Построение расширяемого программного комплекса

Д. С. Порай, Т. А. Порай, А. В. Соловьев

В практике разработки корпоративных информационных систем часто встречаются задачи настройки готовых программных комплексов на нужды конкретной организации. Главные части процесса настройки — выделение из большой системы существующей функциональности в объеме, который необходим заказчику, и ее дополнение модулями, решающими специфические задачи. В данной статье описан подход, позволяющий компоновать систему из бинарных модулей (готовых и вновь разработанных) и объединять их в единый пользовательский интерфейс. При этом процесс компоновки происходит без необходимости перекомпиляции этих модулей из исходных текстов.

1. Введение

В области разработки программного обеспечения сформировалось направление программных комплексов, настраиваемых на нужды организации. Такие программные комплексы представляют собой корпоративные информационные системы (далее — Системы), состоящие из большого числа модулей, решающих задачи разных предметных областей. Установка такой Системы в конкретную организацию требует решения следующих задач:

1. Выделение из всей Системы набора модулей, соответствующих задачам данной организации.
2. Разработка дополнительных модулей, соответствующих специфическим задачам данной организации.
3. Интеграция получившегося набора модулей между собой.
4. Объединение их в единый пользовательский интерфейс.

На практике часто оказывается, что задача выделения части модулей, задача интеграции и, особенно, задача последующего объединения в единый пользовательский интерфейс являются крайне сложными и дорогостоящими. Решением этой проблемы может быть только изначальное про-

ектирование Системы таким образом, чтобы она состояла из относительно независимых модулей, допускающих последующую компоновку. Требования к архитектуре такой Системы можно сформулировать следующим образом:

1. Система должна состоять из набора модулей. Зависимость между модулями допускается, но она должна быть минимизирована.
2. Настройка Системы на конкретного заказчика должна выполняться компоновкой бинарных модулей без необходимости их перекомпиляции.
3. Пользовательский интерфейс должен быть настраиваемым: состав главного меню, названия отдельных пунктов, их порядок, состав локальных меню, набор колонок в отображаемых таблицах и т. д.
4. Настройка пользовательского интерфейса должна выполняться без необходимости перекомпиляции модулей.

В рамках данной работы была создана технология построения расширяемых Систем, удовлетворяющих всем перечисленным требованиям. Далее будут описаны основные принципы этой технологии. Эти принципы были проверены на практике в процессе создания «Электронного Архива». Созданная система обладает следующими возможностями:

- Является распределенной. Учитывается типичная иерархическая структура организации с территориально-распределенными подразделениями.
- Осуществляет ввод бумажных документов и файлов, заверенный электронно-цифровой подписью (ЭЦП). При этом ввод документов осуществляется в подразделениях нижнего уровня, данные затем передаются в вышестоящие архивы по каналам связи или на компакт-дисках.
- Использует роботизированные библиотеки CD/DVD дисков для хранения больших массивов отсканированных изображений и файлов с ЭЦП.
- Осуществляет поиск документов по реквизитам. При этом доступ к документам осуществляется в оперативном режиме независимо от того, находятся ли они в базе данных или вытеснены на CD/DVD диски.

- Имеет возможность распределенного поиска документов. При этом любой архив, имеющий доступ к каналам связи, может запросить документ из любого другого архива.
- Позволяет выводить статистическую информацию с помощью генератора отчетов.
- Контролирует сроки хранения документов с уничтожением документов с истекшим сроком хранения.

Внутреннее устройство этой системы будет использовано далее в качестве иллюстрации предлагаемой технологии.

Необходимо отметить, что в настоящее время существует технология, позволяющая решить похожую задачу компоновки Системы из набора модулей, но эта технология опирается на компоновку из исходных текстов с последующей их компиляцией [1]. В рамках этой технологии сформулировано несколько идей, которые использованы в данной работе. Это понятия «наборное гнездо» и «сменный модуль», а также правила работы с этими объектами.

2. Архитектура расширяемого программного комплекса

Расширяемый программный комплекс состоит из следующих частей (см. рис. 1):

- Набор функциональных компонент. Функциональные компоненты будут подробно рассмотрены далее.
- Ядро. Обеспечивает совместную работу функциональных компонент и связь пользовательского интерфейса с ними.
- Пользовательский интерфейс. Обеспечивает взаимодействие пользователя с Системой таким образом, чтобы он видел Систему как единое целое и не подозревал о структурном разделении функциональности на отдельные компоненты.

Некоторые из описанных частей располагаются на клиенте, некоторые на сервере. Пользовательский интерфейс представляет собой «толстый клиент», что позволяет организовать взаимодействие с пользователем максимально удобным и продуктивным способом. Описываемая технология может быть расширена на область «тонких клиентов» без принципиальных изменений.

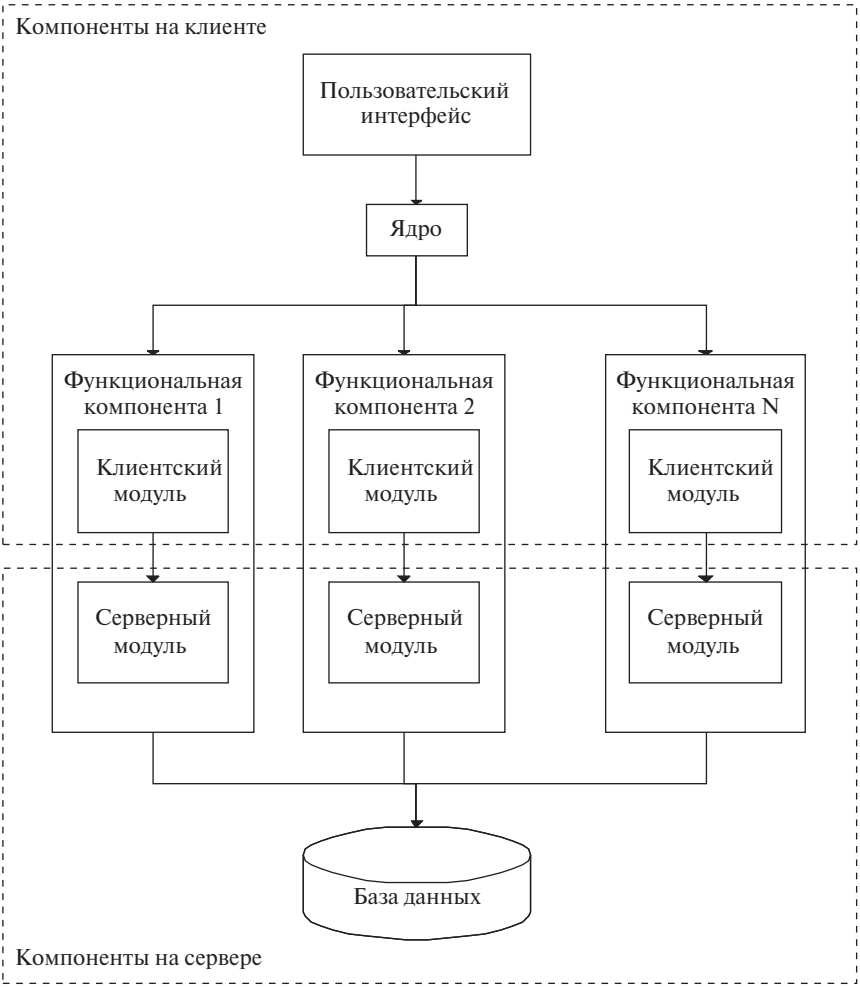

Рис. 1. Архитектура расширяемого программного комплекса

3. Функциональная компонента

Основным понятием расширяемого программного комплекса является понятие функциональной компоненты. Функциональная компо-

нента — это набор программных компонент и данных, обеспечивающих выполнение задач определенной предметной области. Функциональные компоненты могут зависеть друг от друга. Зависимая компонента называется дочерней. Другая компонента называется базовой. Зависимости образуют ациклический направленный граф. Пример такого графа зависимостей функциональных компонент приведен далее (см. рис. 2).

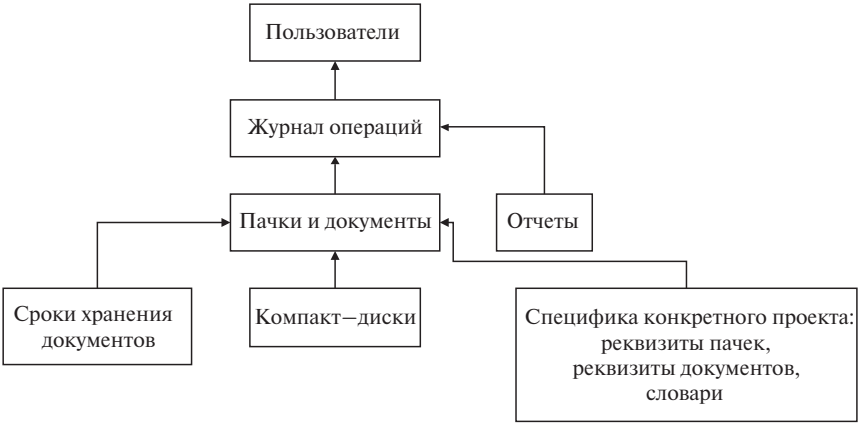

Рис. 2. Граф зависимостей функциональных компонент системы «Электронный Архив»

Каждая функциональная компонента состоит из нескольких частей:

1. Серверный модуль — набор объектов, располагающихся на сервере в базе данных:
   1.1. Таблицы. Могут содержать внешние ключи, ссылающиеся на таблицы базовых компонент.
   1.2. Индексы.
   1.3. Строки в таблицах, принадлежащих базовым компонентам.
   1.4. Хранимые процедуры. Могут использовать как свои таблицы, так и таблицы базовых компонент. Могут вызывать как свои хранимые процедуры, так и хранимые процедуры базовых компонент.
   1.5. Триггеры.

2. Клиентский модуль (DLL-файл). Содержит клиентскую функциональность и элементы пользовательского интерфейса. Может использовать свои серверные модули, а также клиентские и серверные модули, принадлежащие базовым компонентам.

4. Пользовательский интерфейс

4.1. Внешний вид

Расширяемый пользовательский интерфейс объединяет все функциональные компоненты Системы. Интерфейс выполнен в стиле Microsoft Outlook. Главное окно разделено на две части, слева расположено дерево категорий, справа — таблица, раскрывающая содержимое выбранной категории (см. рис. 3).

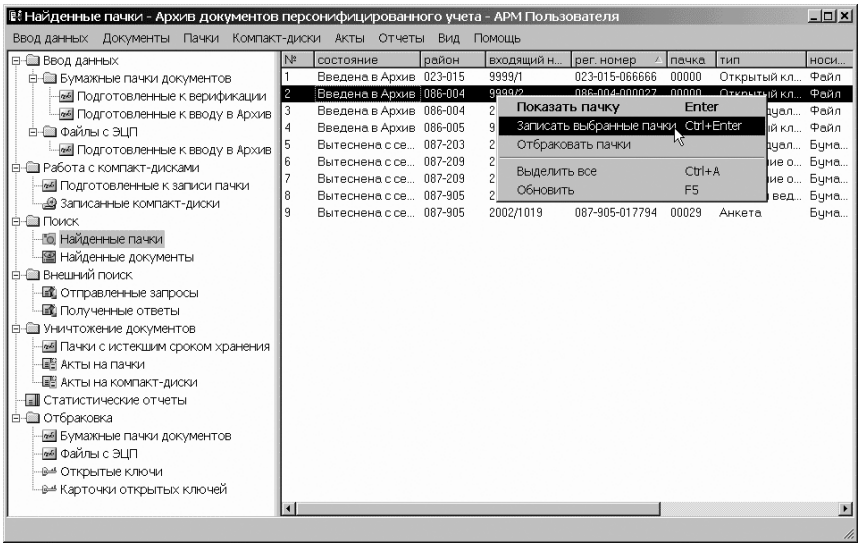

Рис. 3. Главное окно программы

Выбор пользователем строки в таблице в правой части главного окна приводит к отображению диалога со свойствами выбранного объекта (см. рис. 4) [2]. Таким объектом может быть документ, пачка документов, компакт-диск и т. д.

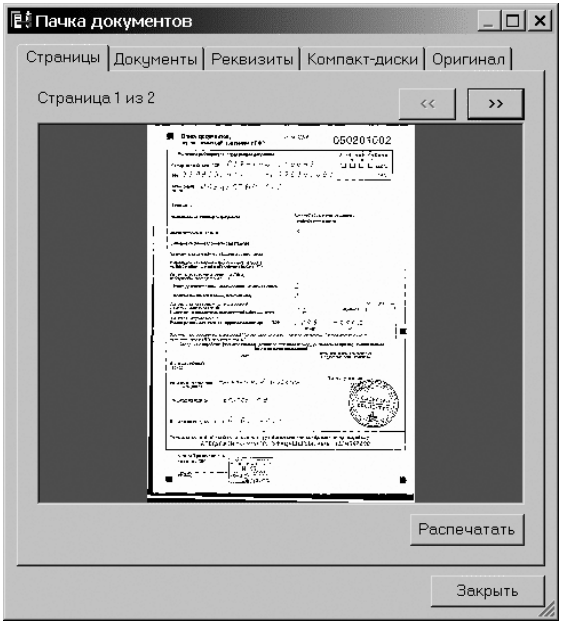

Рис. 4. Диалог свойств

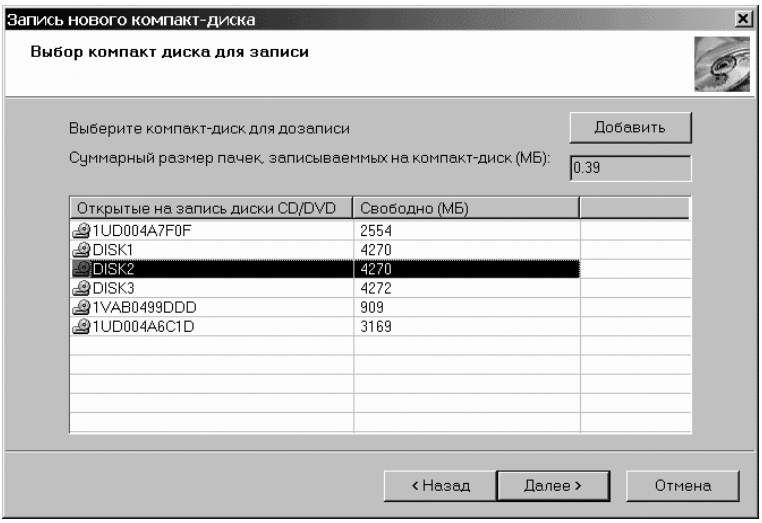

Рис. 5. Окно Мастера

Пользователь имеет возможность указать строку или несколько строк в таблице, после чего выбрать пункт главного или локального меню. Этим он инициирует выполнение операции над выбранными строками. Такой операцией может быть, например, запись выбранных пачек на компакт-диск. Каждая операция представляет собой Мастер (Wizard) в стандарте Microsoft Windows (см. рис. 5) [2].

4.2. Расширяемые элементы

Описанный интерфейс имеет следующие расширяемые элементы:

1. Пункты главного меню:
   1.1. Название пункта меню.
   1.2. Внутреннее название операции.
   1.3. Тип операции: требует в качестве аргумента выделенные строки из таблицы или не требует.
   1.4. Акселератор (сочетание клавиш для быстрого вызова операции).
   1.5. Название группы пользователей, имеющей право на выполнение данной операции.
   1.6. Название вершины, на которую нужно перейти после выполнения операции.
2. Вершины дерева:
   2.1. Название вершины дерева.
   2.2. Графическое изображение вершины дерева.
   2.3. Название родительской вершины.
   2.4. Название группы пользователей, имеющей право на работу с таблицей, соответствующей данной вершине дерева.
3. Таблицы, соответствующие вершинам дерева:
   3.1. Список колонок со следующими параметрами:
      3.1.1. Название.
      3.1.2. Тип.
      3.1.3. Ширина по умолчанию.
      3.1.4. Список названий вторичных колонок для сортировки по нескольким полям.
   3.2. Текст SQL-запроса для выборки содержимого таблицы из базы данных.

4. Локальные меню, соответствующие таблицам. Содержат набор пунктов, ссылающиеся на пункты главного меню.
5. Диалоги свойств:
   5.1. Заголовок диалога.
   5.2. Набор закладок.

Каждый расширяемый элемент представляет собой наборное гнездо. При запуске программы каждое такое наборное гнездо заполняется несколькими сменными модулями. Сменные модули располагаются в клиентских DLL файлах функциональных компонент Системы. Каждый DLL файл может содержать в себе несколько сменных модулей. Например, DLL компоненты «Пачки и документы» может содержать следующие сменные модули:

1. Пункт главного меню «Показать пачку».
2. Пункт главного меню «Найти пачки».
3. Вершина дерева «Найденные пачки».
4. Таблица, соответствующая вершине дерева «Найденные пачки».
5. Пункты локального меню, ссылающиеся на пункты главного меню «Показать пачку» и «Найти пачки».
6. Диалог «Свойства пачки» с закладками «Изображения страниц пачки», «Реквизиты пачки».

Сменный модуль может содержать наборное гнездо. Таким образом обеспечивается неограниченная вложенность элементов пользовательского интерфейса. Например, сменный модуль «Диалог свойств пачки документов» содержит наборное гнездо «Набор закладок диалога свойств пачки». Это наборное гнездо заполняется следующими сменными модулями:

1. «Изображения страниц пачки» из компоненты «Пачки и документы».
2. «Реквизиты пачки» из компоненты «Пачки и документы».
3. «Список компакт-дисков» из компоненты «Компакт-диски».
4. «Информация об оригинале документа» из компоненты «Сроки хранения документов».

4.3. Сборка на этапе выполнения

Сборка пользовательского интерфейса из отдельных элементов происходит при запуске программы. Процесс осуществляется на основе информации, какие именно сменные модули должны быть включены в на-

борные гнезда и в какой последовательности. Эта информация собрана в одном .INI-файле, что позволяет собирать из одних и тех же бинарных модулей несколько инсталляторов, отличающихся функциональностью, сформулированной в .INI-файле.

Конфигурационный .INI-файл состоит из секций, соответствующих сменным модулям. Каждая такая секция содержит параметры данного модуля. Например, секции пункта главного меню «Найти пачки» и диалога «Свойства пачки» выглядят следующим образом:

```
[MenuItem_FolderFind]
parent = Folder
title = &Найти пачки
operation0 = FLD.FLDFIND
groups = SRCH
treeNodeAfterDone = FolderSearch
accelerator = Ctrl + F

[PropertySheet_FOLDER]
title=Пачка документов
pages=PFROCR.PFRIMG;PFR.PFRFORM;FLD.FLDFILE;FLD.FLDDOCS;
FLD.FLDFIELDS;CDD.FLDCDD;FLDDLT.FLDACT
resizable=1
maximized=1
```

5. Реализация

5.1. Клиентский модуль

Клиентский модуль функциональной компоненты содержит класс, реализующий интерфейс `CModule`, и экспортирует функцию `createModule`, возвращающую экземпляр этого класса:

```
extern "C" __declspec(dllexport) void
createModule(CModule** pModule)
{
    CFldModule* pFldModule = new CFldModule();
    *pModule = pFldModule;
}
```

Основные функции `CModule` выглядят следующим образом:

```
class CModule
{
    /// Запоминает указатель на экземпляр класса
    /// CModuleContainer. Этот указатель может быть
    /// использован впоследствии. Например, с его
    /// помощью можно получить указатель на объект
    /// ADODB::_Connection. Функцию можно
    /// переопределить в модуле
    /// и делать дополнительную инициализацию.
    virtual void setModuleContainer(CModuleContainer*
    pContainer);

    /// Возвращает указатель на ModuleContainer
    CModuleContainer* getModuleContainer();

    /// Исполняет безаргументную операцию с именем
    /// pOperName
    virtual void executeOperation0(const
    char* pOperName) = 0;

    /// Исполняет операцию с именем pOperName, имеющую
    /// один аргумент — указатель на Recordset
    virtual void executeOperation1(const
    char* pOperName,
    ADODB::_Recordset* pRecordSet) = 0;

    /// Загружает изображение из ресурсного файла,
    /// возвращает его идентификатор.
    virtual HBITMAP getBitmap(const
    char* pImageName) = 0;

    /// Создает страницу диалога свойств
    /// (PropertyPage)
    virtual CDynamicPropertyPage* getPropertyPage(
    CDynamicPropertySheet* pSheet, const
    char* pPropertyPageName) = 0;
};
```

Использованные в приведенном листинге интерфейсы CdynamicPropertySheet и CDynamicPropertyPage являются обертками над стандартными элементами диалога свойств Windows:

```
class CDynamicPropertySheet
{
    /// Добавляет закладку
    void addPage(CDynamicPropertyPage* pPage);

    /// Возвращает указатель на Recordset
    ADODB::_RecordsetPtr getRecordset();

    /// Открывает диалог
    void DoModal();
};
class CDynamicPropertyPage
{
    /// Следует ли показывать данную закладку?
    virtual bool isVisible() = 0;

    /// Возвращает идентификатор стандартной закладки
    /// диалога свойств
    virtual LPCPROPSHEETPAGE getPage() = 0;
};
```

При построении системы много внимания было уделено вопросам эффективности. Особенно интересны следующие два момента:

- Передача Recordset в качестве параметров. Традиционно схема вызова операций из пользовательского интерфейса выглядит следующим образом: пользователь выбирает объект интерфейса (например, документ), затем инициирует выполнение операции, после этого пользовательский интерфейс вызывает соответствующую функцию, передавая ей идентификатор выбранного объекта, далее вызванная функция загружает из базы данных необходимую информацию о данном объекте. Если пользователь выбрал сразу несколько объектов, то количество обращений к базе возрастает соответственно. В описываемой Системе был реализован другой подход, при котором в вызываемую функцию передается не идентификатор объекта, а объект типа Recordset, представляющий собой строку или несколько строк из базы данных. Каждая строка представляет собой набор полей, часть из которых отображается в таблице в интерфейсе, а часть являются невидимыми для пользователя (те же идентификаторы). Данных подход позволяет во многих случаях исключить обращения к базе данных для извлечения

информации об обрабатываемых объектах. А такие операции, как просмотр реквизитов документа, выполняются вообще без обращения к базе данных, т. е. практически мгновенно.

- Использование серверных курсоров. Второй момент связан с организацией интерфейса в стиле Outlook. Пользователь, выбирая вершину дерева, получает таблицу с информацией обо всех объектах (например, документах), которые необходимо обработать. Размер этой таблицы априори не известен и может оказаться равным нескольким десяткам, сотням или даже тысячам записей. Передача такого объема с сервера базы данных на клиент, и отображение в интерфейсе может занять значительное время. Решение данной проблемы связано с применением редко используемой технологии серверных курсоров. В этом случае запрос выполняется на сервере, а его результаты передаются на клиент порциями, равными количеству строк таблицы, помещающихся на экране. В ряде случаев СУБД способна не только передавать результаты порциями, но и выполнять запросы инкрементно, т. е. пока данные не отображаются в интерфейсе, СУБД даже не извлечет их из базы данных. Использование данной технологии позволяет существенно уменьшить время отклика Системы на действия пользователя.

5.2. Ядро Системы

Ядро Системы представляет собой класс `CModuleContainer`. При запуске программы создается единственный экземпляр этого класса. Он осуществляет загрузку клиентских модулей и переправляет им вызовы, пришедшие от пользовательского интерфейса. Основные функции класса `CModuleContainer` выглядят следующим образом:

```
class CModuleContainer
{
    /// Возвращает соединение с базой данных
    ADODB::_ConnectionPtr getConnection();

    /// Показывает PropertySheet со страницами,
    /// загружаемыми динамически из DLL модулей
    void displayPropertySheet(const
    char* propertySheetName,
    ADODB::_Recordset* pRecordset, HWND
    hWndParent = NULL);
```

```
    /// Исполняет безаргументную операцию с именем
    /// pOperName.
    /// Вызов переправляется в соответствующую DLL
    virtual void executeOperation0(const
    char* pOperName);

    /// Исполняет операцию с именем pOperName,
    /// имеющую один аргумент — указатель на Recordset.
    /// Вызов переправляется в соответствующую DLL
    virtual void executeOperation1(const
    char* pOperName,
    ADODB::_Recordset* pRecordSet);

    /// Загружает изображение из ресурсного файла,
    /// возвращает его идентификатор.
    /// Вызов переправляется в соответствующую DLL
    virtual HBITMAP getBitmap(const char* pImageName);
};
```

При этом в качестве названий операций, закладок диалогов свойств и т. д. передаются строки вида `FLD.DOCFIND`. В ответ на это ядро загружает модуль FLD.DLL и передает ему строку DOCFIND.

6. Заключение

В данной работе сформулирована задача построения расширяемых программных комплексов. Разработана технология, позволяющая создавать такие системы. Важной частью технологии является построение расширяемого пользовательского интерфейса. Эта часть детально проработана для построения Outlook-подобных пользовательских интерфейсов, которые являются в настоящее время стандартом де-факто в области корпоративных информационных систем. Описанная технология проверена на практике в процессе разработки программного комплекса «Электронный Архив».

Литература

1. *Горбунов-Посадов М. М.* Расширяемые программы. М.: Полиптих, 1999.
2. Official Guidelines for User Interface Developers and Designers — http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwue/html/welcome.asp