МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра интеллектуальных информационных технологий

Методическое пособие

“РАЗРАБОТКА ПРИЛОЖЕНИЙ НА БАЗЕ КАРКАСА ДОКУМЕНТ-ВИД (мастер MFC AppWizard)”

БРЕСТ 2012

УДК 681.3 (075.8)
ББК с57

Методическое пособие предназначено для знакомства с основами создания пользовательских Windows-приложений в системе Microsoft Visual Studio C++ средствами мастеров на базе автоматического каркаса документ-вид, поддерживаемого библиотекой MFC. Рассматриваются общие принципы создания и просмотра документов, а также примеры разработки типовых приложений.

Составители: Муравьев Г.Л., к.т.н., доцент
Мухов С.В., к.т.н., доцент
Хвещук В.И., к.т.н., доцент

ОГЛАВЛЕНИЕ

ТИПОВОЙ КАРКАС ДОКУМЕНТ-ВИД

1.1. Общая характеристика функциональных возможностей автокаркасов

Библиотека MFC поддерживает разнообразные каркасы, используемые при создании пользовательских приложений. Каркасы могут быть построены вручную либо автоматически, если в системе программирования есть специальные программы – мастера, генерирующие тексты каркасов в соответствии с заданными пользователем требованиями, параметрами. Полученные каркасы затем дорабатываются с целью придания проекту необходимой функциональности и реализации разрабатываемого приложения.

В системе программирования Visual Studio C++ ряд типовых широко используемых каркасов можно получить, в частности, с помощью мастера MFC AppWizard (exe). Такие каркасы и их характеристики представлены рисунками 1-4.

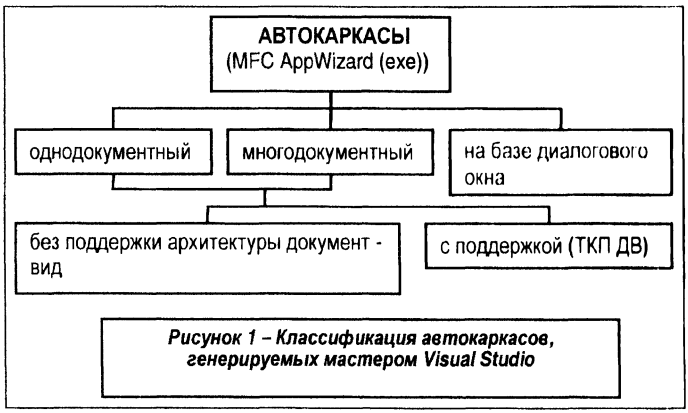

Рисунок 1 – Классификация автокаркасов, генерируемых мастером Visual Studio

Это одно-и многодокументные каркасы соответственно с интерфейсом SDI (Single Document Interface) и MDI (Multiple Document Interface) и каркасы с интерфейсом на базе диалогового окна, используемого в качестве главного.

Первые два каркаса могут базироваться на одной из двух архитектур приложения. Это типовой каркас приложения (ТКП) - архитектура создания приложений - "окно с рамкой" (CFrameWnd) в роли главного для организации всех функций по обработке данных (документов), включая их хранение, визуализацию. Либо типовой каркас приложения (ТКП ДВ) - архитектура создания приложений документ-вид, рассматриваемый далее.

Здесь объект обработки разделяется на объекты класса ВИД (производного от класса CView) и объекты класса ДОКУМЕНТ (производного от класса CDocument). С каждым из документов может быть связан один или несколько объектов классов, производных от класса ВИД.

При этом функциональность класса ВИД обеспечивает, определяет оконный вид, облик, представление документа (на экране, при печати), а функциональность класса ДОКУМЕНТ обеспечивает типовые действия по работе с данными документа и, таким образом, позволяет представлять более абстрактные объекты, чем документ, понимаемый как объект обработки текстового процессора.

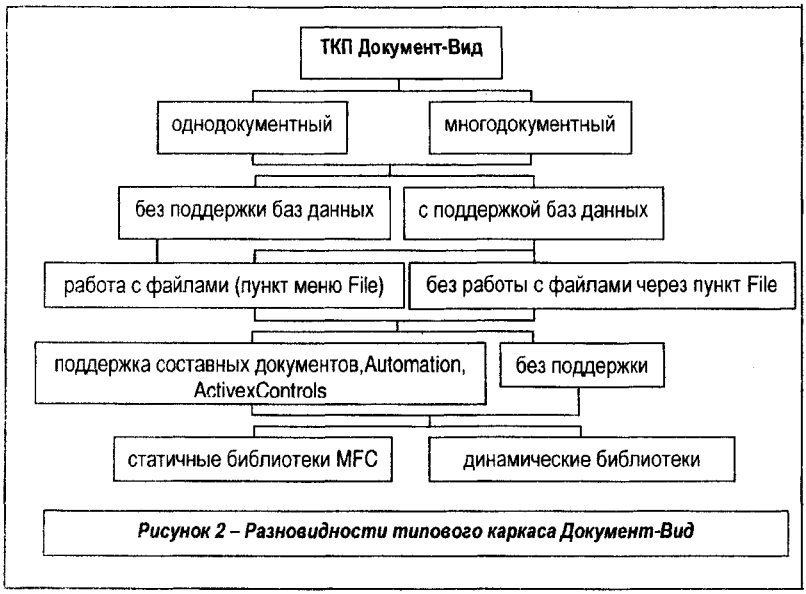

Рисунок 2 – Разновидности типового каркаса Документ-Вид

Базовая специализация указанной архитектуры (ТКП ДВ) – разработка приложений для создания, редактирования, хранения, печати и просмотра документа (-ов), в том числе в различных форматах отображения.

Более развернуто разновидности автокаркасов ТКП ДВ, получаемые при различных настройках, задаваемых пользователем при генерации, представлены на рисунке 2.

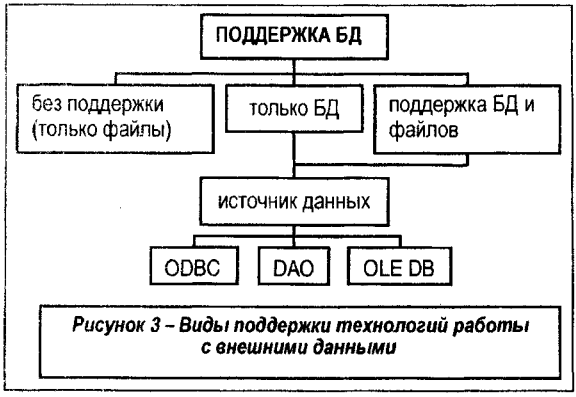

Рисунок 3 – Виды поддержки технологий работы с внешними данными

Отличительной особенностью всех этих каркасов является поддержка механизма сериализации документов – типовых пунктов подменю File для автоматической загрузки документа для работы из выбранного файла и автоматической выгрузки в выбранный файл по завершении работы.

Кроме этого, каркас может быть настроен и на работу с базами данных с поддержкой наиболее распространенных стандартов и технологий, приведенных на рисунке 3.

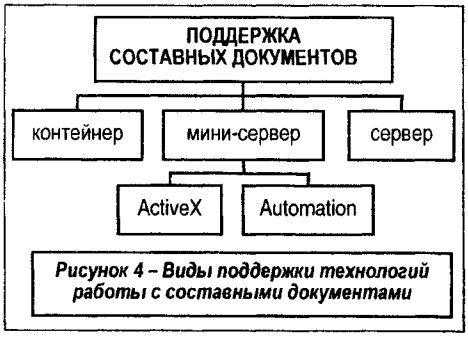

Рисунок 4 – Виды поддержки технологий работы с составными документами

Каркасы ТКП ДВ могут использовать технологии работы с составными документами, позволяя работать с разнотипными документами, реализованными и поддерживаемыми разными программными средствами, например, электронными таблицами Excel, процессорами Word и т.п. Указанное иллюстрируется рисунком 4.

1.2. Классы MFC для обеспечения функциональности видов документов

Окна просмотра в приложениях архитектуры документ-вид базируются на возможностях класса CView и классов, наследуемых от него (рисунок 5). Эти классы предоставляют средства для управления окнами просмотра документов приложения на экране дисплея, а также средства вывода документов на печать. Они же могут быть использованы и для организации редактирования документов.

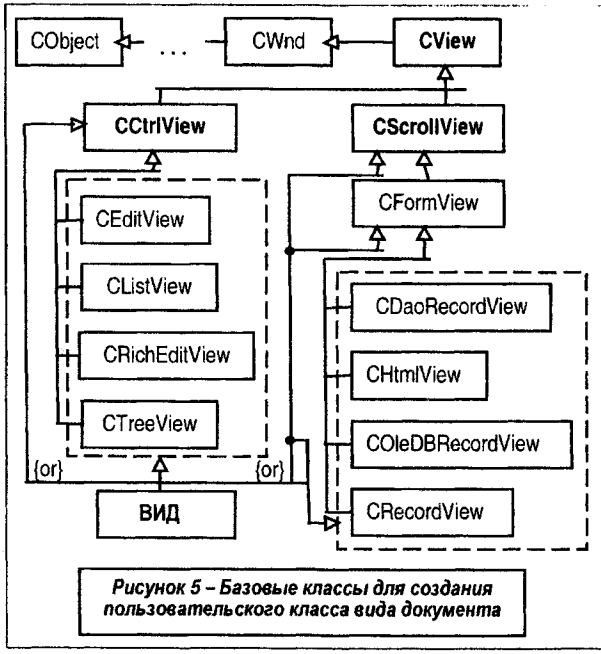

Рисунок 5 – Базовые классы для создания пользовательского класса вида документа

Выделяют разные типы окон вида документа. Это:

- окна, построенные на базе стандартных элементов управления (ЭУ);
- окна, обеспечивающие прокрутку документов в поле окна;
- окна, базирующиеся на формах, подобных диалоговым окнам.

Управление окнами первого типа в MFC требует использования методов двух классов (рисунок 6). Методы, непосредственно связанные с созданием объекта вида на базе ЭУ, инкапсулированы в соответствующем классе вида, который в большинстве случаев представляет собой всего лишь класс-оболочку с минимальным набором членов, через которую и можно получить доступ к методам соответствующего ЭУ. Как правило, для этого используются методы типа GetTreeCtrl, GetEditCtrl и т.п., позволяющие получить указатель на соответствующий ЭУ. А специфические действия, присущие используемому элементу управления, инкапсулированы в его собственном классе.

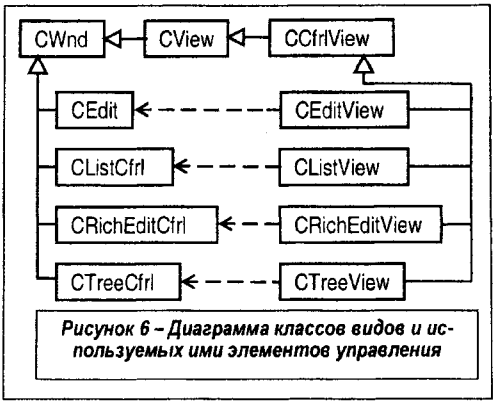

Рисунок 6 – Диаграмма классов видов и используемых ими элементов управления

Соответственно пользователь, исходя из особенностей разрабатываемого приложения, должен определить - от какого из классов вида (базового CView или классов, производных от него в библиотеке MFC) следует наследовать собственные классы просмотра документов.

Как видно из рисунка 5, классы, унаследованные от CView, также разделяются на группы.

Первую группу составляют классы, порождаемые от класса CCtrlView - базового класса вида для элементов управления Win95. Они обеспечивают поддержку окон вида, используемых для отображения документов, формируемых на основе готовых элементов и расширенных элементов управления, обеспечивающих просмотр и редактирование данных.

Здесь используются следующие классы вида:

- класс CEditView, базирующийся на элементе управления Поле редактирования класса CEdit и обеспечивающий основные функции редактирования текстового документа;
- класс CRithEditView, базирующийся на элементе управления Поле редактирования класса CRichEditCtrl и обеспечивающий расширенный набор функций по редактированию и форматированию текстового документа;
- класс CTreeView, базирующийся на элементе управления класса CTreeCtrl и обеспечивающий набор функций для работы, просмотра иерархических структур данных;
- класс CListView, базирующийся на элементе управления Список класса CListCtrl и обеспечивающий основные функции отображения и редактирования списка пиктограмм.

Другую группу составляют классы, порождаемые от класса CFormView, позволяющего создавать окна просмотра документов на основе диалогового окна, диалоговой панели.

Здесь, в частности, используются следующие классы вида:

- класс CRecordView, предназначенный для работы, просмотра записей баз данных ODBC;
- класс CDaoRecordView, предназначенный для работы, просмотра записей баз данных DAO;
- класс COleDBRecordView, предназначенный для работы, просмотра записей баз данных, использующих технологию OLE.

Отдельно можно отметить класс CScrollView, являющийся базовым для CFormView и ориентированный на работу с документами, требующими горизонтальной и вертикальной "прокрутки" документов, обеспечивающий соответствующую функциональность ЭУ полоса прокрутки.

Класс CCtrlView (рисунок 7) специально адаптирован к архитектуре документ-вид. Его назначение во многом напоминает назначение класса CControlBar, обеспечивающего функциональность панелей элементов управления. Служит основой для установки общих параметров вида документа и включает следующий набор атрибутов и методов:

- атрибут CCtrlView::m_strClass хранит имя класса окна Windows для класса вида;
- атрибут CCtrlView::m_dwDefaultStyle хранит стиль, устанавливаемый по умолчанию для класса вида и применяемый при создании соответствующего окна;
- конструктор CCtrlView::CCtrlView(LPCTSTR lpszClass, DWORD dwStyle) вызывается при создании нового фрейма или при разбиении окна на области.

Кроме этого, в классе переопределены функции OnDraw и PreCreateWindow, а также реализован обработчик OnPaint сообщения WM_PAINT.

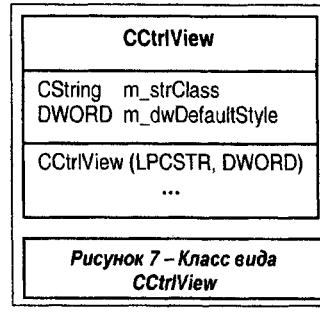

Рисунок 7 – Класс вида CCtrlView

Класс CEditView (рисунок 8) может применяться для реализации простейшего текстового редактора. Широко использовался до появления 32-разрядных версий ОС Windows, достаточно полно адаптирован для различных режимов работы с текстами.

Класс инкапсулирует функциональные возможности элемента управления Windows текстовое поле (EDIT) и обеспечивает работу с видом документа в стиле, напоминающем работу с этим элементом управления. А так как класс наследует функциональность CView, то его объекты могут использоваться с документами и шаблонами документов, как и другие объекты вида. Соответственно для текста, содержащегося в данном объекте элемента управления, выделяется собственная область глобальной памяти.

Кроме этого, функциональность класса обеспечивает:

- поддержку печати документов, предварительный просмотр документов (обработку стандартных команд типа ID_FILE_PRINT);
- управление табуляцией;

- поддержку функций поиска и замены фрагментов текста (обработку стандартных команд типа ID_EDIT_FIND, ID_EDIT_REPLACE, ID_EDIT_REPEAT), управляемых пользователем посредством специализированного диалогового окна Найти и Заменить (при этом разработчик может дописать обработчики команд кнопок Найти, Заменить и Заменить все).

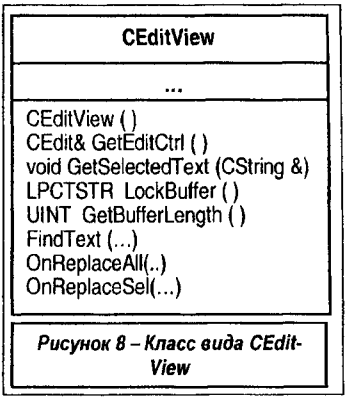

Рисунок 8 – Класс вида CEditView

Недостатки объекта вида и соответствующего редактора, созданного на базе указанного класса, что вытекает из ограниченных возможностей элемента управления EDIT, показаны ниже:

- не поддерживается технология WYSIWYG;
- практически отсутствуют средства форматирования, для отображения текста используется только один шрифт;
- ограничен размер текста, который может содержать объект.

Ниже представлены базовые методы класса:

- метод CEdit& CEditView::GetEditCtrl( ) возвращает ссылку на объект - элемент управления CEdit, связанный с объектом вида, что позволяет использовать его методы непосредственно из объекта вида;
- метод void GetSelectedText(CString& strResult) копирует выделенный текст или символы, предшествующие первому символу "возврат каретки", в объект strResult класса CString;
- метод LPCTSTR LockBuffer( ) блокирует буферную память элемента управления, запрещая модификацию содержимого, и возвращает ее указатель;
- метод void CEditView::UnlockBuffer( ) разрешает модификацию буферной памяти, заблокированной ранее методом LockBuffer;
- метод UINT GetBufferLength( ) возвращает текущее число символов буферной памяти (без учета нуль-символа);
- метод void CEditView::SerializeRaw (CArchive &ar) является внутренней реализацией функции Serialize для класса CEditView и обеспечивает сериализацию содержимого объекта CEditView в текстовый файл (параметр ar). При этом сериализация текста не требует предварительного описания данных объекта, т.к. данные хранятся не в объекте документа, а в объекте вида класса CEditView;
- метод BOOL CEditView::FindText (LPCTSTR lpszFind, BOOL bNext = TRUE, BOOL bCase = TRUE) обеспечивает поиск текста lpszFind в буферной памяти элемента управления, начиная с текущей позиции и в направлении bNex (при значении TRUE - от нача-

ла к концу буфера) с учетом значения bCase (при значении TRUE учитывается регистр). Как правило, приводит к вызову переопределенной функции OnFindText. Возвращаемый результат FALSE, если текст не найден;

- метод virtual void CEditView::OnFindText (LPCTSTR lpszFind, BOOL bNext = TRUE, BOOL bCase = TRUE) аналогичен методу FindText, однако вызывается нажатием кнопки Следующий диалогового окна Поиск, визуализируемом при выборе команды ID_EDIT_FIND. Если текст не найден, то вызывается метод OnTextNotFound;

- метод virtual void CEditView::OnReplaceAll (LPCTSTR lpszFind, LPCTSTR lpszReplace, BOOL bCase) инициируется нажатием кнопки "Заменить все" диалогового окна "Замена" и обеспечивает поиск в буферной памяти с учетом чувствительности к регистру bCase текста lpszFind, начиная с текущей позиции, с последующей заменой его на текст lpszReplace;

- метод virtual void CEditView::OnReplaceSel (LPCTSTR lpszFind,BOOL Next, BOOL bCase, LPCTSTR lpszReplace) инициируется нажатием кнопки "Заменить" диалогового окна "Замена" и после замены выделенного текста ищет следующий фрагмент, совпадающий с lpszFind, в направлении bNext. Сам поиск реализуется методом FindText.

Класс CRichEditView (рисунок 9) обеспечивает создание приложений на базе архитектуры документ-вид (ТКП ДВ), где окно вида документа базируется на элементе управления - расширенное текстовое поле (rich edit control). Содержит методы для редактирования и форматирования текста и может быть использован для реализации текстовых редакторов. Класс используется совместно с классами CRichEditCtrl, CRichEditDoc и CRichEditCntrItem.

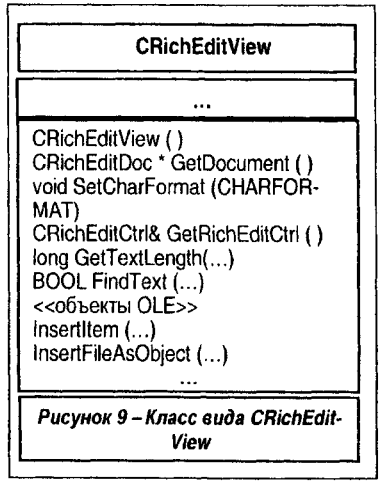

Рисунок 9 – Класс вида CRichEdit-View

Использование расширенного текстового поля, как и других общих ЭУ (Windows Common control), а следовательно и объектов класса CRichEditCtrl и связанных с ним классов, возможно только при работе под управлением операционной системы Windows 95 или Windows NT версии 3.51 или более поздних версий. Описание данного класса содержится в файле заголовка afxrich.h (#include <afxrich.h> и #include <afxcmn.h>).

Расширенное текстовое поле представляет собой окно, в котором пользователь может вводить и редактировать текст. Текст (символы, абзацы и т.д.) могут форматироваться, в том числе с использованием различных шрифтов. В текст документа могут включаться, внедряться различные объекты OLE.

Соответственно класс CRichEditDoc позволяет в приложении использовать и работать со списками объектов, клиентов OLE. При этом класс CRichEditCntrItem обеспечивает доступ к клиентам OLE со стороны контейнера.

Указанный ЭУ обеспечивает программный интерфейс для работы с текстом, форматирования текста, а разработчик должен использовать подходящие элементы пользовательского интерфейса для организации доступа к ним.

Базовые методы класса CRichEditView:

- метод CRichEditDoc* GetDocument( ) возвращает указатель на объект-документ класса CRichEditDoc, связанный с объектом вида и обеспечивает доступ к его содержимому;
- метод void SetCharFormat(CHARFORMAT cf) обеспечивает задание атрибутов форматирования символов текста cf (при этом заменяются только атрибуты, специфицированные маской dwMask структуры CHARFORMAT);

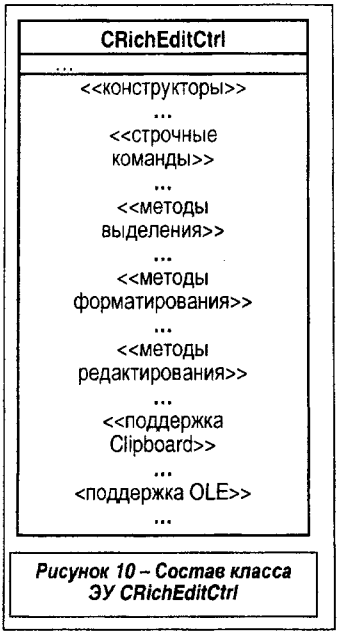

Рисунок 10 – Состав класса ЭУ CRichEditCtrl

- метод CRichEditCtrl& GetRichEditCtrl( ) возвращает ссылку на объект ЭУ класса CRichEditCtrl, связанный с объектом вида и обеспечивает доступ к его методам для работы с текстом;
- метод CRichEditCtrlItem* GetSelectedItem( ) возвращает указатель на OLE объект класса CRichEditCtrlItem, выделенный в объекте вида CRichEditView или NULL, если выбранных объектов нет;
- метод long GetTextLength( ) возвращает длину текста объекта класса CRichEditView.

Базовые методы класса CRichEditCtrl (рисунок 10) включают следующие группы: - методы работы с текстом (выделение, форматирование, редактирование); - методы работы с внедренными объектами OLE; - методы поддержки типовых действий обменного буфера и действий по копированию, вставке выделенных фрагментов текста.

При этом, если указанный ЭУ используется в диалоговом окне, а не совместно с объектом класса CRichEditView, то необходимо в методе InitInstance приложения выполнять его однократную инициализацию методом AfxInitRichEdit до визуализации диалогового окна.

Класс CTreeView (рисунок 11) предназначен для разработки приложений на базе архитектуры документ-вид (ТКП ДВ), где реализована поддержка средств управления видом документа на базе иерархических описаний, списков, отображаемых в виде соответствующего дерева. Соответственно окно вида строится на базе древовидного ЭУ (tree view control), чья функциональность (управление структурным видом) обеспечивается классом CTreeCtrl.

Использование древовидного ЭУ, как и других общих ЭУ (Windows Common control), а следовательно и объектов класса CTreeCtrl, возможно только при работе под управлением операционной системы Windows 95 или Windows NT версии 3.51 или более поздних версий. Описание данного класса содержится в файле заголовка afxcmn.h (#include <afxcmn.h>).

Указанный ЭУ является окном, которое выводит на экран иерархический список элементов, таких как заголовки в документе, списки файлов, каталогов диска и т.п. Каждый элемент состоит из метки и сопутствующей пиктограммы - растрового изображения элемента. У каждого элемента может быть список подэлементов, связанных с ним. Щелкая по элементу (или используя специальную кнопку), пользователь может развернуть или свернуть связанный с текущим элементом список подэлементов.

Подключение класса CTreeView реализуется как #include <afxcview.h>. Работа с видом уточняется набором параметров стилей, некоторые из них представлены ниже:

- стиль TVS_HASLINES обеспечивает отображение связей родительских элементов дерева с подчиненными элементами;

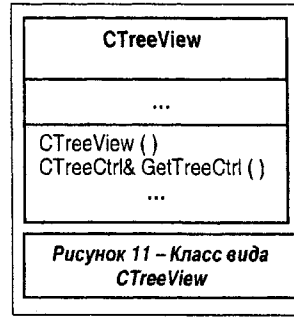

Рисунок 11 – Класс вида CTreeView

- стиль TVS_LINESATROOT обеспечивает добавление связей к корневому дереву;

- стиль TVS_HASBUTTONS обеспечивает добавление кнопок раскрытия подчиненных элементов дерева. Кнопки располагаются справа от каждого элемента дерева;

- стиль TVS_EDITLABELS позволяет организовать редактирование элемента дерева "по месту";

- стиль TVS_SHOWSELALWAYS разрешает элементу дерева оставаться выбранным, когда само дерево теряет фокус.

Класс CListView (рисунок 12) предназначен для разработки приложений на базе архитектуры документ-вид (ТКП ДВ), где реализована поддержка средств управления видом документа на базе списка (списка пиктограмм), отображаемого различными способами (четыре взаимоисключающих стиля) в зависимости от задаваемого динамически режима отображения.

Соответственно окно вида строится на базе спискового ЭУ ("list view control"), чья функциональность (управление структурным видом) обеспечивается классом CListCtrl. Этот элемент управления, в частности, использован в Windows Explorer для отображения списка файлов.

Для получения доступа к методам управления списком в классе реализован метод CListCtrl& CListView::GetListCtrl( ), возвращающий ссылку на объект списка, связанный с текущим объектом вида документа

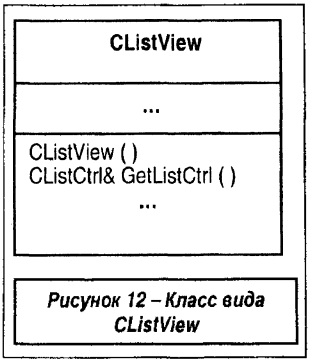

Рисунок 12 – Класс вида CListView

Списковый ЭУ представляет собой окно класса "SysListView32", предназначенное для отображения набора, списка элементов, состоящих из пиктограммы, текстового описания (метки), при необходимости – из дополнительной информации, соответствующей каждому элементу списка и отображемой, например, в колонках, расположенных справа от пиктограммы.

Стиль ЭУ формируется как комбинация стилей WS_CHILD, WS_VISIBLE, а также стилей, специфичных для данного элемента управления:

- стиль LVS_ICON обеспечивает вывод элементов списка в виде пиктограмм (размером 32 x 32 пикселя) и сопроводительного текста под пиктограммами;
- стиль LVS_SMALLICON обеспечивает вывод элементов списка в виде пиктограмм (размером 16 x 16 пикселей) и сопроводительного текста справа от пиктограмм;
- стиль LVS_LIST (аналогично LVS_SMALLICON) обеспечивает вывод элементов списка в упорядоченном виде;
- стиль LVS_REPORT обеспечивает вывод элементов списка построчно. Каждый строка может содержать подэлементы, отображаемые в отдельных колонках, каждая из которых, как правило, имеет заголовок. Пользователь может изменять с помощью мыши ширину колонок и перемещать их друг относительно друга.

Дополнительные стили:

- стиль LVS_AUTOARRANGE для упорядочивания элементов списка в режимах отображения ICON и SMALLICON;
- стиль LVS_ALIGNTOP для выравнивания элементов списка по его верхней границе (слева направо, сверху вниз);

- стиль LVS_ALIGNLEFT для выравнивания элементов списка по его левой границе (сверху вниз, слева направо);

- стиль LVS_SORTASCENDING для сортировки элементов списка по возрастанию и др.

В начале работы приложения в методе BOOL Create(DWORD fdwStyle, const RECT& rect, CWnd* pParentWnd, UINT uID), а затем в процессе работы можно программно изменять стиль списка методом BOOL ModifyStyle(DWORD fdwRemove, DWORD fdwAdd, UINT fuFlags = 0), меняя тем самым визуальное представление элементов списка (стиль задается параметром fdwStyle).

Помимо средств задания исходного списка класс содержит и методы динамического изменения состава списка. Это – метод вставки InsertItem нового элемента в список, метод удаления элемента или всех элементов списка DeleteItem, DeleteAllItems, методы SortItems сортировки и др.

Класс CScrollView (рисунок 14), наследуя все возможности класса CView, также используется для разработки приложений на базе архитектуры документ-вид (ТКП ДВ). Здесь, в отличие от средств базового класса CView, где разработчик должен реализовать обработку сообщений от полос прокрутки, используя для этого функции CWnd::OnHScroll и CWnd::OnVScroll, осуществлена поддержка автоматической прокрутки и масштабирования документов, т.е.:

- обрабатывается информация о размерах окна и его рабочей области, а также о режимах отображения документа;

- прокручивается изображение документа в ответ на сообщения от полос прокрутки;

- прокручивается изображение документа в ответ на сообщения от клавиатуры, "классической" мыши и от колесика мыши типа IntelliMouse, что реализуется методами OnMouseWheel и OnRegisteredMouseWheel.

Полосы прокрутки бывают стандартные, имеющие стиль окна Windows с параметрами WS_HSCROLL и-или WS_VSCROLL, и полосы прокрутки, добавляемые в главное окно приложения, содержащего объект класса вид. В последнем случае главное окно приложения само посылает сообщения WM_HSCROLL и WM_VSCROLL объекту вида. В перечисленных случаях полосы отображаются.

По умолчанию "прокручиваемая" страница составляет 1/10 размера области вида, а строка - 1/10 от размера страницы, при щелчке левой кнопкой мыши в полосе прокрутки за пределами бегунка документ передвигается на страницу, при щелчке по стрелкам прокрутки документ передвигается на строку. Чтобы изменить параметры прокрутки применяется метод SetScrollSizes.

Для работы объект вида создается конструктором CScrollView::CScrollView( ), после чего до использования объекта необходимо вызвать метод SetScrollSizes или метод SetScaleToFitSize.

Метод void CScrollView::SetScrollSizes (int nMapMode, SIZE sizeTotal, const SIZE sizePage = sizeDefault, const SIZE sizeLine = sizeDefault) вызывается при обновлении окна вида (обычно в методе CView::OnInitialUpdate или CView:: OnUpdate) для настройки параметров прокрутки и позволяет установить режим отображения документа nMapMode, общий размер прокручиваемой области и шаг вертикальной и горизонтальной прокрутки. Все размеры задаются в логических единицах. Соответственно используются параметры:

- общий размер sizeTotal прокручиваемой области вида;

- шаг sizePage перемещения изображения по горизонтали или по вертикали при щелчке левой кнопкой мыши в полосе прокрутки за пределами бегунка;

- шаг sizeLine перемещения изображения по горизонтали и по вертикали при щелчке левой кнопкой мыши по стрелкам полосы прокрутки.

В альтернативном варианте можно совсем исключить полосы прокрутки с помощью метода CScrollView::SetScaleToFitSize и применить автоматическое масштабирование области вида, чтобы ее размеры совпадали с размерами рабочей области окна.

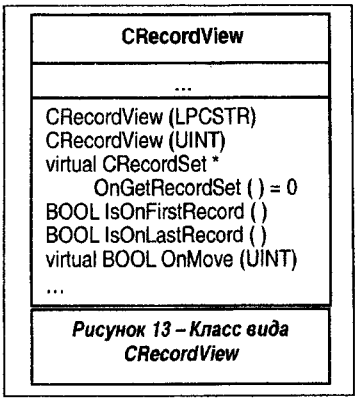

Рисунок 13 – Класс вида CRecordView

Класс CFormView - базовый класс, поддерживающий функциональность окон просмотра документов в виде форм, создаваемых на основе шаблона диалогового окна, диалоговой панели. Как результат, могут быть созданы приложения, основанные на формах.

Соответственно такой вид, форма включает в качестве средства управления типовые элементы управления, обеспечивающие интерактивное взаимодействие пользователя и приложения. Также поддерживается "прокрутка" документа на базе функциональности класса CScrollView.

В MFC на основе класса CFormView порождены другие классы, специализирующие его функциональность для работы с документами, документами баз данных для конкретных технологий доступа.

Класс CRecordView (рисунок 13) является производным от класса CFormView и предназначен для обеспечения функциональности видов, ориентированных на отображение и работу с записями баз данных. Непосредственно использует объект класса CRecordset.

Здесь поля записи (объекта класса CRecordset) отображаются с помощью интерфейсных форм в элементах управления. При работе формы для автоматизации передачи информации между ЭУ формы и полями записи объект вида CRecordView использует механизм диалогового обмена данными (DDX) и механизм обмена полями записи (RFX).

Кроме этого, обеспечена возможность автоматического перехода к первой, последней, следующей, предыдущей записям, а также редактирование текущей, отображаемой записи.

Мастер AppWizard позволяет сгенерировать шаблон приложения, включающий объект вида и связанный с ним объект класса запись. Для управления формой AppWizard создает меню и панель инструментов. Подключение выполняется как #include <afxdb.h>.

Класс CHtmlView является производным от класса CFormView и обеспечивает функциональность приложений архитектуры документ-вид, специализированных на работе с web-документами (и-или HTML-документами). Соответственно на базе этого класса могут быть созданы приложения-браузеры.

Здесь представление вида документа основывается на использовании элемента управления WebBrowser. Этот ЭУ по аналогии с полноценными браузерами предоставляет пользователю окно для просмотра страниц сайтов, данных локальной или сетевой

файловой системы, обеспечивает навигацию с использованием гиперссылок, указателя ресурсов (URL), сохраняет предысторию переходов. Подключение класса выполняется как #include <afxhtml.h>.

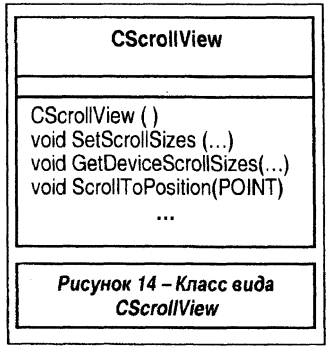

Рисунок 14 – Класс вида CScrollView

2. АВТОКАРКАС

2.1. Особенности. Создание. Пользовательский интерфейс

Каркас, созданный мастером, может отличаться от аналогичного каркаса, созданного вручную с минимальным набором необходимых средств:

- составом интерфейса, куда входит панель инструментов (Toolbar), дублирующая главное меню, строка подсказки, само меню расширено (содержит дополнительно пункт Edit для обеспечения работы с буфером, пункт View для управления видом окна – включения-отключения компонентов ToolBar, StatusBar, пункт Help для вызова диалогового окна со справочной информацией), диалоговое окно со справочной информацией о приложении;
- составом ресурсов (Toolbar, Version, набор пиктограмм приложения Icon, Dialog, Accelerator);
- составом классов (class CAboutDlg : public CDialog для поддержки функциональности справочного окна);
- сами классы, непосредственно образующие каркас, включают ряд дополнительных методов, обеспечивающих, в частности, отладочный режим приложения, поддержку дополнительных пунктов меню и т.д.;
- проект приложения имеет классическую многофайловую компоновку (по два файла с расширениями *.h и *.cpp на каждый используемый класс).

Каркас приложения (в данном случае на базе технологии документ-вид) создается мастером в виде последовательности шагов, уточняющих вид и функциональность каркаса будущего приложения.

Для создания простейшего каркаса (ТКП ДВ) без поддержки баз данных и составных документов с клиентской областью на базе класса CView следует выполнить следующие действия:

1. Запустить среду программирования. Выбрать пункт меню File-New, а в появившемся окне New вкладку Projects. Среди списка мастеров выбрать – MFC AppWizard(exe), в поле project name задать имя проекта (здесь DV), в поле Location определить место для хранения папки проекта (рисунок 15).

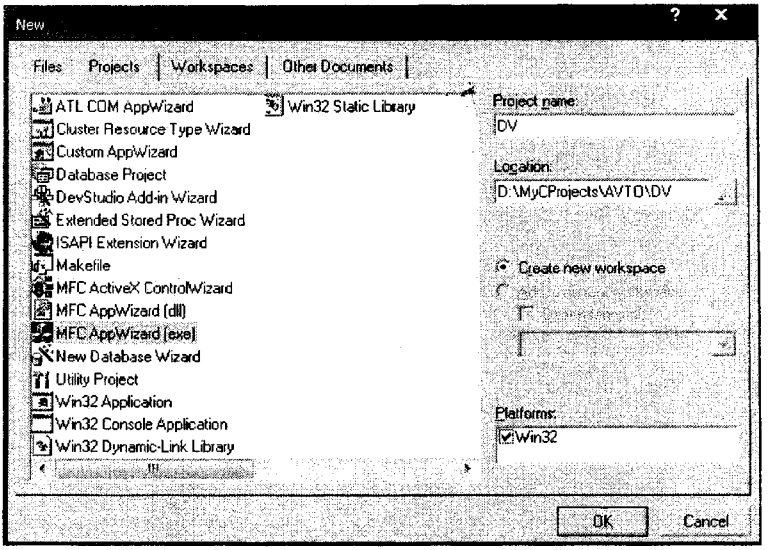

Рисунок 15 – Окно выбора мастера

2. На вкладке Шаг 1 выбрать опцию – одно документное приложение (Single document) с поддержкой технологии документ-вид (Document-View architecture support).

3. На вкладке Шаг 2 выбрать опцию – без поддержки баз данных (Data base support – none).

4. На вкладке Шаг 3 выбрать опцию – без поддержки составных документов (Compound document support – none).

5. На вкладках Шаг 4, 5 определить необходимые сервисные опции (рисунок 16). Расширенный выбор опций находится на вкладке Advanced... – там, в частности, можно выбрать расширение для автоматической загрузки документов, доопределить дизайн окна и т.д.

6. На вкладке Шаг 6 можно по своему желанию переименовать пользовательские классы, использованные в каркасе, выбрать базовый класс для пользовательского класса вида из выпадающего списка имен классов (Base class) и выполнить другие настройки (рисунок 17).

Мастер построения каркасов поддерживает для создания вида все перечисленные в разделе 1 классы вида. Причем классы CRecordView, CDaoRecordView, COleDBRecordView (рисунок 5) подключаются в автокаркасе, если была выбрана опция поддержки работы с БД с источниками данных соответственно ODBC, DAO, OLE DB типов.

Соответственно при запуске приложения (каркаса) отображается главное окно с меню (пункты File, Edit, View, Help) (рисунок 18).

Меню Edit позволяет управлять панелью инструментов и строкой состояния.

Меню Help вызывает диалоговое окно со справочной информацией о приложении.

Пункты меню Edit для работы с буферной памятью не активны и требуют дополнительной поддержки приложением.

Меню File позволяет инициировать группы команд сохранения-загрузки документа и организации печати документа. Однако их корректная работа требует дальнейшей настройки приложения.

В частности, сохранение-загрузка документа реализуется на базе механизма сериализации и требует дописания соответствующего метода класса документа void Serialize(CArchive& ar).

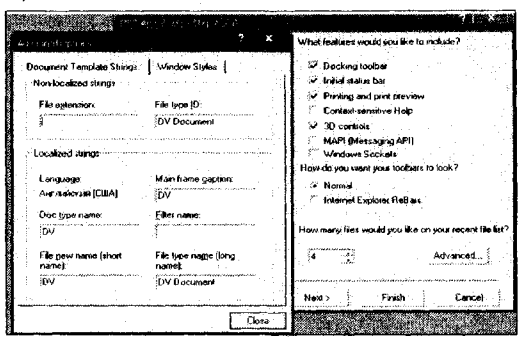

Рисунок 16 – Окно мастера (шаг 4)

2.2. Состав каркаса

Типовой состав каркаса, получаемого автоматически, иллюстрируется рисунком 19. Каркас включает:

1) программу на языке C++, описывающую функциональность каркаса в виде набора классов. Описания, декомпозированные на отдельные файлы, образуют программную составляющую проекта;

2) "графические" ресурсы, образующие основу пользовательского интерфейса. Их описания собраны в специальном файле и хранятся отдельно от программной составляющей проекта. При этом главное меню, пиктограмма и панель инструментов приложения имеют общий идентификатор (здесь IDD_MAINFRAME);

3) документы, для обработки которых создается приложение (каркас). Документы хранятся во внешней памяти и, при необходимости, загружаются в память с использованием механизма сериализации.

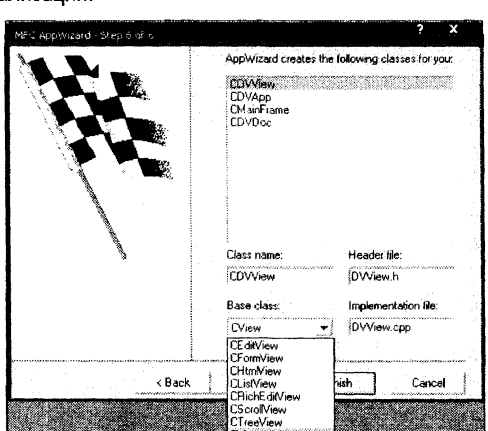

Рисунок 17 – Окно мастера (шаг 6)

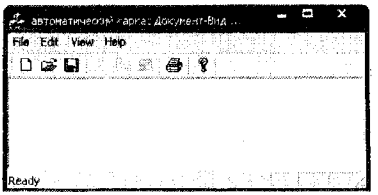

Рисунок 18 – Интерфейс ТКП ДВ

2.3. Настройка каркаса

Настройка каркаса производится специальной строкой, которая хранится в ресурсе - таблице строк. Формат строки описания настроек представлен на рисунке 20.

Строка настройки каркаса с идентификатором IDR_MAINFRAME изначально выглядит как

IDR_MAINFRAME DV\n\nDV\n\n\nDV.Document\nDV Document .

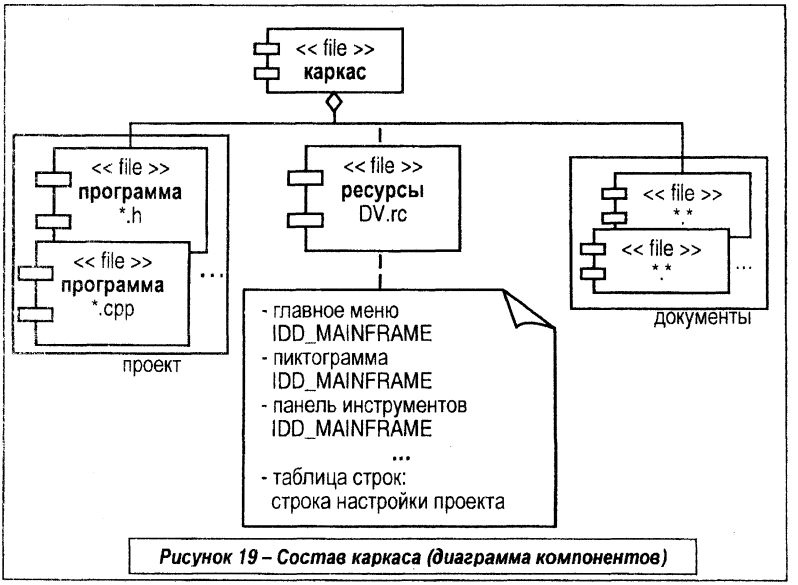

Рисунок 19 – Состав каркаса (диаграмма компонентов)

В строке:

- ЗО заголовок окна (здесь DV);
- НД название документа по умолчанию (т.к. поле не задано, то в качестве первичного имени документа системой будет предлагаться "Безымянный");
- ТД тип документа (расширение), если приложение предусматривает работу с несколькими типами (здесь DV);
- ФФ описание фильтра файлов, используемого в типовых окнах загрузки-сохранения документов (список "тип файлов");
- РД расширение файлов для документов данного типа;
- ИТСР идентификатор типа файлов для базы системной регистрации;
- ИмТСР имя типа файлов для базы системной регистрации.

Указанные поля приводятся в одной строке в том же порядке. В качестве разделителя полей используется символ "новая строка" ("\n"). Соответственно этот же символ замещает отсутствующее поле строки.

ЗО	НД	ТД	ФФ	РД	ИТСР	ИмТСР

Рисунок 20 – Формат строки описания настроек

Указанный ресурс-строка описывается в файле ресурсов оператором STRINGTABLE как

```
STRINGTABLE
{
    ID_Ресурсов_ШД "строка"
}
```

и может быть задан средствами редактора ресурсов в таблице строк (рисунок ...). Здесь ID_Ресурсов_ШД - идентификатор строки (как правило, совпадающий с идентификатором ресурсов шаблона документа – меню и т.д.).

При настройке расширения документа как qwu (само имя формируется как Безымянный.qwu) строка настройки выглядит как

IDR_MAINFRAME DV\n\nDV\n\n.qwu\nDV.Document\nDV Document .

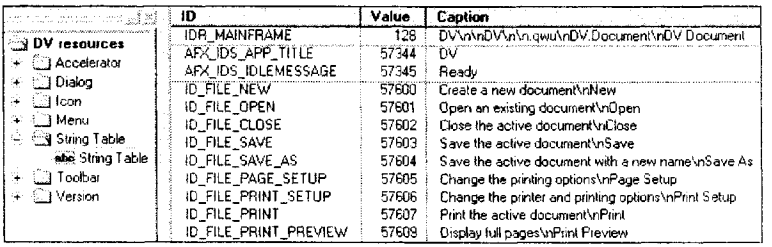

Рисунок 21 – Ресурс (String Table) таблица строк

2.4. Состав файлов и классов

Описания, декомпозированные на отдельные файлы (рисунок 22), образуют программную составляющую проекта. Она представлена на рисунке 23 в виде диаграммы программных компонентов.

Диаграмма классов каркаса документ-вид представлена на рисунке 24. Здесь жирным шрифтом выделены пользовательские классы и их базовые классы из библиотеки MFC.

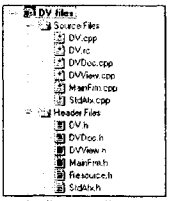

Рисунок 22 – Файловый состав проекта

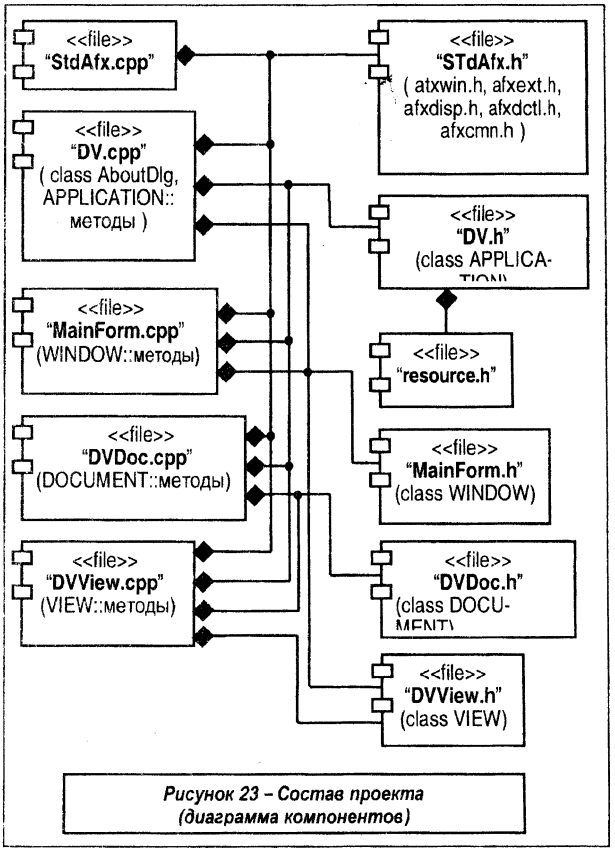

Рисунок 23 – Состав проекта (диаграмма компонентов)

Каркас, шаблон документа строится на базе классов CMultiDocTemlate или CSingleDocTemlate соответственно при генерации много-или однодокументного каркаса приложения.

Состав пользовательских классов представлен на рисунке 25.

2.5. Редактирование каркаса

При дальнейшем использовании каркаса для разработки собственного приложения необходимо:

1. Убрать лишние элементы интерфейса, лишние компоненты каркаса, например,

- в редакторе ресурсов удалить не используемые пункты и подпункты главного меню, удалить не используемые кнопки панели инструментов;
- в обозревателе классов (вкладка ClassView), используя контекстное меню, удалить соответствующие обработчики сообщений (например, для удаленных пунктов меню), а затем скорректировать и карты сообщений соответствующих классов, убрав чувствительность к одноименным сообщениям.

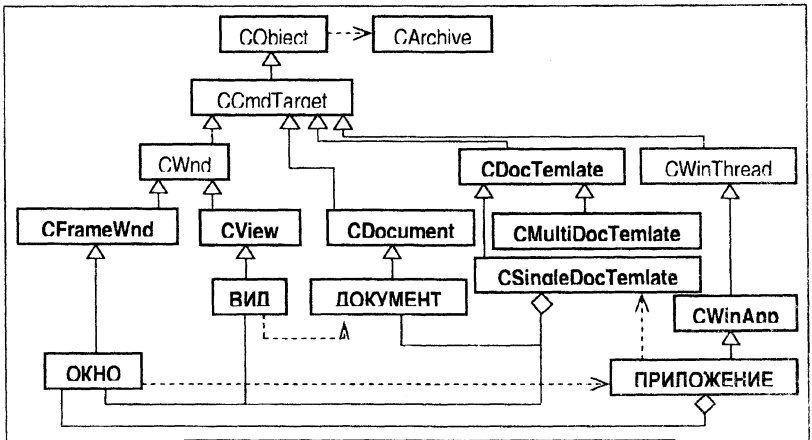

Рисунок 24 – Диаграмма классов ТКП ДВ

2. Внести пользовательские правки. Так, при необходимости, можно:

- в редакторе ресурсов добавить пользовательские пункты и подпункты в главное меню, добавить соответствующие обработчики, отредактировать карты сообщений классов;
- в редакторе ресурсов скорректировать вид и информацию, выводимую в справочном окне приложения;
- в редакторе ресурсов (Icon) скорректировать вид пиктограмм, используемых при работе приложения;
- изменить название главного окна (в редакторе ресурсов изменить в String Table строку IDR_MAINFRAME) в соответствии с рисунком 26.

3. Внести пользовательские правки в классы каркаса.

Ниже представлены диаграммы классов, где показаны базовые методы классов, а жирным шрифтом выделены методы, требующие, как правило, пользовательского редактирования.

Кроме этого, при необходимости, могут быть уточнены пользователем методы инициализации окон PreCreateWindow, OnCreate, управление печатью OnBeginPrinting, OnEndPrinting и др.

Соответствующий листинг приведен в ПРИЛОЖЕНИИ.

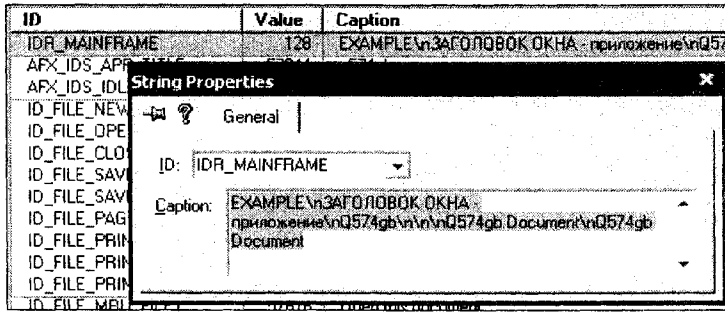

Рисунок 26 – Изменение строки настройки

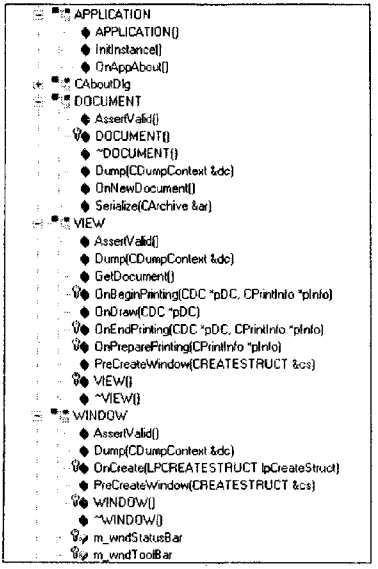

Рисунок 25 – Состав классов ТКП ДВ

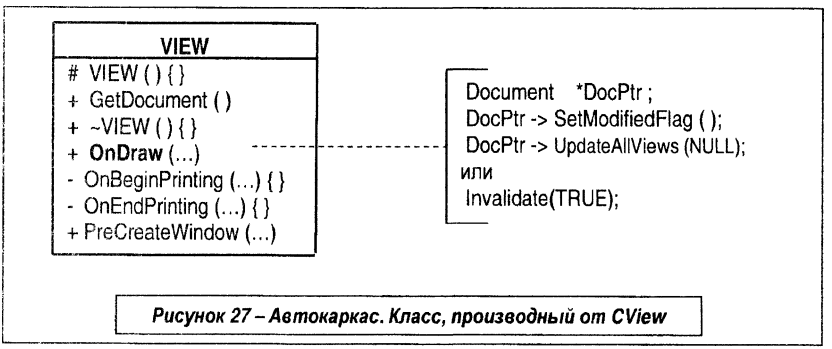

Рисунок 27 – Автокаркас. Класс, производный от CView

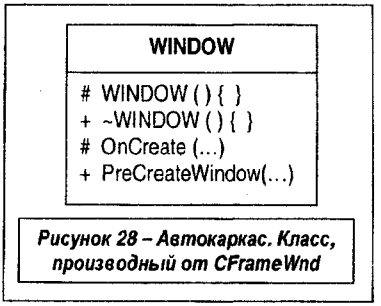

Рисунок 28 – Автокаркас. Класс, производный от CFrameWnd

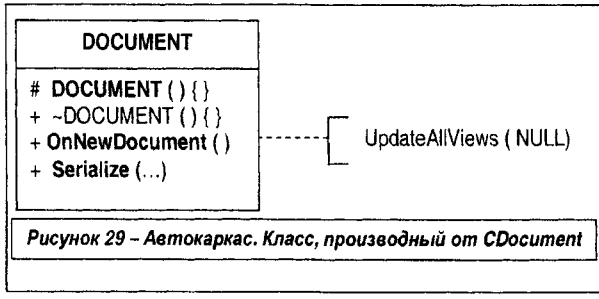

Рисунок 29 – Автокаркас. Класс, производный от CDocument

3. ИСПОЛЬЗОВАНИЕ КАРКАСА В ПРИЛОЖЕНИЯХ

3.1. Приложение для обработки строк

ПРИМЕР 1 *использования автоматического каркаса для создания основы приложения для работы с документом - строкой. Приложение должно поддерживать следующий набор действий:*

- создание нового документа с пустой строкой;

- ввод строки путем последовательного набора символов с клавиатуры;

- отображение как загруженного документа, так и редактируемого (вводимого) в клиентской области окна;

- файловые операции с документом (подготовка нового документа, сохранение, загрузка документа).

Примерный вид главного окна со строкой, отображенной в клиентской области окна, и справочным окном представлен ниже (рисунки 31, 32).

Строку – документ будем представлять пользовательским классом aSTR. Соответственно диаграмма классов приложения на рисунке 33 включает пользовательский класс aSTR, композируемый классом документа DOC.

Таким образом, необходимо:

1. Создать, используя мастер, автоматический однодокументный каркас (ТКП ДВ), переименовать классы и файлы по своему усмотрению (здесь в соответствии с диаграммами классов), отменить поддержку печати документов.

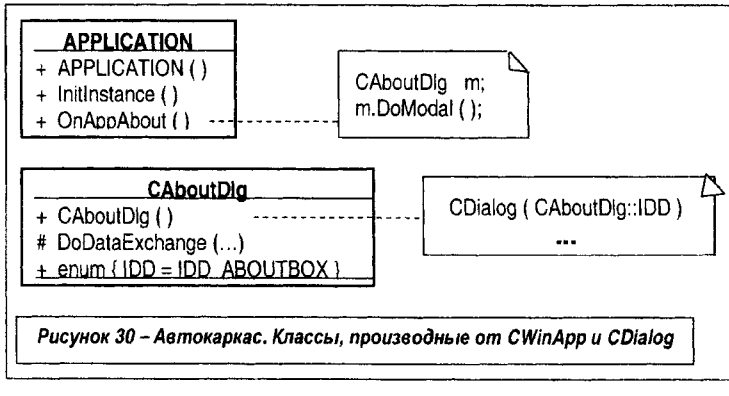

Рисунок 30 – Автокаркас. Классы, производные от CWinApp и CDialog

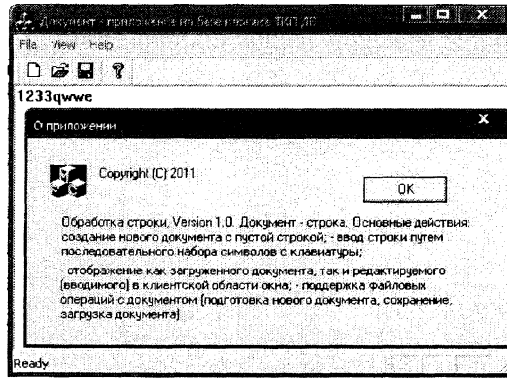

Рисунок 31 – Интерфейс ТКП ДВ

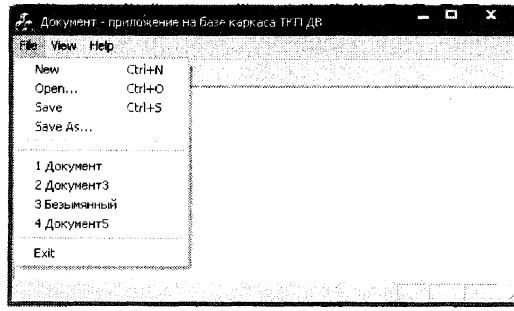

Рисунок 32 – Интерфейс ТКП ДВ

2. Скорректировать состав и вид ресурсов каркаса. Например, изменить главное меню, убрав добавив пункты, установить текст справочного сообщения о приложении, а также скорректировать строку настройки IDR_MAINFRAME, представив ее как

приложение на базе ТКП ДВ\n Документ\n DV\n DV Files (*.str)\n .str\n DV.Document\n DV Document .

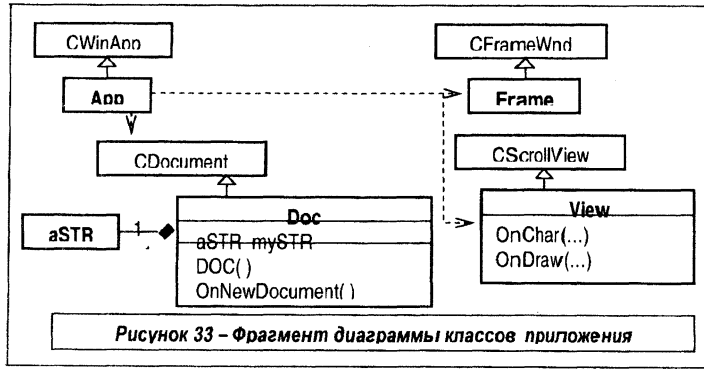

Рисунок 33 – Фрагмент диаграммы классов приложения

3. Добавить к проекту описание документа – класса aSTR

```
class aSTR
{
    CString Str;
public:
    aSTR ( );
    void TO_CLEAR ( ) ;
    void TO_ADD (CString InStr);
    CString&  STR_AS_VARIABLE ( );
};

aSTR :: aSTR ( ) { Str = ""; };
void aSTR : TO_CLEAR ( ) { Str = ""; };
void aSTR :: TO_ADD (CString InStr) {  Str  +=  InStr;  };
CString&  aSTR ::STR_AS_VARIABLE ( ) { return  Str; };  .
```

3.1. Для этого представим описание класса в виде двух файлов aSTR.h, aSTR.cpp как

```
//aSTR.h :  interface of the aSTR class
class aSTR
{
    CString Str;
public:
    aSTR ( );
    void TO_CLEAR ( ) ;
    void TO_ADD (CString InStr);
    CString&  STR_AS_VARIABLE ( );
};

//aSTR.cpp : implementation of the aSTR class
#include "stdafx.h"
#include "DV.h"
#include "aSTR.h"
aSTR ::  aSTR ( ) { Str = ""; };
void aSTR ::  TO_CLEAR ( ) { Str = ""; };
void aSTR ::  TO_ADD (CString InStr) {  Str  +=  InStr;  };
CString&  aSTR ::  STR_AS_VARIABLE ( ) { return  Str; };  .
```

3.2. При необходимости скопировать файлы aSTR.h, aSTR.cpp в папку создаваемого проекта.

3.3. Находясь в среде, добавить файлы в состав проекта. Для этого использовать обозреватель классов (вкладка ClassView), контекстное меню и последовательно добавить файлы, как показано на рисунках 34, 35, командой Add Files to Folder в папки Source Files, Header Files.

4. Для обеспечения видимости в файл описания класса документа добавить директиву

```
#include "aSTR.h"   ,
```

в само описание класса Doc добавить член-данное **aSTR myStr.** Для этого использовать обозреватель классов (вкладка ClassView), контекстное меню и описать данное, как показано на рисунках 36, 37.

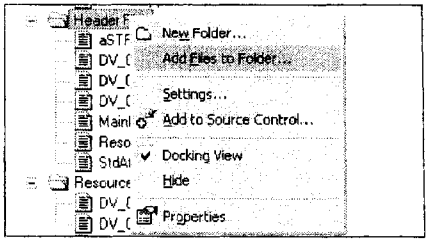

Рисунок 34 – Вставка файлов в проект

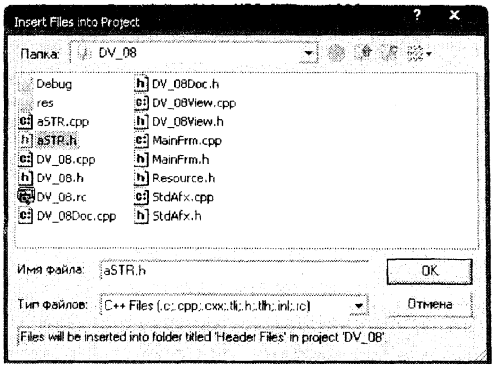

Рисунок 35 – Вставка файлов в проект

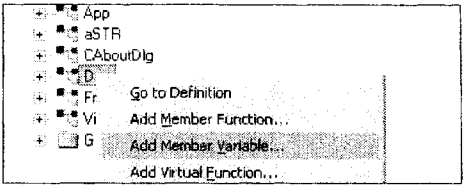

Рисунок 36 – Добавление переменной к классу

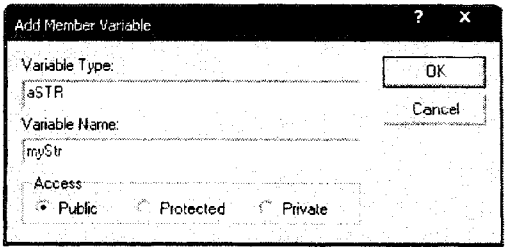

Рисунок 37 – Окно добавления переменной к классу

```
class Doc : public CDocument
{
....
public:
    aSTR myStr;
.....
};
```

В конструктор Doc( ) добавить подготовку пустого документа

```
Doc::Doc( )
{
    myStr.TO_CLEAR( );
}
```

В методе OnNewDocument( ) обеспечить подготовку пустого документа и обновление области вида

```
BOOL Doc::OnNewDocument( )
{
    if ( ! CDocument::OnNewDocument( ) )
        return FALSE;
    myStr.TO_CLEAR( );
    UpdateAllViews( NULL );
    return TRUE;
}
```

В методе Serialize обеспечить сохранение-загрузку документа

```
void Doc::Serialize( CArchive& ar )
{
    if ( ar.IsStoring( ) )
    {
        ar<<myStr.STR_AS_VARIABLE( );
    }
    else
    {
        ar>>myStr.STR_AS_VARIABLE( );
    }
}
```

5. Для реализации ввода символов строки с клавиатуры в класс вида View надо, используя мастер ClassWizard (вызов через Ctrl-W), вставить метод-обработчик события – WM_CHAR – нажатие клавиши клавиатуры (как показано на рисунке 38).

В окне мастера MFC ClassWizard выбрать вкладку редактирования карт сообщений (Message Maps).

На ней указать название класса – View (в окошке Class name), объект - View (в окошке Object IDs), имя сообщения - WM_CHAR (в окошке Messages).

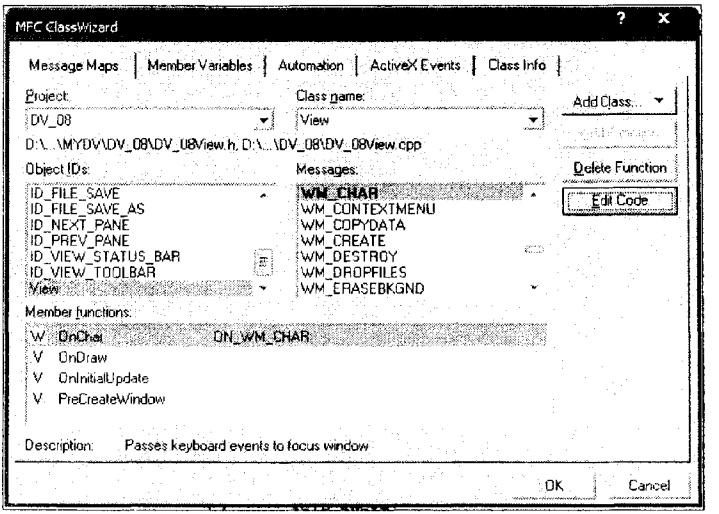

Рисунок 38 – Окно добавления обработчика

Выбрать команды Add Function и Edit Code, что позволит добавить каркас метода-обработчика OnChar (смотрите результат на вкладке в окне Member functions) и ввести его текст:

```
void View::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    Doc* pDoc = GetDocument( );
    ASSERT_VALID(pDoc);
    pDoc->myStr.TO_ADD(nChar);
    Invalidate( );
    pDoc->SetModifiedFlag( );
    CView::OnChar(nChar, nRepCnt, nFlags);
}
```

При этом автоматически вносится изменение в текст описания класса - в секцию автовставок, ограниченную скобками //{{AFX_MSG(View) ... //}}AFX_MSG (текст автовставок править вручную нежелательно!), как

```
class View : public CScrollView
{
....
// Generated message map functions
protected:
    //{{AFX_MSG(View)
    afx_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);
    //}}AFX_MSG
.....
};
```

и изменяется описание карты сообщений соответствующего класса – здесь класса вида приложения

```
BEGIN_MESSAGE_MAP(View, CScrollView)
    //{{AFX_MSG_MAP(View)
    ON_WM_CHAR()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

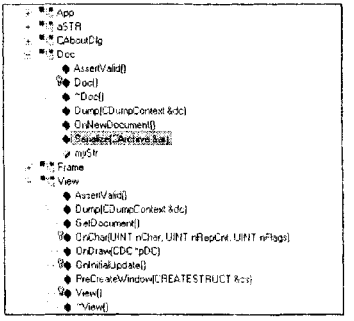

Рисунок 39 – Состав классов

6. В классе вида следует отредактировать метод для вывода строки-документа

```
void View::OnDraw(CDC* pDC)
{
    Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    pDC->TextOut( 1, 1, pDoc->myStr.STR_AS_VARIABLE( ) );
}
```

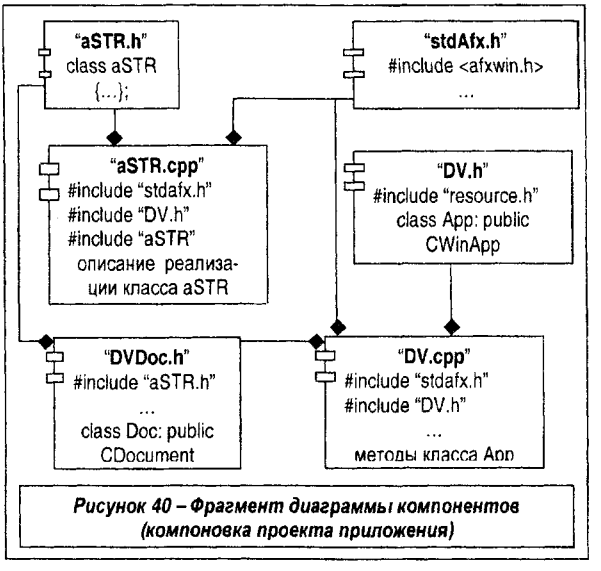

Рисунок 40 – Фрагмент диаграммы компонентов (компоновка проекта приложения)

Состав классов и методов приложения, а также диаграмма компонентов представлены на рисунках 39, 40.

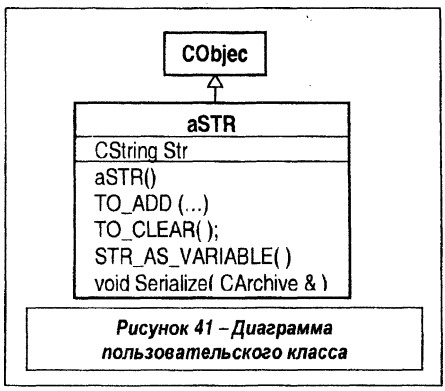

Рисунок 41 – Диаграмма пользовательского класса

3.2. Модификация приложения для обработки строк

Для примера, описанного в предыдущем параграфе, внесем следующее изменение. Пользовательский класс aSTR унаследуем от класса CObject (рисунок 41). Это позволит переописать и использовать механизм сериализации как собственный метод пользовательского класса aSTR.

Тогда

```
//aSTR.h : interface of the aSTR class
class aSTR : public CObject
{
    CString Str;
    DECLARE_SERIAL(aSTR)
public:
    aSTR ( ) { Str = ""; };
    void TO_CLEAR ( ) { Str = ""; };
    void TO_ADD (CString InStr) { Str += InStr; };
    CString& STR_AS_VARIABLE ( ) { return Str; };
    void Serialize (CArchive& Archive);
};

//aSTR.cpp : implementation of the aSTR class
#include "stdafx.h"
#include "DV.h"
#include "aSTR.h"
void aSTR :: Serialize (CArchive& Archive)
    {
        CObject:: Serialize (Archive);
        if (Archive.IsStoring( ) )   Archive << Str;
        else                         Archive >> Str;
    };
IMPLEMENT_SERIAL( aSTR, CObject, 0)
```

Для этого необходимо:

1. Создать автоматический однодокументный каркас (ТКП ДВ), при необходимости переименовать файлы, отменить поддержку печати.

2. Скорректировать ресурсы каркаса (меню, справку), а также строку состояния приложения.

3. Добавить описание класса aSTR (представленное в виде набора файлов aSTR.h, aSTR.cpp), предварительно скопировав в папку созданного приложения, к его проекту.

4. В класс документа (Doc.h) добавить директиву #include "aSTR.h", а через контекстное меню добавить в документ (класс Doc) член-данное aSTR myStr.

5. Отредактировать как в предыдущем примере методы:

- Doc::Doc( ) ;
- BOOL Doc::OnNewDocument( ) ;
- void View::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags) ;
- void View::OnDraw(CDC* pDC) .

6. Отредактировать метод сериализации документа как

```
void Doc::Serialize(CArchive& ar)
{
    myStr.Serialize(ar);
} .
```

3.3. Многодокументное приложение для обработки строк

ПРИМЕР 2. *Для примера, описанного в предыдущем параграфе, внесем следующее изменение: реализуем приложение как многодокументное, обрабатывающее один тип документов (т.е. строку символов с расширением - str) и отображающее документы в одном и том же виде (т.е. строкой в окне вида).*

Это позволит:

- работать с одним документом в одном или нескольких окнах одновременно (при этом, чтобы обеспечить идентичность всех отображений в окнах одного документа – используется метод pDoc->UpdateAllViews(this));
- работать сразу с несколькими документами (здесь одного типа) в разных окнах.

При запуске приложения окно может выглядеть, как показано на рисунке 42. При этом главное меню поддерживает возможность создания нового документа или открытие существующего.

Как только начинается работа с документом, т.е. появляется хотя бы одно окно документа, то меню изменяется – здесь появляется пункт меню Window, позволяющий как создавать дополнительные окна для отображения уже загруженного документа так и управлять их размещением (рисунок 43).

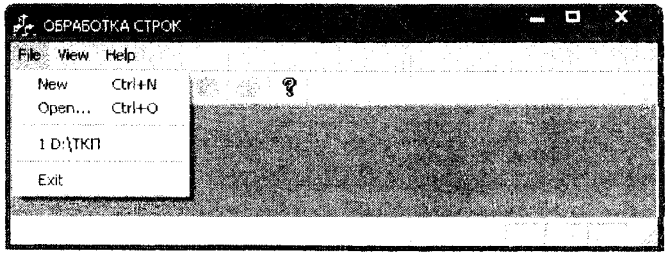

Рисунок 42 – Интерфейс приложения

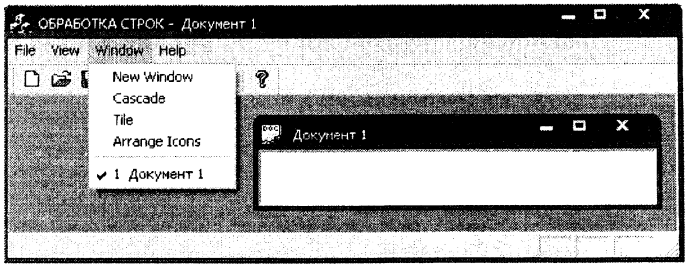

Рисунок 43 – Интерфейс приложения

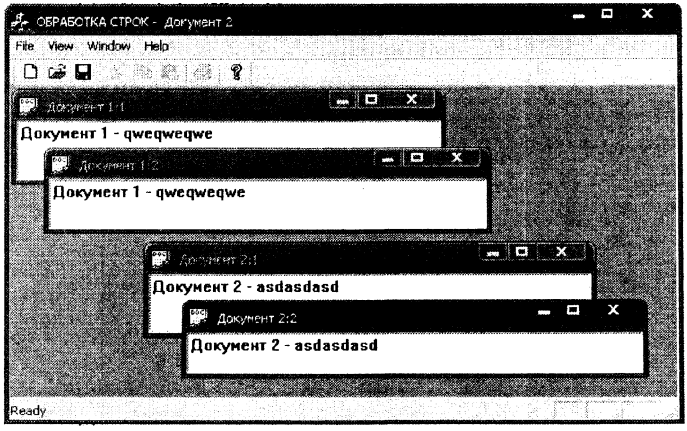

Рисунок 44 – Интерфейс приложения

При этом, если изменять документ в одном окне, то его изображение может меняться синхронно и во всех других окнах.

Если загружается или создается еще один документ, то для этого используется отдельное окно. Указанное иллюстрируется рисунком 44, где загружены два документа и каждый из них отображается одновременно в двух окнах.

Для этого необходимо:

1. Создать автоматический многодокументный каркас (ТКП ДВ), при необходимости переименовать файлы, отменить поддержку печати (в качестве имени приложения здесь задано DV).
2. Скорректировать ресурсы каркаса (меню, справку), а также строки состояния приложения.

При корректировке строк учесть, что здесь используется отдельная строка для настройки всего приложения и отдельная строка для настройки работы с документами конкретного типа (здесь один тип).

Соответственно в методе BOOL App::InitInstance( ) есть фрагмент создания шаблона документа

```
CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(
```

```
            IDR_DVTYPE,
            RUNTIME_CLASS(Doc),
            RUNTIME_CLASS(ChildFrame),
            RUNTIME_CLASS(View));
AddDocTemplate(pDocTemplate); ,
```

где использован идентификатор IDR_DVTYPE. Он же в таблице строк идентифицирует настроечную строку, которую можно заменить, например, на строку

\n **Документ** \n DV\n **DV Files (*.str)**\n **.str**\n **DV.Document**\n DV Document .

Там же есть фрагмент создания главного окна

```
Frame* pMainFrame = new Frame;
if (!pMainFrame->LoadFrame( IDR_MAINFRAME ))
            return FALSE;
m_pMainWnd = pMainFrame;
```

где использован идентификатор IDR_MAINFRAME. Он в таблице строк идентифицирует настроечную строку

ИмяПриложения

заданное мастеру в качестве имени при генерации каркаса (здесь DV), которую можно заменить, например, на строку

IDR_MAINFRAME – ОБРАБОТКА СТРОК .

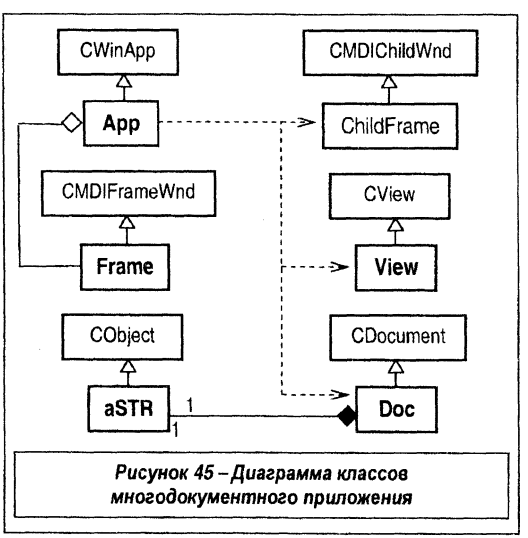

Рисунок 45 – Диаграмма классов многодокументного приложения

3. Добавить описание класса aSTR. Итоговая диаграмма классов приложения представлена на рисунке 45, а состав классов приведен на рисунке 46.

4. В класс документа добавить член-данное aSTR myStr.

5. Отредактировать как в предыдущем примере методы:

- Doc::Doc( ) ;

- BOOL Doc::OnNewDocument( ) ;
- void View::OnDraw(CDC* pDC) ;
- void Doc::Serialize(CArchive& ar).

А также метод void View::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags), обязательно добавив команду

pDoc->UpdateAllViews(this)

для обеспечения обновления отображения документа сразу во всех его окнах.

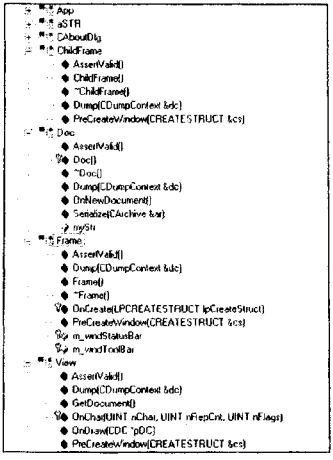

Рисунок 46 – Состав классов

3.4. Приложение с пользовательской иерархией классов

ПРИМЕР 3. *Создать MFC-приложение на базе ТКП ДВ для ведения списка записей о книгах. Каждая запись включает название книги, фамилию автора и год издания.*

Должен быть обеспечен доступ к каждому атрибуту записи, возможность представления записей одной строкой, возможность сравнения записей, что обеспечивает их сортировку в списке. При выводе списка записи должны отображаться в отсортированном виде последовательными строками.

При отображении списка записей должна быть обеспечена возможность горизонтальной и вертикальной прокрутки документа в окне.

Соответственно в качестве базового класса для создания пользовательского класса вида для обеспечения отображения списка будет использован класс CScrollView.

Для организации хранения и обработки документа следует:

- создать пользовательский класс (запись) для хранения записи, производный от класса CObject, что впоследствии позволит организовать сериализацию всего документа;
- создать пользовательский класс (список с использованием шаблона STL list< запись >) для хранения и обработки списка записей.

При этом для описания документа (соответствующих классов предметной области) можно использовать либо механизм наследования, либо агрегацию, что иллюстрируется рисунком 47 на примере организации списка для хранения целых значений. Здесь на диаграмме представлены соответствующие функционально-адекватные классы MY_LIST_1 (с использованием наследования) и MY_LIST_2 (с использованием агрега-

ции) с минимальным набором членов (конструкторы, методы вывода списка и добавления значений-записей). Атрибут int n используется для хранения числа записей в списке, а list < int > L для хранения списка.

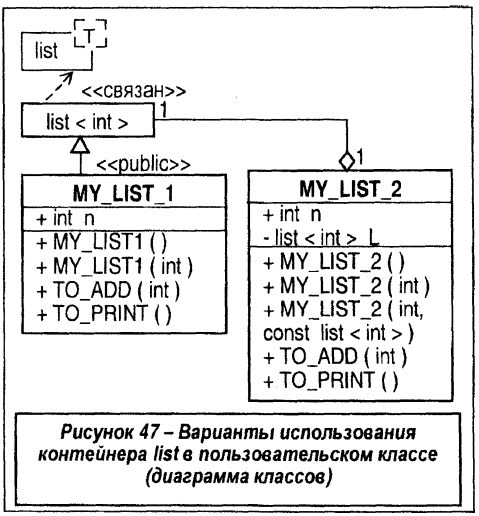

Рисунок 47 – Варианты использования контейнера list в пользовательском классе (диаграмма классов)

Ниже приведено примерное описание класса MY_LIST_1

```
class MY_LIST_1: public list < int >
{
public:
        int n;
        MY_LIST_1( ): list< int >( ) { n = 0; };
        MY_LIST_1( int N ): list< int >( N, 0) { n = N; };
        TO_ADD( int X )
        {
                push_back( X );
                n++;
        };
        TO_PRINT( )
        {
                list< int >: :iterator p = begin( );
                while ( p != end ( ) )
                {
                        cout << endl << ( *p );
                        p++;
                };
        };
```

А. Разработка классов предметной области.

Опишем предметную область задачи - список книг. Здесь будет использован первый вариант, соответственно предметная область может быть представлена следующим набором классов:

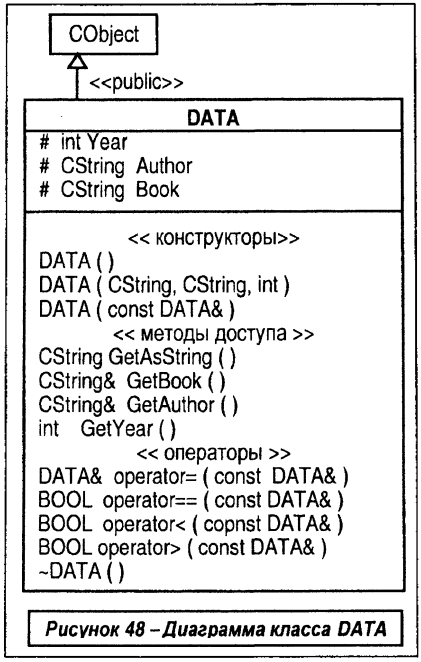

Рисунок 48 – Диаграмма класса DATA

1) класс **DATA**: public CObject для хранения и обработки записи о книге (Название, Автор, Год издания), представленный на рисунке 48;

2) класс list< DATA > для хранения самого списка записей;

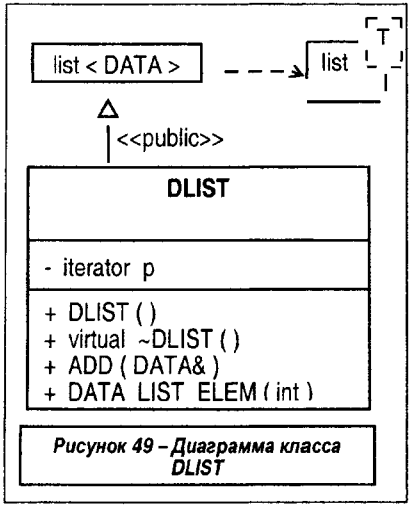

Рисунок 49 – Диаграмма класса DLIST

3) класс **DLIST**: public list< DATA > для хранения и обработки всех атрибутов списка, включая сам список, дополнительную информацию (например, итератор для доступа к списку и т.п.), представленный на рисунке 49.

Здесь методы ADD (DATA&), DATA LIST_ELEM (int) используются соответственно для добавления новой записи и для получения доступа к записи по ее номеру.

Это описание предметной области – иерархию классов (рисунок 49) можно заранее отладить и протестировать, например, создав консольное приложение на базе ТКП с использованием библиотеки MFC. Полученный результат - соответствующие файлы, описывающие диаграмму классов, далее можно включить в проект основного приложения и использовать для его разработки.

Распределение описаний классов предметной области по файлам проекта представлено на рисунке 50.

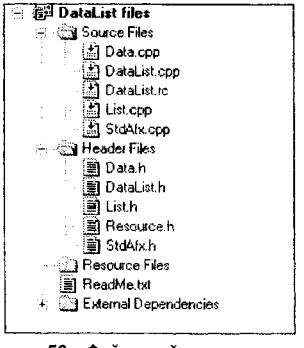

Рисунок 50 – Файловый состав проекта

Примерное описание файлов проекта предметной области приведено ниже

Файл Data.h

```
using namespace std;

class DATA : public CObject
{
public:
    CString GetAsString( );
    DATA( ):Book(" "), Author(" "), Year( 0 ) { }
    DATA( CString F, CString D, int P ) : Book( F ), Author( D ), Year( P ) { }
    DATA( const DATA & d ) : Book( d.Book ), Author( d.Author ), Year( d.Year ) { }
    const DATA & operator=( const DATA & d )
    {
        Book = d.Book;
        Author = d.Author;
        Year = d.Year;
        return *this;
    }
    BOOL operator==( const DATA & d )
    {   return ( Book == d.Book && Author == d.Author && Year == d.Year );   }
    BOOL operator<( const DATA & );
    BOOL operator>( const DATA & );
```

```
    virtual ~DATA();
    CString &GetBook( ) { return Book; }
    CString &GetAuthor( ) { return Author; }
    int GetYear( ) { return Year;}
protected:
    int Year;
    CString Author;
    CString Book;
};
```

Файл Data.cpp

```
#include "stdafx.h"
#include "Data.h"

DATA::~DATA( ) { };

CString DATA::GetAsString( )
{
    CString strTemp = "";
    CString strRet = Book;
    strRet += " - автор ";
    strRet += Author;
    strRet += " - ";
    strTemp.Format("%.2d",Year );
    strRet += strTemp;
    strRet += " год ";
    return strRet;
};

BOOL DATA::operator<( const DATA & d )
{
    if( Book < d.Book ) return TRUE;
    if( Book > d.Book) return FALSE;
    if( Author < d.Author ) return TRUE;
    return FALSE;
};

BOOL DATA::operator>( const DATA & d )
{
    if( Book > d.Book) return TRUE;
    if( Book < d.Book) return FALSE;
    if( Author > d.Author ) return TRUE;
    return FALSE;
};
```

Файл List.h

```
#include "Data.h"

class DLIST : public list< DATA >
{
    iterator p;
public:
```

```
    ADD( DATA & d );
    DLIST();
    virtual ~DLIST();
    DATA LIST_ELEM( int i )
    {
            int j;
            p = begin( );
            if (( i >= 0 ) && ( i<size( ) ) )
            {
                    for ( j=0; j<i; j++ )   p++;
            }
            return  *p;
    };
};
```

List.cpp

```
#include "stdafx.h"
#include "List.h"

DLIST::DLIST( ) { p = begin( ); }
DLIST::~DLIST( )  { }
DLIST::ADD( DATA &d )
{
     push_back( d );
     sort( );
}
```

Файл DataList.h

```
#include  "resource.h"
```

Файл DataList.cpp

```
#include  "stdafx.h"
#include  "DataList.h"
#include  "List.h"
...
CWinApp  theApp;
using namespace std;
int _tmain( int argc, TCHAR* argv[ ], TCHAR* envp[ ] )
{
          ....
}
```

Файл stdafx.h

```
#include <afxwin.h>
...
#include <iostream>
#include <list>
...
```

Файл stdafx.cpp

```
#include "stdafx.h"   .
```

Б. Разработка основного приложения на базе ТКП ДВ.

Для этого необходимо выполнить следующие действия.

1. Создать однодокументный автокаркас (например, с именем STU) без поддержки баз данных и составных документов с классом CScrollView в качестве базового для создания пользовательского класса вида.

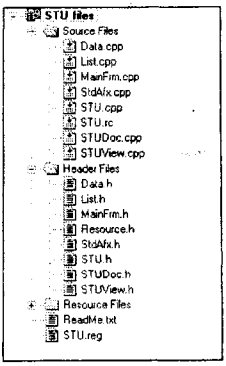

Рисунок 51 – Файловый состав проекта

2. Для последующей вставки в проект ранее разработанных классов документа (предметной области) следует скопировать файлы Data.h, Data.cpp, List.h и List.cpp в рабочий каталог приложения (STU).

3. В окне рабочей среды выбрать вкладку – броузер FileView, щелкнуть правой кнопкой мыши на значке проекта с именем STU Files. В контекстном меню выбрать пункт Add Files to Project и последовательно осуществить добавку к проекту файлов Data.h, Data.cpp, List.h и List.cpp. Примерный состав проекта представлен на рисунке 51.

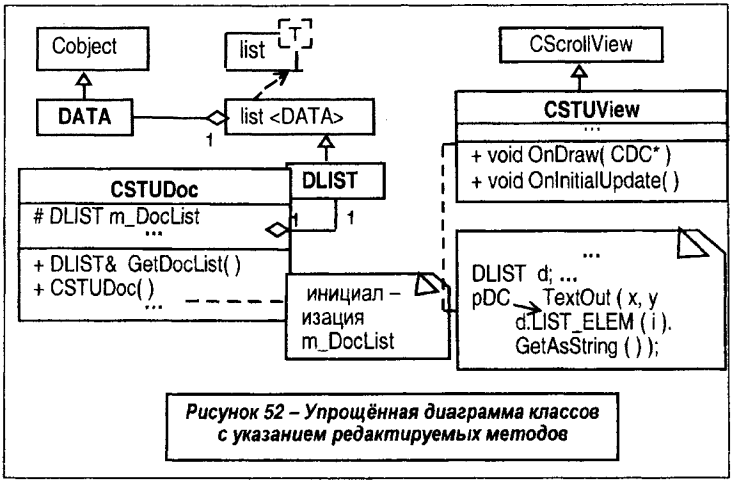

Рисунок 52 – Упрощённая диаграмма классов с указанием редактируемых методов

Теперь необходимо внести изменения в состав членов классов, в методы (см. рисунок 52).

4. Описать в классе документа приложения CSTUDoc объект-переменную **m_DocList** класса DLIST.

Для этого выбрать вкладку – броузер-ClassView (рисунок 53), щелкнуть правой кнопкой мыши на значке класса CSTUDoc. В контекстном меню выбрать пункт Add Member Variable.

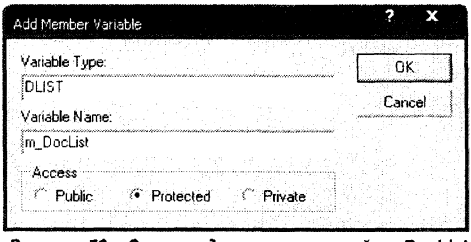

Рисунок 53 – Окно создания переменной m_DocList

В диалоговом окне Add Member Variable в поле тип переменной (Variable Type) ввести тип DLIST, а в поле имя переменной (Variable Name) указать m_DocList. В качестве спецификатора доступа задать Protected.

Для добавления переменной нажать кнопку OK.

5. Подключить к проекту описания данных. Для этого в файл STUDoc.h надо вставить инструкцию

```
#include "List.h"
```

6. Добавить в раздел public в объявлении класса CSTUDoc функцию, имеющую доступ к защищенной переменной m_DocList и возвращающей ее значение

```
const DLIST &CSTUDoc::GetDocList( ) { return m_DocList; }
```

7. Инициализировать в конструкторе документа (для целей отладки) создаваемый документ, то есть список значений m_DocList, например как

```
CSTUDoc::CSTUDoc( )
{
    m_DocList.ADD(DATA("Первая книга","Иванов",2011));
    m_DocList.ADD(DATA("Первая книга","Сидоров",2012));
    m_DocList.ADD(DATA("Вторая книга","Иванов",2002));
    ...
}
```

8. В методе CSTUView::OnDraw( ) необходимо реализовать (для отладки) ее упрощенную версию – вывод данных из документа, например как

```
void CSTUView::OnDraw( CDC* pDC )
{
    CSTUDoc* pDoc = GetDocument( );
    ASSERT_VALID( pDoc );
    DLIST pData = pDoc->GetDocList( );
    int i;
    for( i = 0; i < pData.size( ); i++ )
    {
        pDC->TextOut( 0, 10 + i * 15, pData.LIST_ELEM( I ).GetAsString( ) );
    };
}
```

9. Для обеспечения прокрутки текста в обоих направлениях выполнить настройку в методе

```
void CSTUView::OnInitialUpdate( )
{
    CScrollView::OnInitialUpdate();
    CSize sizeTotal;
    sizeTotal.cx = sizeTotal.cy = 1000;
    SetScrollSizes(MM_TEXT, sizeTotal);
}
```

Указанные действия при запуске приложения приведут к отображению инициализированного списка в следующем виде (рисунок 54).

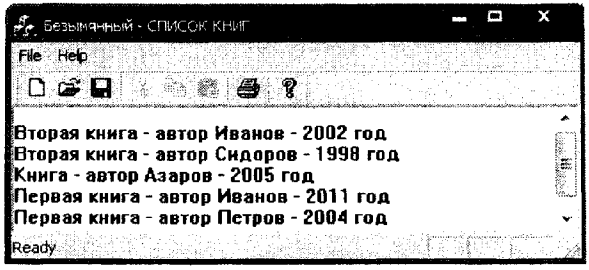

Рисунок 54 – Интерфейс приложения

4. ПОРЯДОК ВЫПОЛНЕНИЯ

ТЕМА РАБОТЫ: "Типовой автоматический каркас MFC-приложения архитектуры документ-вид (ТКП ДВ). Мастер AppWizard(exe)".

ЦЕЛЬ РАБОТЫ:

- изучение типовых каркасов приложений, а также базовых классов вида;
- изучение каркаса MFC-приложений архитектуры документ-вид (ТКП ДВ);
- изучение особенностей сооздания каркаса с помощью мастера Visual Studio AppWizard(exe) и особенностей автокаркаса;
- настройка и использование каркаса в пользовательских приложениях.

СОДЕРЖАНИЕ ОТЧЕТА:

- описание ТКП ДВ (интерфейса, диаграмм классов, диаграмм взаимодействия объектов);
- описание разработанных приложений.

ПРИМЕЧАНИЕ:

- для всех заданий в отчете следует приводить диаграммы классов и компоновки (в нотации UML);
- каждое реализованное задание (приложение) предъявлять на ПЭВМ для проверки преподавателю.

ПОРЯДОК ВЫПОЛНЕНИЯ.

1. **Изучить теоретический материал**

- ознакомиться с разновидностями каркасов приложений и их возможностями (см. § 1.1);
- изучить особенности базовых классов MFC для создания пользовательских классов вида (см. § 1.2);
- ознакомиться с процедурой создания каркаса ТКП ДВ с помощью мастера; изучить особенности каркаса, создаваемого автоматически, ознакомиться с архитектурой каркаса, используемых классах и методах (см. § 2.1-2.5).

2. **Создать** "пустое" приложение (типовой каркас приложения) (с именем EX1) в соответствии с § 2.1. Это каркас MFC-приложения с главным окном, но без обработки сообщений. Изучить свойства приложения, разработать диаграммы UML.

3. **Модифицировать** "пустое" приложение (типовой каркас приложения) (с именем EX1):

- изменить настройки (строку настройки);
- редактировать пиктограммы каркаса;
- редактировать главное меню;
- редактировать содержимое справочного окна;
- удалить справочное окно из состава каркаса.

4. **Воспроизвести** приложение (с именем EX2) для обработки строки символов (см. § 3.1 – ПРИМЕР 1).

5. **Воспроизвести** приложение (с именем EX3) для обработки строки символов (см. § 3.2 – ПРИМЕР 1).

6. **Воспроизвести** приложение (с именем EX4) для обработки строки символов (см. § 3.3 – ПРИМЕР 2).

7. **Воспроизвести** приложение (с именем EX5) (см. § 3.4 – ПРИМЕР 3).

8. **Модифицировать** приложение (с именем EX1, 2 – ПРИМЕР 1), внеся следующие изменения:

- измените главное меню, внеся пункты редактирования строки для удаления, добавления символов;
- используйте левую клавишу мыши для стирания последнего символа строки;
- создайте кнопки панели инструментов для добавленных пунктов меню;
- обеспечьте поддержку команд работы с буфером приложения;
- обеспечьте возможность переноса символов на следующую строку в окне вида;
- обеспечьте прокрутку документа;
- добавьте диалоговое окно для просмотра и редактирования строки.

9. **Модифицировать** приложение (с именем EX5 – ПРИМЕР 3), внеся следующие изменения:

1) обеспечить автоматическую сортировку записей;

2) оснастить интерфейс пользовательским меню, в частности для выполнения команды – добавить запись;

3) обеспечить добавку новых записей к списку с учетом сортировки через диалоговое окно (например, в составе пользовательского класса вида) на базе метода ADD (DATA&). При этом обновленный список должен отображаться в окне просмотра автоматически;

4) обеспечить сохранение-загрузку документов на базе механизма сериализации;

5) разработать аналогичное приложение с использованием механизма агрегации для описания предметной области.

10. **Разработать** приложение (с именем EX6) для хранения, редактирования и отображения плоских фигур, соединенных прямыми линиями. Ввод координат линий производить в поле окна просмотра, используя клавиши "мыши". Примерный вид интерфейса со стандартным видом документа – графическим изображением фигуры представлен на рисунке 55. Предусмотреть дополнительный вид документа - вывод массива координат.

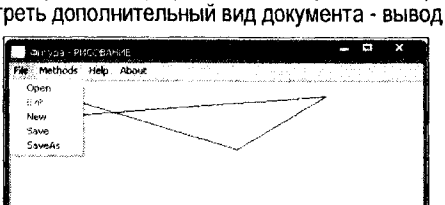

Рисунок 55 – Интерфейс приложения

ЛИТЕРАТУРА

1. Шилдт, Г. Самоучитель C++. – 3-е изд. – СПб.: БХВ-Петербург, 2003. – 688 с.

2. Паппас, К. Visual C++. Руководство для профессионалов / К. Паппас, У. Мюррей; пер. с англ. - СПб: BHV -Санкт-Петербург, 1996.

3. Паппас, К. Эффективная работа: Visual C++.NET / К. Паппас, У. Мюррей. – СПб.: Питер, 2002. – 816 с.

4. Поляков, А.Ю. Методы и алгоритмы компьютерной графики в примерах на Visual C++ / А.Ю. Поляков, В.А. Брусенцев. – СПб.: БХВ-Петербург, 2003. – 560 с.

5. Орлов, С.А. Технологии разработки программного обеспечения: учебник для вузов. – СПб.: Питер, 2004. – 527 с.

6. Шилдт, Г. Справочник программиста по C/C++, 3-е изд. – М.: Изд. Дом Вильямс, 2003. – 432 с.

7. Шмуллер, Дж. Освой самостоятельно UML за 24 часа. – М.: Изд. Дом Вильямс, 2002. – 352 с.

8. Страуструп, Б. Язык программирования СИ++. – М.: Радио и связь, 1991. – 352 с.

9. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++, 2-е изд.; пер. с англ. – М.: Издательство БИНОМ; СПб: Невский диалект, 1998. – 560 с.

10. Финогенов, К.Г. Win32. Основы программирования. – М.: ДИАЛОГ-МИФИ, 2002. – 416 с.

11. Павловская, Т.А. C++. Объектно-ориентированное программирование: практикум / Т.А Павловская, Ю.А. Щупак. – СПб.: Питер, 2004. – 265 с.

ПРИЛОЖЕНИЕ. Листинг каркаса

Здесь представлены основные фрагменты однодокументного каркаса архитектуры документ-вид, сгенерированного автоматически для приложения с именем DV, без поддержки баз данных и составных документов. Показаны основные места вставки текста по желанию разработчика.

```
// DV.h
...
#include "resource.h"
//////////////////////////////////////////////////////////////////////
class APPLICATION : public CWinApp
{
public:
    APPLICATION( );
    //{{AFX_VIRTUAL(APPLICATION)
    public:
    virtual BOOL InitInstance( );
    //}}AFX_VIRTUAL
    //{{AFX_MSG(APPLICATION)
    afx_msg void OnAppAbout( );
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
...

// DV.cpp
#include "stdafx.h"
#include "DV.h"
#include "MainFrm.h"
#include "DVDoc.h"
#include "DVView.h"
...
//////////////////////////////////////////////////////////////////////
BEGIN_MESSAGE_MAP(APPLICATION, CWinApp)
    //{{AFX_MSG_MAP(APPLICATION)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    //}}AFX_MSG_MAP
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

APPLICATION::APPLICATION( ) {  }

APPLICATION theApp;

BOOL APPLICATION::InitInstance( )
{
...
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
            IDR_MAINFRAME,
```

```
            RUNTIME_CLASS(DOCUMENT),
            RUNTIME_CLASS(WINDOW),
            RUNTIME_CLASS(VIEW));
    AddDocTemplate(pDocTemplate);
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    ...
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();
    return TRUE;
}

////////////////////////////////////////////////////////////////////////
class CAboutDlg : public CDialog
{
public:
    CAboutDlg( );
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    //}}AFX_VIRTUAL
...
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg( ) : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

void APPLICATION::OnAppAbout( )
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal( );
}
```

// MainFrm.h

```
////////////////////////////////////////////////////////////////////
...
class WINDOW : public CFrameWnd
{
protected:
    WINDOW( );
    DECLARE_DYNCREATE(WINDOW)
public:
    //{{AFX_VIRTUAL(WINDOW)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL
    virtual ~WINDOW( );
protected:
    CStatusBar  m_wndStatusBar;
    CToolBar    m_wndToolBar;
    //{{AFX_MSG(WINDOW)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

// MainFrm.cpp
#include "stdafx.h"
#include "DV.h"
#include "MainFrm.h"
////////////////////////////////////////////////////////////////////
IMPLEMENT_DYNCREATE(WINDOW, CFrameWnd)
BEGIN_MESSAGE_MAP(WINDOW, CFrameWnd)
    //{{AFX_MSG_MAP(WINDOW)
    ON_WM_CREATE( )
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
...
WINDOW::WINDOW( ) { }

WINDOW::~WINDOW( ) { }

int WINDOW::OnCreate(LPCREATESTRUCT lpCreateStruct) { ... }

BOOL WINDOW::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // код пользователя
    return TRUE;
}
...

// DVDoc.h
////////////////////////////////////////////////////////////////////
...
class DOCUMENT : public CDocument
{
```

```
protected:
    DOCUMENT( );
    DECLARE_DYNCREATE(DOCUMENT)
public:
    //{{AFX_VIRTUAL(DOCUMENT)
    public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    //}}AFX_VIRTUAL
    virtual ~DOCUMENT( );
...
    DECLARE_MESSAGE_MAP()
};
```

// DVDoc.cpp

```
#include "stdafx.h"
#include "DV.h"
#include "DVDoc.h"
////////////////////////////////////////////////////////////////////////
IMPLEMENT_DYNCREATE(DOCUMENT, CDocument)
BEGIN_MESSAGE_MAP(DOCUMENT, CDocument)
    //{{AFX_MSG_MAP(DOCUMENT)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

DOCUMENT::DOCUMENT( )  {  }

DOCUMENT::~DOCUMENT( )  {  }

BOOL DOCUMENT::OnNewDocument( )
{
    if (!CDocument::OnNewDocument( ))
        return FALSE;
    // код пользователя
    return TRUE;
}

void DOCUMENT::Serialize(CArchive& ar)
{
    if (ar.IsStoring( ))
    {
        // код пользователя
    }
    else
    {
        // код пользователя
    }
}
...
```

// DVView.h

```
////////////////////////////////////////////////////////////////////////
...
```

```
class VIEW : public CView
{
    VIEW( );
    DECLARE_DYNCREATE(VIEW)
public:
    DOCUMENT* GetDocument( );
    //{{AFX_VIRTUAL(VIEW)
    public:
    virtual void OnDraw(CDC* pDC);
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    protected:
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    //}}AFX_VIRTUAL
public:
    virtual ~VIEW( );
...
    DECLARE_MESSAGE_MAP()
};
...
```

// DVView.cpp

```
#include "stdafx.h"
#include "DV.h"
#include "DVDoc.h"
#include "DVView.h"
...
////////////////////////////////////////////////////////////////////
IMPLEMENT_DYNCREATE(VIEW, CView)
BEGIN_MESSAGE_MAP(VIEW, CView)
    //{{AFX_MSG_MAP(VIEW)
    //}}AFX_MSG_MAP
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

VIEW::VIEW( )
{
    // код пользователя
}

VIEW::~VIEW( ) { }

BOOL VIEW::PreCreateWindow(CREATESTRUCT& cs)
{
    // код пользователя
    return CView::PreCreateWindow(cs);
}

void VIEW::OnDraw(CDC* pDC)
{
```

```
    DOCUMENT* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // код пользователя
}

BOOL VIEW::OnPreparePrinting(CPrintInfo* pInfo)  {  ... }

void VIEW::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // код пользователя
}

void VIEW::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // код пользователя
}
...

DOCUMENT* VIEW::GetDocument( ) {  ...     }
```

Учебное издание

Муравьев Геннадий Леонидович
Мухов Сергей Владимирович
Хвещук Владимир Иванович

Методическое пособие

"РАЗРАБОТКА ПРИЛОЖЕНИЙ НА БАЗЕ КАРКАСА ДОКУМЕНТ-ВИД (мастер MFC AppWizard)"

Ответственный за выпуск: Муравьев Г.Л.
Редактор: Боровикова Е.А.
Компьютерная вёрстка: Горун Л.Н.
Корректор: Никитчик Е.В.

Подписано к печати 30.07.2012 г. Бумага «Снегурочка». Формат 60x84 $^{1}/_{16}$.
Гарнитура Arial Narrow. Усл. печ. л. 3,02. Уч. изд. л. 3,25.
Заказ № 841. Тираж 50 экз. Отпечатано на ризографе Учреждения образования
«Брестский государственный технический университет»
224017, г. Брест, ул. Московская, 267.