistemului de operare.

Pentru a ușura procesul de afișare a unei imaginii aceasta, în primul pas, va fi încărcată drept resursă a programului, iar ulterior va fi afișată. Pentru afișarea unei imagini urmați pașii:

1. Se încarcă imaginea drept o nouă resursă:
   a. Selectați *tab*-ul "*Resource View*" în mediul Visual Studio;
   b. Aici importați fișierul Win_EC.bmp din directorul de lucru: click dreapta pe *tab*-ul selectat → alegeți *Add Resurce…* → selectați Bitmap → apăsați butonul Import → din structura de directoare selectați fișierul Win_EC.bmp → OK;
   c. Selectând imagina inclusă în proiect drept resursă redenumiți **ID**-ul în IDB_WINEC_BITMAP.
2. Se încarcă în memorie noua imagine pentru a o utiliza ulterior:
   a. Se declară o variabilă de tip CBitmap (de exemplu, CBitmap bmp). Clasa CBitmap conține atât o structură de date în care se va stoca imaginea cât și funcțiile ce vor procesa această imagine (aceste date);
   b. Se încarcă imaginea în variabila bmp prin intermediul funcției LoadBitmap, funcție internă clasei CBitmap:

      bmp.LoadBitmap(IDB_WINEC_BITMAP);
3. Pentru aflarea diferitelor informații în legătură cu imaginea încărcată se:
   a. Definește o structură de date de tipul: BITMAP bmpStruct;
   b. Cu ajutorul funcției bmp.GetBitmap(&bmpStruct) se preiau diferitele informații caracteristice imaginii (dimensiuni, număr plane de culoare etc.) ce o caracterizează și se salvează în structura de date bmpStruct;

Exercițiu: Identificați membrii structurii BITMAP și înțelegeți rolul fiecăruia din ei.

4. Pentru determinarea coordonatelor spațiului activ de afișare:
    a. CRect clientRect – definiți structura de date ce va stoca coordonatele zonei active unde poate avea loc afișarea;
    b. GetClientRect(&clientRect) – preluați datele;
5. Atunci când veți afișa imaginea prin intermediul funcției BitBlt veți avea nevoie să introduceți coordonatele poziției colțului stânga sus a acestei imagini relative la spațiul de afișare.

    Exercițiu: Deoarece imaginea trebuie să fie centrată în mijlocul zonei de afișare, determinați coordonatele de mai sus cu ajutorul informațiilor stocate în structurile de date clientRect și bmpStruct.
6. Se creează un suport de memorie, similar (memorie video) cu cel în care se va afișa imaginea, unde imaginea se va depozita:
    a. CDC memdc – declarare date;
    b. memdc.CreateCompatibleDC(pDC) – creare cadru compatibil DC. Variabila pDC a fost primită drept argument de funcția OnEraseBkgnd.
7. memdc.SelectObject(&bmp) – preluare imagine în DC;
8. Prin intermediul unei operații de tip **"bit block transfer"** se va copia imaginea. Astfel: pDC->BitBlt(, , , , &memdc, 0, 0, SRCCOPY) – copiere imagine din cadru memorie DC în zonă de afișaj DC. Această funcție impune trimiterea primelor două argumente drept coordonatele unde se va face afișare, iar următoarele două argumente cuantizează lățimea și înălțimea imaginii.

    Având informațiile din structurile clientRect (coordonatele zonei de afișare) și bmpStruct (datele despre imagine) puteți determina coordonatele necesare afișării imaginii bitmap în centrul zonei active de afișare.

Pentru a schimbare culoare *background*-lui trebuie să lucrați cu un element de tip ***brush*** (pensulă):

1. CBrush backBrush(RGB(0,0,0)) – creare element pensulă și setare culoare (negru);
2. pDC->FillRect(&clientRect, &backBrush) – umplere *background*, dacă clientRect este obținut cu ajutorul funcției GetClientRect(&clientRect) ca mai înainte și nu se modifică structura variabilei clientRect atunci se schimbă culoarea întregului background;
3. Dacă dorim salvarea și refacerea ulterioară a culorii inițiale a *beckground*-ului trebuie să urmăm pașii:
    a. Se salvează vechea culoare simultan cu utilizarea celei noi prin:

        CBrush* pOldBrush = pDC->SelectObject(&backBrush)

    b. Urmată ulterior, când se dorește de refacerea noii culori, de utilizarea funcției:

        pDC->SelectObject(pOldBrush)

5.5. Meniuri

Pentru introducerea unui nou meniu și asocierea unei funcții care va trata selectarea meniului, urmați pașii:

1. Selectați *tab*-ul *Resurce View* → dați dublu click pe Meniu → IDR_MAINFRAME și înserați un nou câmp;
2. Pe câmpul înserat dați click dreapta și alegeți opțiunea: ***Add Event Handler …***

3. Selectați clasa CMainFrame unde veți însera funcția ce va trata activarea meniului introdus de dumneavoastră;
4. Apăsați butonul **Add** and **Edit**.

5.6. Ferestre de dialog

Pentru a crea o nouă fereastră de dialog și a o afișa trebuie să asociem o nouă clasa cu această fereastră pe care ulterior să o utilizăm. Pașii sunt:

1. Creați-vă o nouă fereastră de dialog: *Resurce View* → dați dublu click pe Dialog → click dreapta → Insert Dialog → Properties → schimbați ID ferestrei IDD_MY_DIALOG (sau orice alt nume relevant);
2. Plasați-vă elementele cerute pe acest nou dialog - toți pașii necesari sunt cunoscuți din laboratoarele precedente;
3. Asocierea clasă cu fereastra nou creată: click dreapta → Add Class → în câmpul „Class name:" introduceți un nume, de exemplu MyDialog (observați cum câmpurile ".h file:" și ".cpp file:" se completează simultan cu introducerea numelui dat de dvs.) → Atenție ! La "Base class:" selectați CDialog (CDHtmlDialog nu este suportată de SO Windows Embedded Compact) → Finsh.

   Observați cum în fereastra Class View a mai apărut o clasă cu numele de MyDialog. Această clasă este descrisă de un prototip ce se găsește în fișierul MyDialog.h și de funcțiile interne clasei ce se găsesc în fișierul MyDialog.cpp.

4. Asociați o funcție cu butonul Start, vezi Figura 5.1, și o variabilă cu elementul de tip *Progress Control*;
5. Petru ușurința atingerii obiectivelor propuse se pot utiliza următoarele funcții cu elementul de tip *Progress Control*: SetRange, SetStep și StepIt.

Ultimul pas pe care trebuie să-l mai facem este să afișăm această fereastră atunci când meniul introdus de noi a fost selectat. Pentru aceasta pașii:

1. În funcția asociată cu meniul introdus (din clasa CMainFrame) introduceți o nouă variabilă de tip clasă:

   ```
   MyDialog dialog;
   ```

   Această variabilă de tip dialog este tocmai clasa generată și asociată cu fereastra de dialog creată de noi – deoarece descrierea prototipului clasei o avem în MyDialog.h acest fișier trebuie în mod obligatoriu inclus.
2. Pentru afișarea ferestrei folosiți codul:

   ```
   dialog.DoModal();
   ```