Costruzione di Interfacce
Lezione 17
Primi passi MFC 2

cignoni@iei.pi.cnr.it
http://vcg.iei.pi.cnr.it/~cignoni

Menu

- Aggiungiamo un menu' per far partire il dialogo.
- Al solito i menu'sono risorse.
- Ad ogni entry di un menu' è associato un id (definito al solito in resource.h)
- Quando si sceglie un menu' parte un messaggio WM_COMMAND che contiene quell'id.
- L'ide e MFC semplificano (nei casi normali) la gestione del mapping di questi messaggi:

```
BEGIN_MESSAGE_MAP(CHelloApp, CWinApp)
  ON_COMMAND(ID_ABOUT_TEST, OnAboutTest)
END_MESSAGE_MAP()
```

Menu

- Il menu creato lo nella init instance attacchiamo alla finestra principale.
- Si fa a mano perchè stiamo facendo tutto a mano…

```
BOOL CHelloApp::InitInstance(){
   m_pMainWnd = new CHelloWindow();
   m_pMainWnd->ShowWindow(m_nCmdShow);
   m_pMainWnd->UpdateWindow();
   CMenu mm; mm.LoadMenu(IDR_MENU1);
   m_pMainWnd->SetMenu(&mm);
   return TRUE;
}
```

Risorse in linea

- Dove trovare nell'help cose da leggere ad alto livello:
- Visual C++
  - Adding Functionalities
    - Visual C++ Libreries
      - MFC
        - General MFC Topics
        - MFC Overview

MFC framework

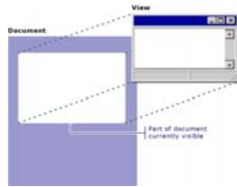
- MFC sono generali e permettono di scendere molto a basso livello nella definizione della struttura dell'applicazione
- MFC mettono anche a disposizione un paio di framework tipici entro cui strutturare le proprie applicazioni:
  - Dialog based application
  - Document/view application

Il framework Document View

- La struttura di molte applicazioni *vere* è basata sui concetti di *documento* e *vista*:
- Un'applicazione serve all'utente per creare, modificare e gestire un documento.
- Due tipi principali di interfaccia
  - SDI *Single Document Interface* (notepad, I.E.) ogni istanza dell'applicazione gestisce un solo documento
  - MDI *Multiple Document Interface* (Excel, word), ogni istanza dell'applicazione gestisce più documenti

Documenti e viste

- Un documento è un data object con cui l'utente interagisce durante una sessione di editing. E' creato con i comandi New o Open dell'applicazione nel menu File ed è tipicamente salvato in un file.
- Una vista è un window object attraverso cui l'utente interagisce con il documento



Documenti

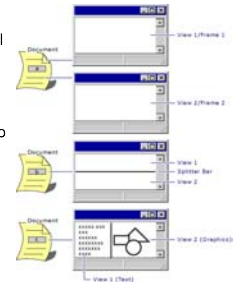
- Insieme documenti e viste:
- Contengono, gestiscono e mostrano i dati propri dell'applicazione
- Gestiscono la maggior parte dei messaggi dell'applicazione e dei comandi

Viste

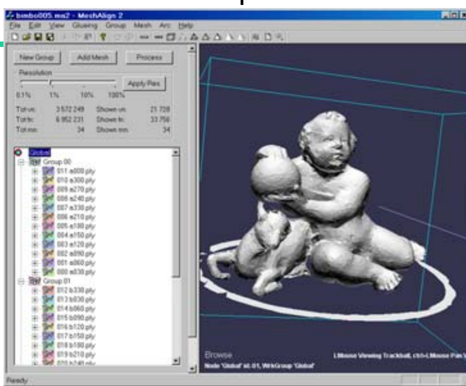
- Devono fare il rendering dei dati dell'applicazione
- Devono permettere l'editing dei dati stessi
- Non contengono dati, se non quelli relativi ad un particolare modo di mostrare I dati stessi
- Ad es. in un viewer 3d
  - il colore dello sfondo
  - La posizione della camera, ecc.

Viste

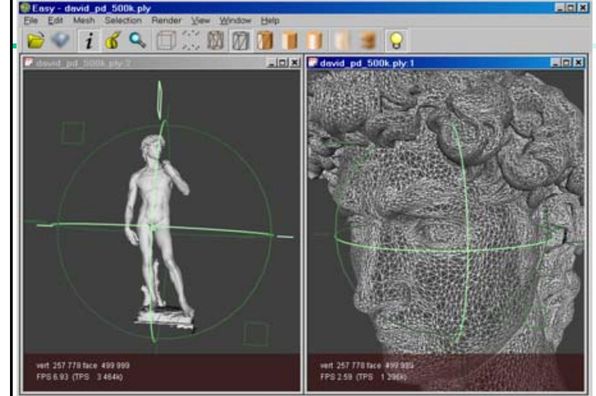
- Piu' viste per ogni documento
  - Piu istanze di una stessa classe: il mio documento è mostrato in più finestre all'interno della stessa finestra principale o in finestre diverse
  - Più classi vista per uno stesso tipo di documento, ho modi fondamentalmente diversi di vedere lo stesso documento
- Un solo documento per ogni vista



Viste di tipo diverso

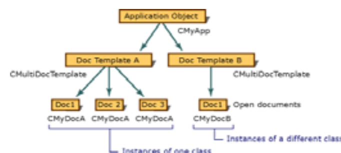


Più viste per uno stesso documento



Document Template

- Tutte le relazioni tra documenti viste e finestre di un’applicazione sono gestite dalla classe CDocTemplate
- Gli oggetti CDocTemplate sono gestiti dall’application object (quello derivato CWinApp)



CDocTemplate

- CDocTemplate è una classe base astratta
  - CSingleDocTemplate
  - CMultiDocTemplate
- Mettono in relazione
  - Una Document Class (derivata da Cdocument)
  - Una View Class (derivata da Cview)
  - Una frame window class (derivata da CFrameWindow o CMDIChildWindow)

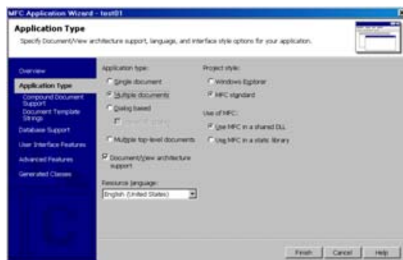
Document Template

- Ogni app ha un doctemplate per ogni tipo di documento che supporta.
- Tra le altre cose contiene l’id della risorsa stringa che contiene la descrizione testuale del tipo del documento (nome, estensione ecc)

Riassunto Relazioni

- Un ***documento*** tiene una lista delle sue viste e un puntatore al DocTemplate che lo ha creato
- Una ***vista*** tiene un puntatore al documento che sta mostrando e è figlia di una frame window
- Una ***document frame window*** tiene un puntatore alla finestra correntemente attiva
- Un ***document template*** tiene la lista dei documenti aperti
- ***L’applicazione*** tiene la lista dei document template
- Windows tiene traccia di tutte le finestre aperte

Partiamo con il wizard



Wizard

- Quattro tipi di Applicazione
  - SDI Single document
  - MDI Multiple Document
  - Dialog based: l’app non ha una struttura doc/view, ma è semplicemente un dialogo (come la calcolatrice)
  - Multiple Top level doc: come se il main frame fosse il desktop (ogni volta che faccio new il document frame non sta nella client area del main frame dell’applicazione)

SDI Example

- Il wizard crea 15 file e 5 classi, i cui nomi dipendono dal nome del progetto, derivate rispettivamente da
  - CWinApp: l'application object
  - CDocument: la classe per gestire il documento
  - CView: la classe per gestire la visualizzazione del documento
  - CFrameWnd: la finestra
  - CDialog: un dialogo di about

WinApp

- Overridden InitInstance
  - Tra le molte cose che fa
  - DocTemplate
  - Lettura registro
  - Parsing commandline
- OnAppAbout() gestore del comando del menu' che fa apparire il dialogo dell'about

Dialog Box

- Sono definiti nelle risorse, o meglio nelle risorse si definisce un template da usare per creare una classe che incapsula quell'oggetto
- Ad ogni dialogo si associa una classe che lo gestisce, viene fatto dall'ide…
- Per far vedere un dialogo si deve creare un oggetto di quel tipo e poi farlo partire
- Il modo più semplice per invocare un dialogo è quello di farlo modale:
  - L'applicazione che lo ha invocato è fermata (ragionevolment) finchè non si preme ok o cancel.

```
CAboutDlg dlgAbout;
dlgAbout.DoModal();
```

Document

- Proviamo ad aggiungere qualcosa, disegno di alcuni pallini nella finestra principale
- Nel document aggiungo un vettore per tenere i centri

```
// Attributes
public:
   vector<CPoint > cc;
```

- Nel .h di tutta l'applicazione aggiungo

```
#include <vector>
using namespace std;
```

View

- Nella OnDraw

```
void CTest002View::OnDraw(CDC* pDC)
{
   CTest002Doc *pd=GetDocument();
   for(int i=0;i<pd->cc.size();++i)
        pDC->Ellipse(CRect(pd->cc[i],pd->cc[i]+CPoint(15,15)));
}
```

- E il gestore dell'evento rilascio del bottone del mouse sinistro

```
void CTest002View::OnLButtonUp(UINT nFlags, CPoint point)
{
   CView::OnLButtonUp(nFlags, point);
   GetDocument()->cc.push_back(point);
   Invalidate();
}
```

Load e save

```
void CTest002Doc::Serialize(CArchive& archive)
{
   if (archive.IsStoring()){
     archive << cc.size();
     for(int i=0;i<cc.size();++i)
         archive << cc[i];
   }else{
       cc.clear();
       int num;
       archive >> num;
       cc.resize(num);
       for(int i=0;i<cc.size();++i)
         archive >> cc[i];
   }
```

Load e Save

- Tutti gli oggetti MFC derivati da Cobject possono essere serializzati se si overrida il membro serialize
- Si aggiunge nella dichiarazione
  `DECLARE_SERIAL( CMyObject )`
- Si aggiunge nel .cpp
  `IMPLEMENT_SERIAL(CMyObject,CObject,1)`
- La serializzazione è fatta tramite una classe CArchive che virtualizza le operazioni di lettura/scrittura buffered.

Considerazioni

- Il meccanismo di serializzazione delle mfc è generale e facile da usare se si hanno nel nostro documento tutti e soli oggetti mfc.
- In generale capita spesso che, specialmente se si deve rispettare un qualche standard di formato di file, si faccia overriding diretto delle funzioni OnOpenDocument e OnSaveDocument che ricevono in ingresso il nome del file…