Урок №6. Создание MDI приложений

Содержание

1. Определение архитектуры MDI

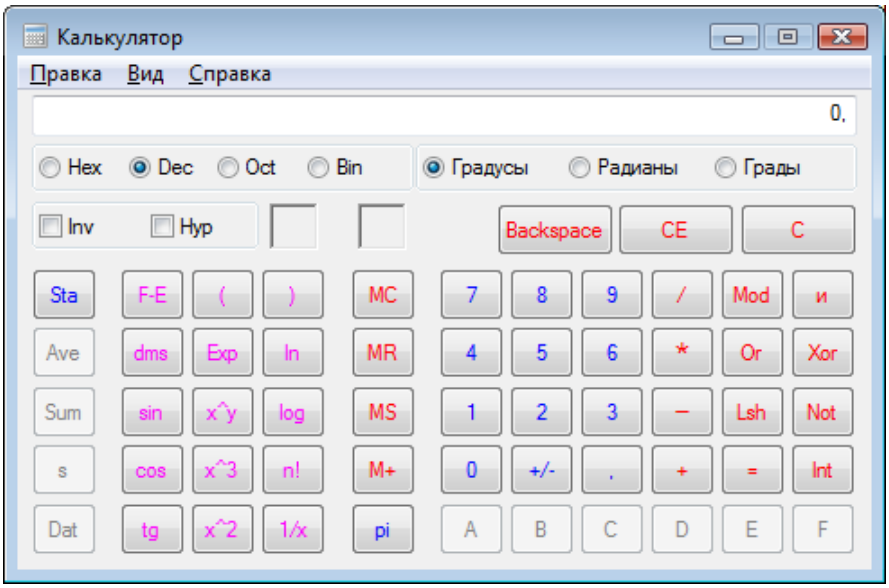
Архитектура MDI – это архитектура интерфейса приложения, при котором определяется глобальное окно (Контейнер) с общими элементами управления и панелями инструментов, а также множество дочерних окон, которые могут размещаться внутри контейнера и отображать разные документы, выборки данных, разнообразные представления данных.

Существует следующие разновидности интерфейсов приложений:

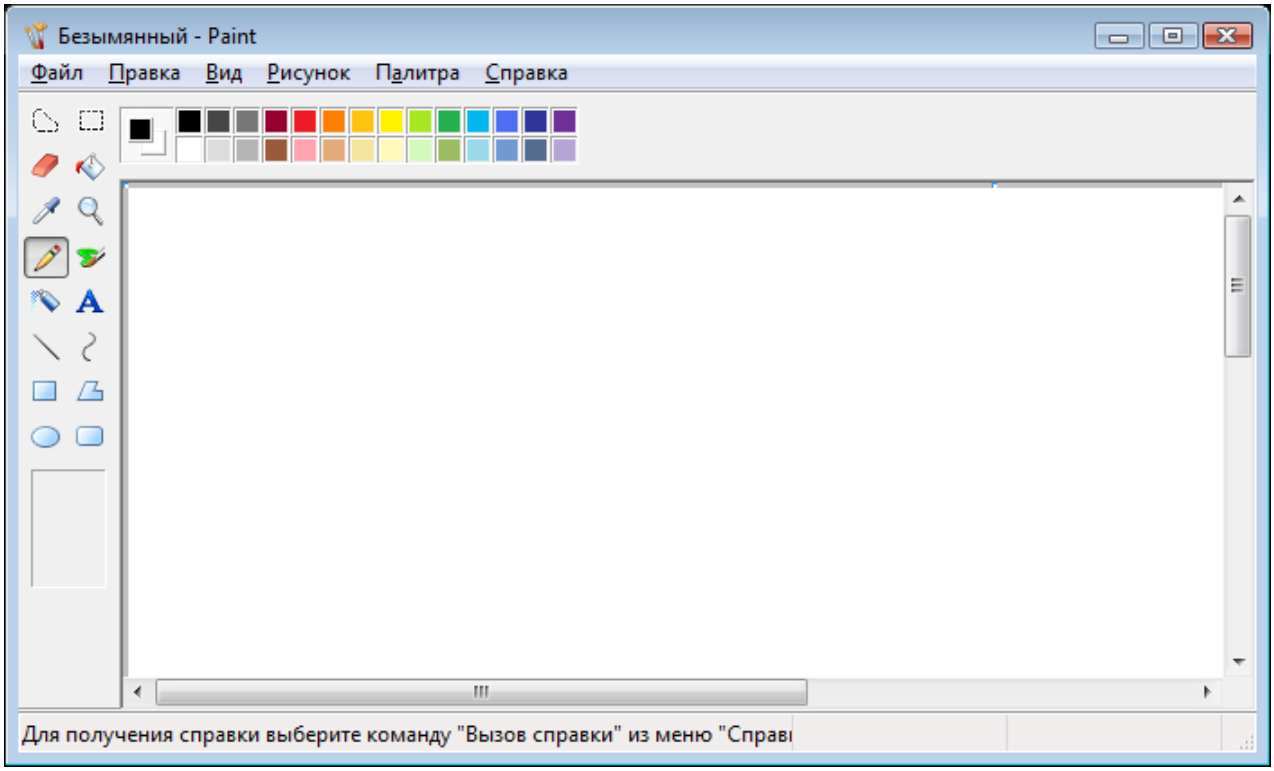
- Приложения, основанные на диалоговом окне. Такие приложения предоставляются пользователям в виде единого диалогового окна, с помощью которого может быть осуществлен доступ ко всем функциональным возможностям;
- Однодокументные интерфейсы (Single Document Interface, SDI). Такие приложения предоставляются пользователям в виде меню, одной или нескольких линеек инструментов и одного окна, в котором пользователь и может выполнять определенные действия;
- Многодокументные интерфейсы (Multi-Document Interface, MDI). Такие приложения предоставляются пользователям в таком же виде, что и SDI-приложения, однако обладают способностью одновременно поддерживать несколько открытых дочерних окон в главном окне;
- Многодокументный интерфейс с независимыми по размещению и отображаемыми на панели задач окнами;
- Многодокументный интерфейс с вкладками (Tabbed document interface) — разновидность графического интерфейса пользователя, в котором каждый документ находится на отдельной вкладке одного окна.

2. Классификация приложений

Приложения, основанные на диалоговом окне, обычно представляют собой небольшие одноцелевые приложения, которые ориентированы либо для решения конкретной задачи, требующей ввода небольшого количества данных, либо для работы с какими-то необычными типами данных. В качестве примера такого приложения можно привести Calculator, поставляемый вместе с MS Windows.

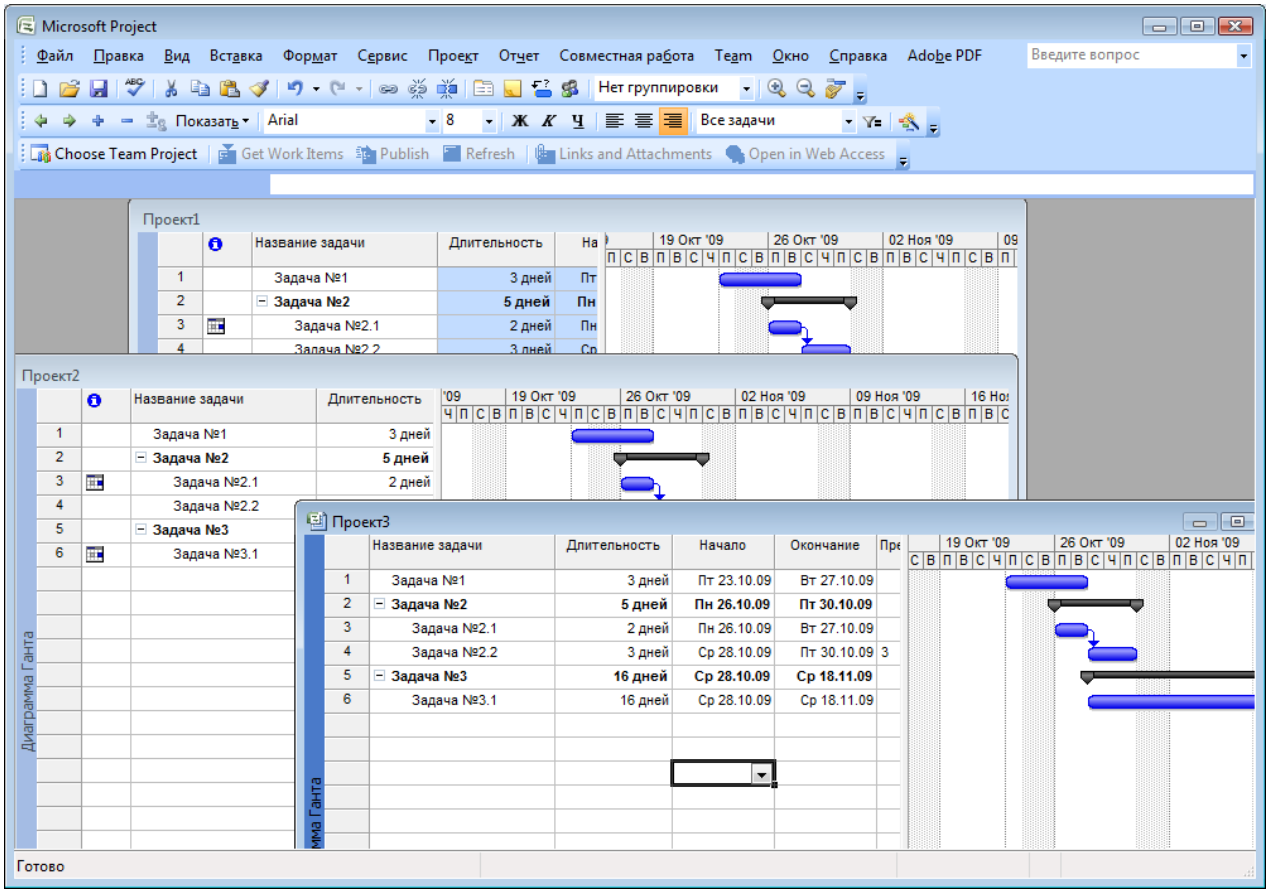


Однодокументные интерфейсы, как правило, предназначаются для решения какой-то одной конкретной задачи, при этом они позволяют пользователю загружать в приложение единственный документ, с которым он и будет вести работу. Эта задача предполагает выполнение пользователем большого количества действий, и зачастую пользователю могут потребоваться возможности, позволяющие сохранять или загружать плоды своего труда. Хорошим примером SDI-приложений могут служить MS Paint и WordPad, также поставляемые совместно с MS Windows.



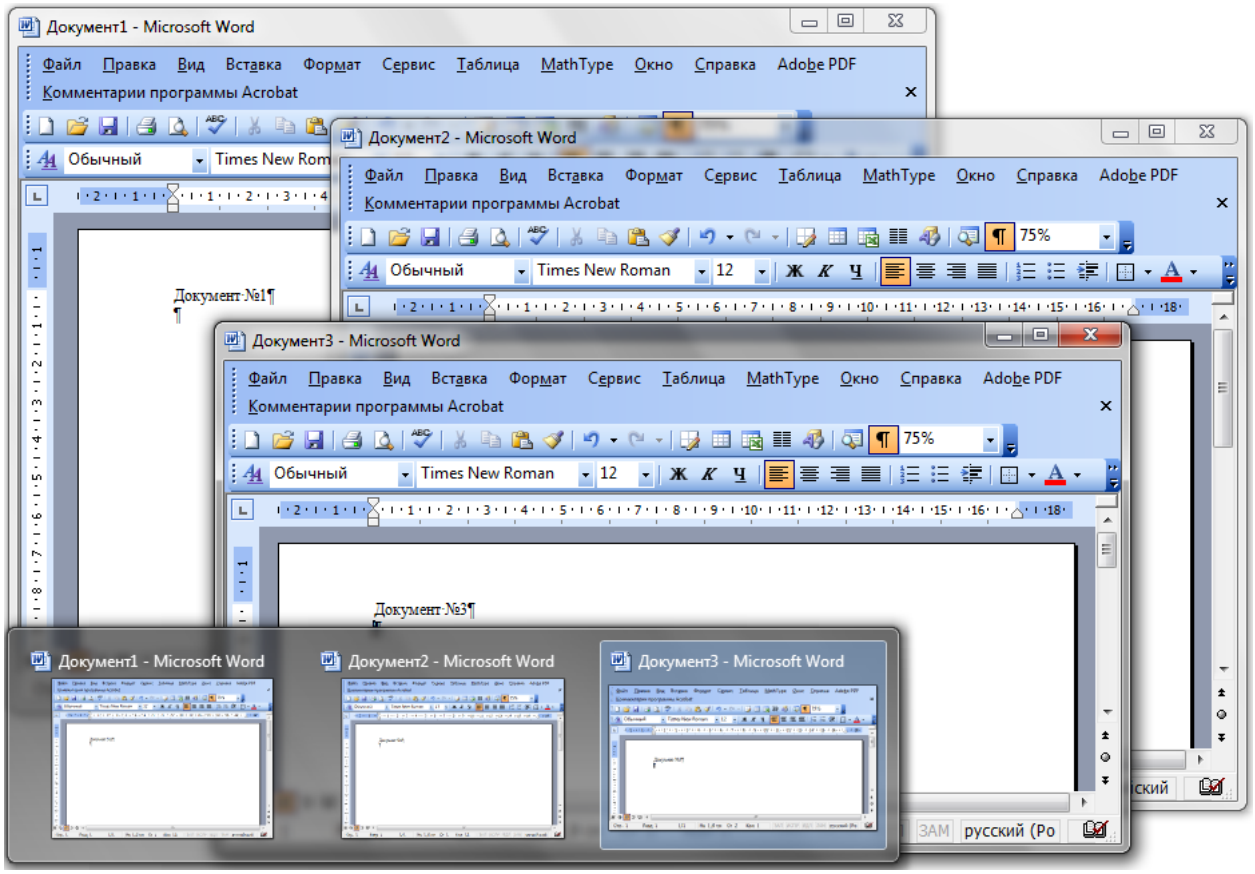
Однако такие приложения допускают открытие только одного документа в каждый конкретный момент времени, поэтому если пользователю требуется открыть второй документ, то ему необходимо открывать новый экземпляр SDI-приложения, у которого будут отсутствовать связи с первым документом и, следовательно, конфигурация, созданная для первого экземпляра, не окажет никакого влияния на конфигурацию второго. Например, мы в MS Paint выбрали красный цвет в качестве цвета рисования, затем открываем второй экземпляр MS Paint, а здесь в качестве цвета, используемого для рисования, выбирается цвет по умолчанию. Он будет черный.

Многодокументные интерфейсы почти полностью аналогичны SDI-приложениям за исключением того, что они обладают возможностью поддерживать более одного открытого документа в различных дочерних (внутренних) окнах, которые могут быть открыты одновременно. Одним из простых признаков MDI-приложения является наличие пункта Window (Окно) на правой стороне линейки меню перед пунктом Help. Примерами MDI-приложений служат Adobe Acrobat Reader и MS Project.



Четвертый тип приложений представлен MS Office 2003. Этот тип является смесью SDI- и MDI-приложений: окна, предоставляемые пользователю, имеют различное место-

положение, и каждое окно отображается на панели задач. Такое приложение представляет собой несколько MDI-приложений, поскольку основное приложение не будет закрыто до тех пор, пока не будут закрыты все окна, а с помощью пункта меню Windows можно выбирать, какой именно из открытых документов будет просматриваться, хотя при этом собственно пользовательский интерфейс представляет собой SDI-интерфейс.



3. Преимущества MDI

- В интерфейсе типа MDI панель меню и панель инструментов общая для всех дочерних окон, что уменьшает загроможденность экрана элементами интерфейса и увеличивает его полезную площадь.
- Все дочерние окна приложения можно прятать/показывать, сворачивать/разворачивать и проводить с ними другие манипуляции, как с одним окном.
- Дочерние окна можно автоматически размещать «черепицей» или «каскадом» в главном окне.

- Упрощается управление разными окнами (документами, представлениями), их взаимодействие между собой, обмен данными.
- Увеличение скорости и экономия памяти при работе в одном окне (приложении), скорость переключения между дочерними окнами также выше, чем между равноправными в среде операционной системы.
- В некоторых приложениях предусмотрены «горячие сочетания клавиш» для быстрой навигации, в частности, для переключения между окнами. Это ещё более повышает скорость и удобство работы с приложением, так как не задействуются дополнительные ресурсы операционной системы.

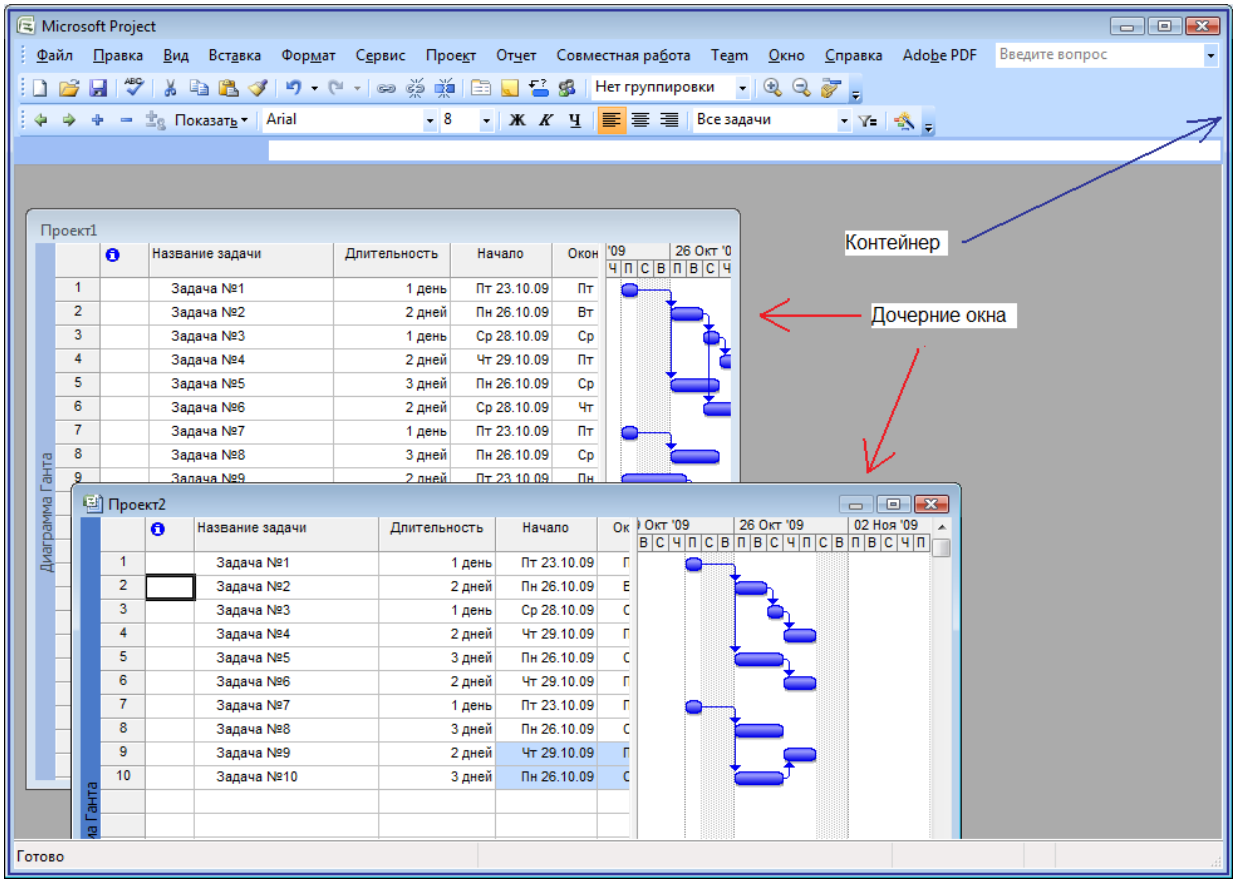
4. Недостатки архитектуры MDI

- Затруднительно (чаще всего, невозможно) выводить содержимое разных дочерних окон на разные мониторы.
- Также невозможно выводить их содержимое на разные виртуальные рабочие столы.
- MDI может затруднить параллельную работу с разными приложениями, так как переключение между внешними окнами разных программ и дочерними окнами другой неудобно.
- Плавающие панели инструментов одного приложения могут перекрывать рабочее окно другого, загораживая обзор, а иногда и сбивая пользователя с толку - какая панель к какому приложению относится.
- Пользователю нужно привыкать к обоим типам интерфейса, так как введение MDI не отменяет полностью использование SDI, который заложен в большинстве операционных систем.
- Многие программные менеджеры окон предоставляют более гибкие возможности для работы с группами окон, чем MDI-интерфейс того или иного приложения.

5. Общая схема интерфейса MDI приложения

Каждое MDI приложение имеет три основные составляющие: Одну (и только одну) родительскую форму MDI, одну и более (обычно больше) дочерних форм MDI, и основное меню MDI

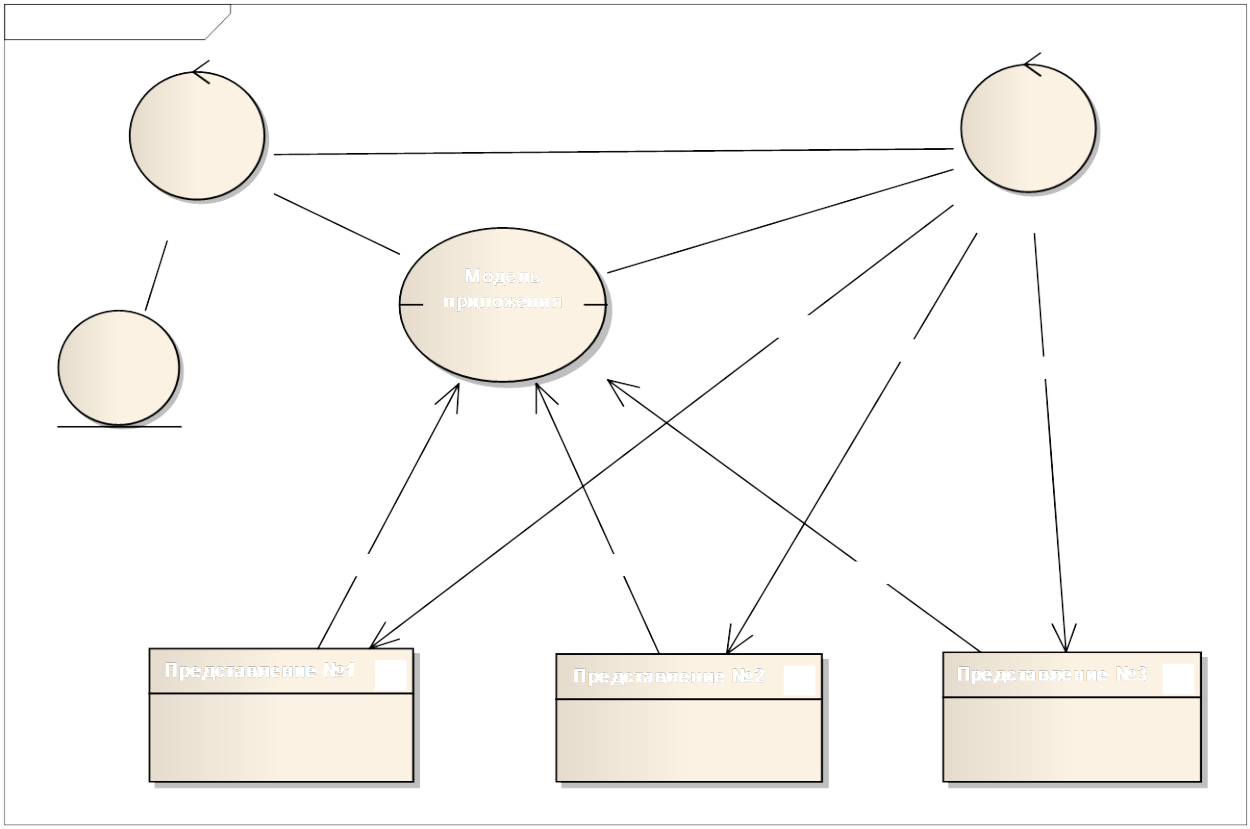
Для создания MDI, во-первых, необходимо, чтобы решаемая пользователем задача требовала одновременно несколько открытых документов, или разных представлений одних и тех же данных. Примером задач такого рода является текстовый редактор или программа просмотра документов. Во-вторых, необходимо предусмотреть панели инструментов для наиболее часто выполняемых в приложении операций, таких как изменение стиля шрифта, загрузка и сохранение документов. В-третьих, обязательно следует предусмотреть пункт меню Window, который бы позволял пользователю изменять положение открытых окон друг относительно друга (накладывая их друг на друга в виде черепицы или каскада) и предоставлял бы ему список всех открытых окон. Еще одной особенностью MDI-приложений является то, что оно должно быть интегрировано в основное меню приложения.



6. Принципы создания MDI приложений

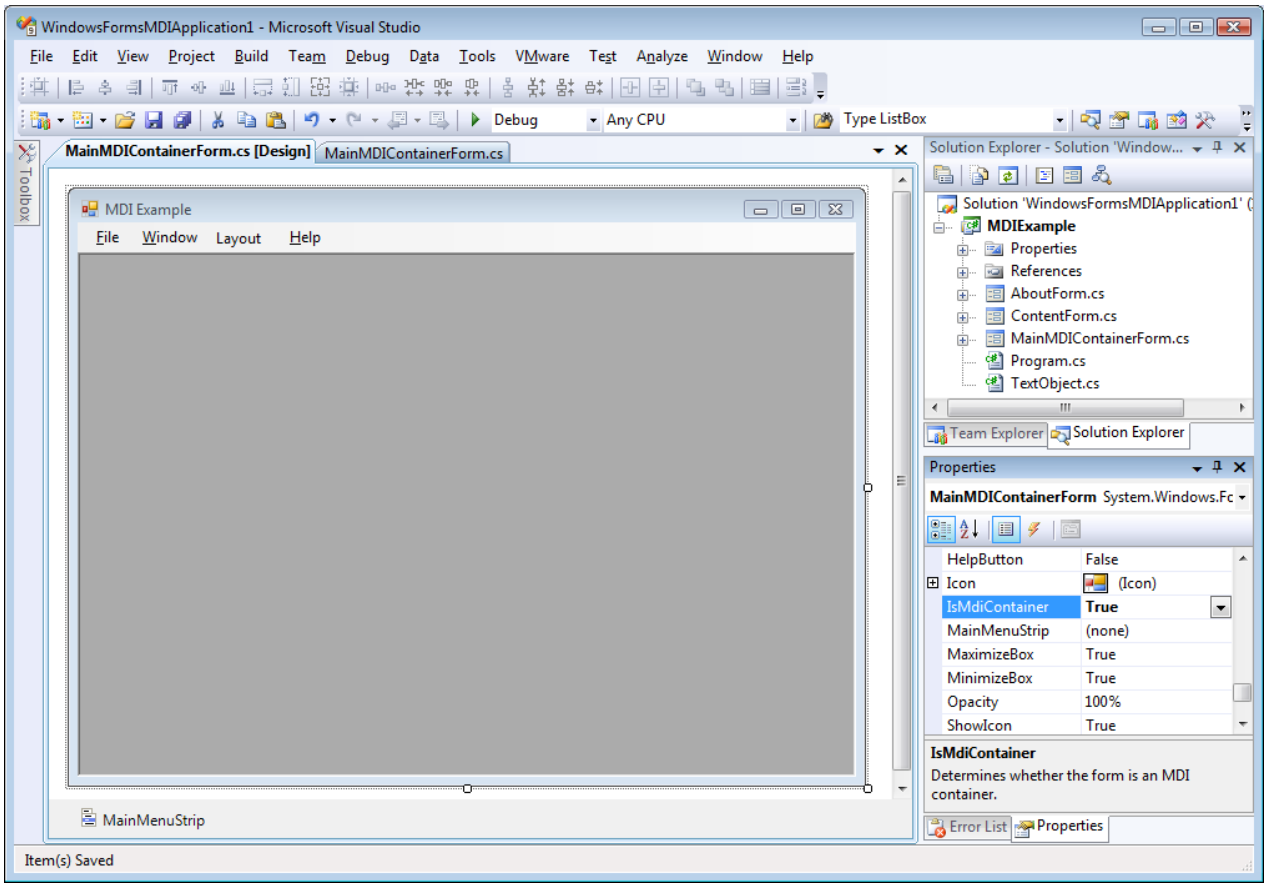
6.1. Принципы построения приложений

Реализация приложения на основе архитектуры MDI предусматривает отделение модулей представления данных, обеспечивая возможность множественного представления:

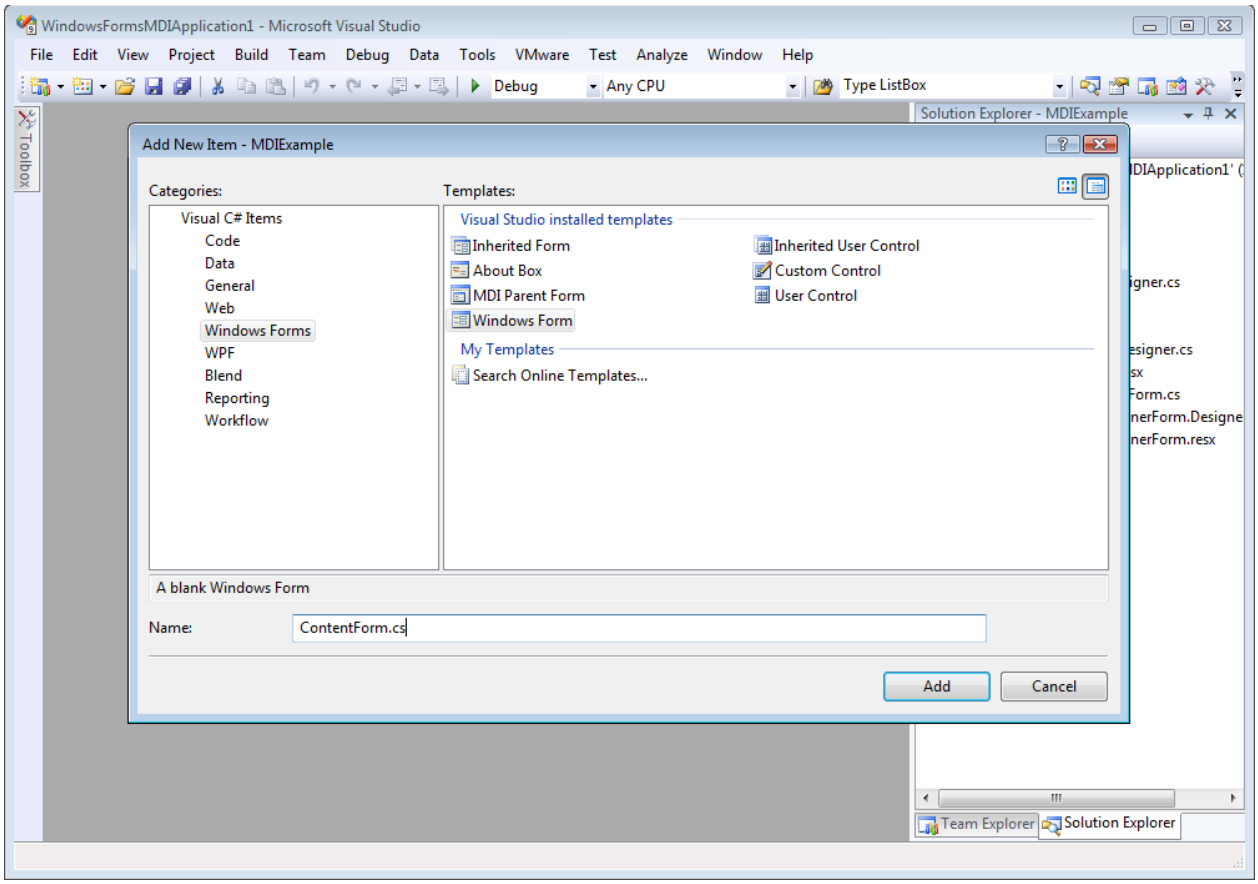


6.2. Создание MDI приложения

При создании MDI-приложения следует начинать с того же, с чего начинается создание любого другого приложения, – с создания Windows Application в Visual Studio. Чтобы превратить основное окно приложения из формы в MDI-контейнер, достаточно просто присвоить свойству формы IsMdiContainer значение true. При этом произойдет изменение цвета фона, указывающее на то, что там не следует размещать видимые управляющие элементы, хотя такая возможность по-прежнему существует и может при определенных обстоятельствах оказаться полезной.



Для создания дочернего окна следует добавить в проект новую форму, выбрав Windows Form из диалогового окна, которое открывается при выборе пункта меню Project| Add New Item.



Эта форма становится дочерним окном, когда его свойству MdiParent присваивается ссылка на основное окно. Этому свойству нельзя присваивать значение с помощью панели Properties, это необходимо выполнять программным путем.

```
using System;
using System.Windows.Forms;

namespace MDIExample
{
    public partial class MainMDIContainerForm : Form
    {
        public MainMDIContainerForm()
        {
            InitializeComponent();
        }

        private void tsmiNew_Click(object sender, EventArgs e)
        {
            //Create a new instance of the MDI child template form
            ContentForm cf = new ContentForm();
```

```
                //Set parent form for the child window
                cf.MdiParent = this;

                //Display the child window
                cf.Show();
            }
        }
}
```

До того, как появится возможность выводить MDI-приложение на экран в его основном виде, нужно выполнить две вещи. Необходимо передать MDI-контейнеру информацию о том, какие окна должны выводиться, а затем вывести их, для чего следует просто создать новый экземпляр формы, которую вы собираетесь выводить, а затем вызвать для нее метод Show(). Конструктор формы, предназначенной для вывода в качестве дочернего окна, должен привязаться к родительскому контейнеру. Это достигается за счет присваивания его свойства MdiParent экземпляру MDI-контейнера.

Для манипуляций с дочерними окнами используется метод главной формы `LayoutMdi`

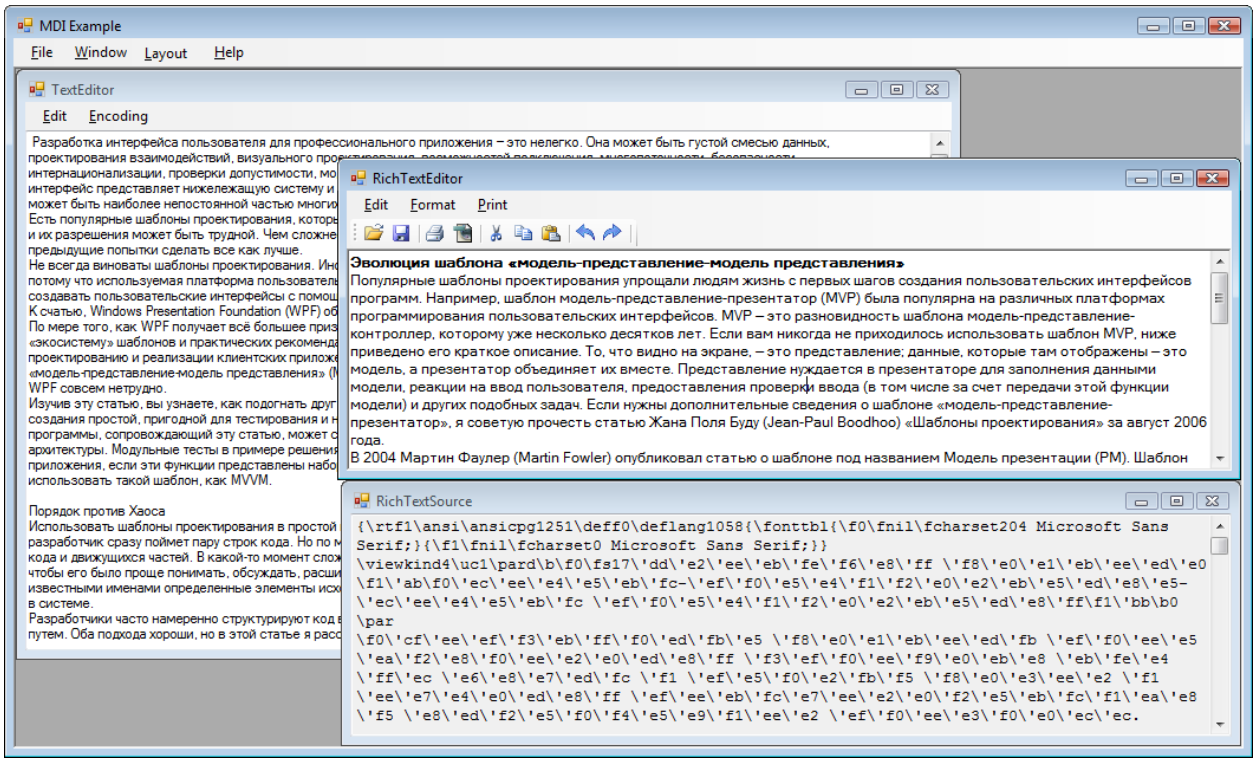
```
    // Расположить окна каскадом
    private void tsmiCascade_Click(object sender, EventArgs e)
    {
        this.LayoutMdi(System.Windows.Forms.MdiLayout.Cascade);
    }

    // Упорядочить
    private void tsmiArrangeIcons_Click(object sender, EventArgs e)
    {
        this.LayoutMdi(System.Windows.Forms.MdiLayout.ArrangeIcons);
    }

    // Окна черепицей сверху вниз
    private void tsmiTileHorizontal_Click(object sender, EventArgs e)
    {
        this.LayoutMdi(System.Windows.Forms.MdiLayout.TileHorizontal);
    }

    // Окна черепицей слева направо
    private void tsmiTileVertical_Click(object sender, EventArgs e)
```

```
    {
        this.LayoutMdi(System.Windows.Forms.MdiLayout.TileVertical);
    }
```

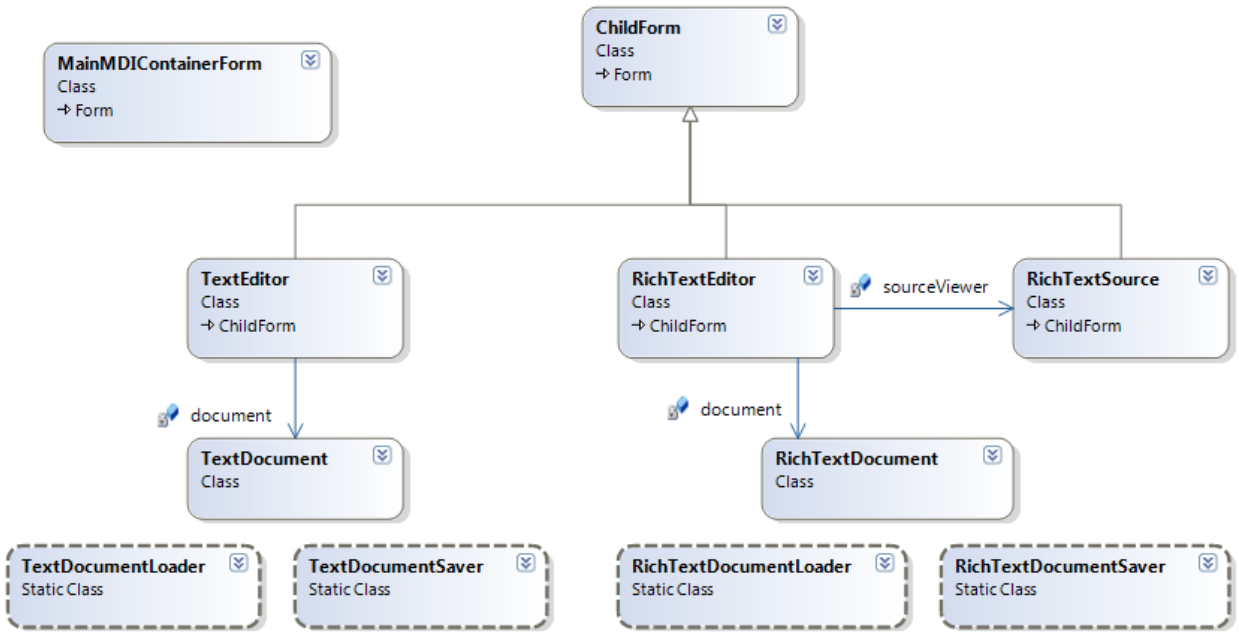
Максимизация и минимизация дочерних окон осуществляется получением всех дочерних окон и изменением их свойств:

```
    // Минимизировать все окна
    private void tsmiMinimazeAll_Click(object sender, EventArgs e)
    {
        // получаем все дочерние формы
        Form[] forms = this.MdiChildren;

        // каждое дочернее окно минимизируем
        foreach (Form cf in forms) cf.WindowState = FormWindowState.Minimized;
    }

    // Максимизируем все окна
    private void tsmiMaximizeAll_Click(object sender, EventArgs e)
    {
        Form[] forms = this.MdiChildren;
        foreach (Form cf in forms) cf.WindowState = FormWindowState.Maximized;
    }
```

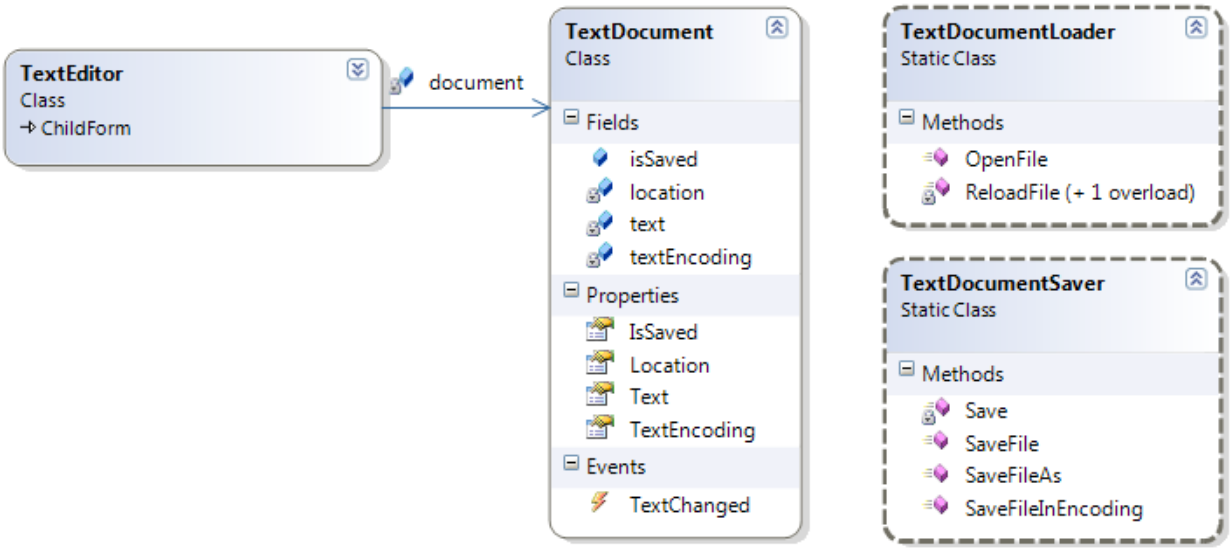
Рассмотрим подробнее приложение MDI Application Example, позволяющее редактировать текстовые файлы и файлы формата обогащенного текста (Rich Text). Задача этого примера не реализовать полностью функционал серьезных редакторов, а обратить внимание на некоторые архитектурные и дизайнерские особенности разработки MDI-приложений.



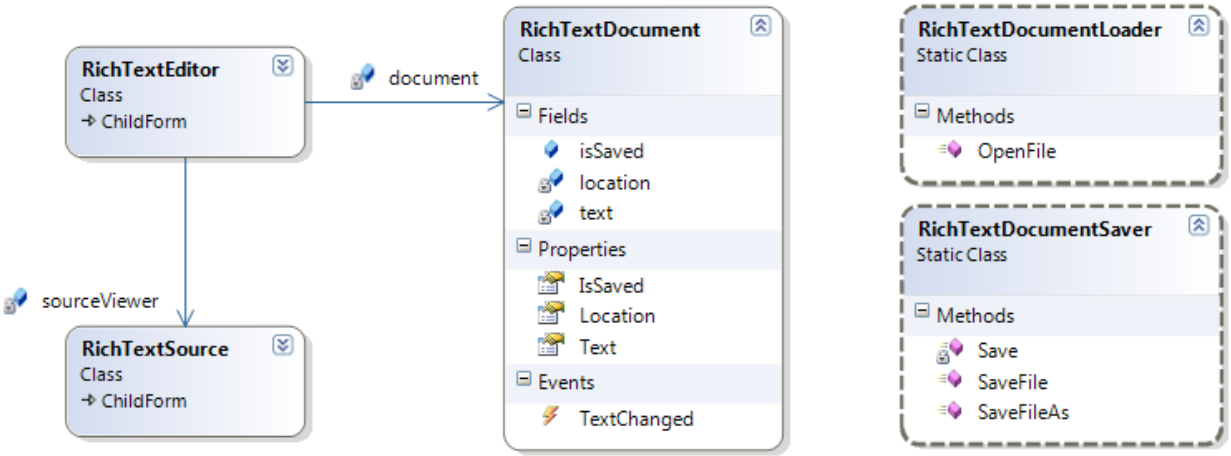
Данное приложение состоит из главной формы MainMDIContainerForm, формы для работы с текстовыми файлами TextEditor, и двух форм для работы с «обогащенным текстом»: RichTextEditor для редактирования текста и RichTextSource для отображения исходного кода формата. Общая архитектура представлена на следующей диаграмме классов:



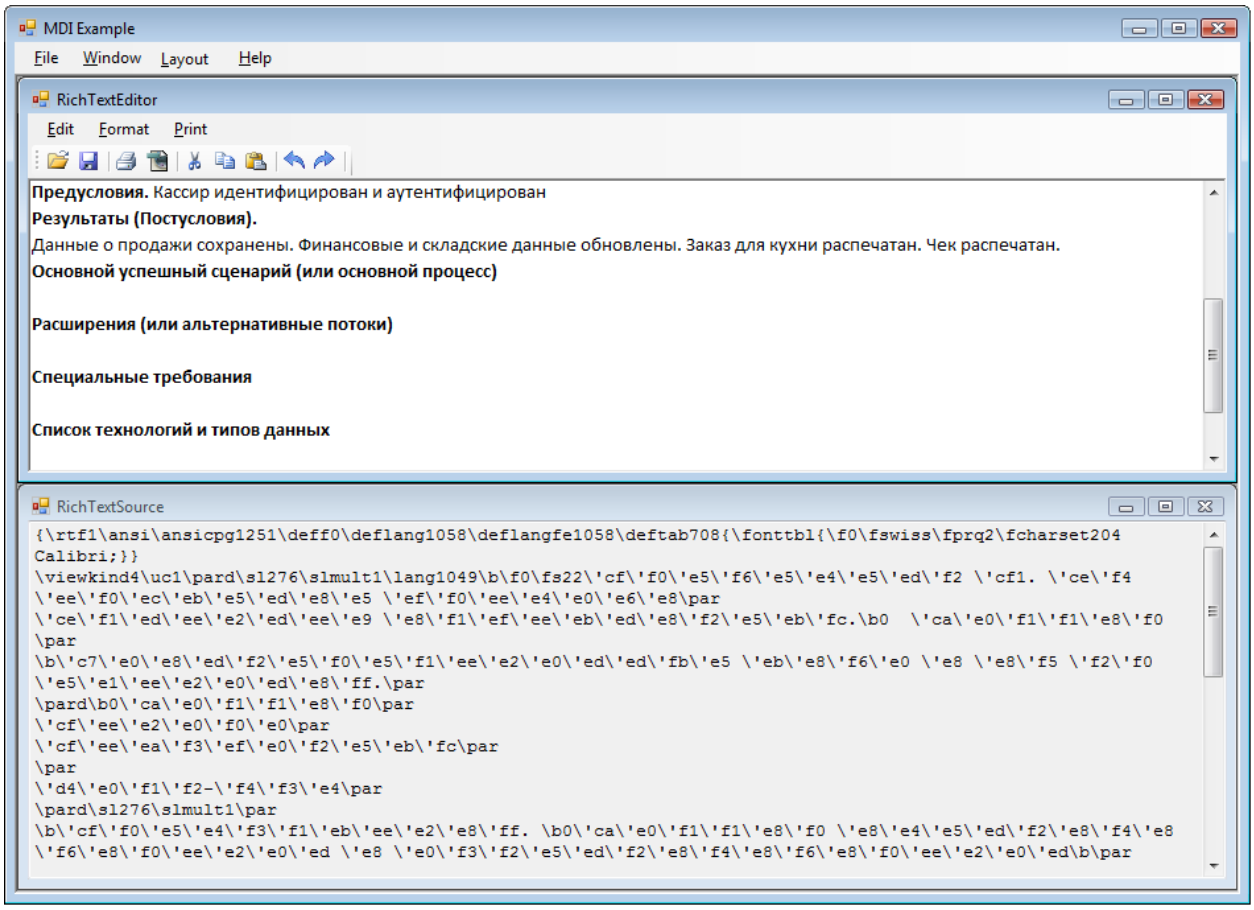
Функционал модуля работы с текстовыми файлами осуществляется четырьмя классами: `TextDocument` – описывает класс текстового документа; статический класс `TextDocumentLoader` – реализующий открытие и загрузку данных; статический класс `TextDocumentSaver` – реализующий сохранение и класс формы `TextEditor` выполняющий роль представления документа и контроллера. Следующая диаграмма иллюстрирует более подробно данный модуль приложения:



Аналогичную архитектуру имеет модуль работы с rtf-документами:



Основным отличием, которого, является наличие дополнительной формы представления `RichTextSource` отображающей исходный код rtf-документа:



Для детального рассмотрения работы MDI-приложения рекомендуется ознакомиться с исходным кодом данного примера.

Выводы

Вопрос, какой тип интерфейса предпочтителен - MDI или SDI - часто становится предметом обсуждений в сообществе разработчиков и пользователей программного обеспечения. Очевидно, что SDI более удобен при работе с несколькими приложениями разных типов. Разработчики широко используют оба типа интерфейса, а зачастую и интерфейс смешанного типа. Например, Microsoft меняла интерфейс Microsoft Office от SDI к MDI, а потом вернулась обратно к SDI, хотя степень реализации включает и первое, и второе.

Среди недостатков MDI часто указывали отсутствие наглядной информации об открытых окнах, для просмотра текущего списка открытых окон в приложении пользователю было необходимо выбрать в меню пункт «открытые окна/window list», или подобный ему. В последнее время в приложениях стали появляться панели задач и вкладки для отображения открытых окон в MDI. Такой тип интерфейса называют: «Tabbed document

interface» (TDI), хотя фактически это разновидность MDI, после распространения, которой критики заметно поубавилось.

Экзаменационное задание

Разработать программу для генерации и печати кроссвордов на основе заданной пользователем конфигурации.

Входные данные:

База знаний слов и заданий (пара слово - задание)

К примеру,

Лисица [хищное млекопитающее семейства псовых]

Слон [млекопитающее отряда хоботных]

База знаний конфигураций кроссвордов

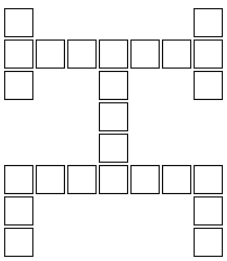
Каждая конфигурация задается следующими параметрами:

- *W* (максимальная ширина в миллиметрах, которую может занимать кроссворд)
- *H* (максимальная длина в миллиметрах, которую может занимать кроссворд)
- *X* (координата по оси абсцисс, которая указывает расположение слова в сетке кроссворда)
- *Y* (координата по оси ординат, которая указывает расположение слова в сетке кроссворда)
- *L* (длинна слова)
- *O[V||H]* (направление слова *V* - вертикальное, *H*-горизонтальное)

К примеру, конфигурация

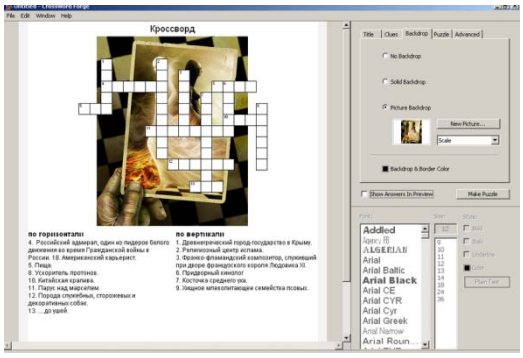
650 80, 0 0 3 V, 0 1 7 H, 6 0 3 V, 3 1 5 V, 0 5 3 V, 0 5 7 H, 6 5 3 V

является представлением кроссворда, изображенном на рисунке ниже.



Выходные данные:

Графическое представление "чистого" и "отгаданного" кроссворда со списком вопросов. Пример предполагаемого интерфейса программы приведен ниже.



Кроме того, при разработке программы необходимо учесть, что пользователь должен иметь возможность:

- создавать и редактировать, указанные выше базы знаний, через интерфейс программы
- изменять язык интерфейса программы
- изменять внешний вид кроссворда (цвет, шрифт, фоновый рисунок)
- печатать кроссворд

Большими плюсами при создании программы также являются:

- реализация режима “игры”, при котором пользователь имеет возможность отгадывать кроссворд и смотреть статистику других игроков
- взаимодействие с СУБД для хранения заданий и конфигураций
- каталогизация слов для создания тематических кроссвордов
- пополнение баз заданий из внешних источников
- возможность изменения "скина" программы