МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждения высшего образования

«Юго-Западный государственный университет»

(ЮЗГУ)

Кафедра информационной безопасности

УТВЕРЖДАЮ

Проректор по учебной работе

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.Г. Локтионова

«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г.

Аппаратные и программные средства построения распределенных систем

Методические рекомендации по выполнению лабораторной работы №1

для студентов направления подготовки магистратуры 10.04.01«Информационная безопасность»

Курск 2017

УДК 621.(076.1)

Составитель: А.Г. Спеваков

Рецензент

Кандидат технических наук, доцент кафедры «Информационная безопасность» И.В. Калуцкий

Аппаратные и программные средства построения распределенных систем [Текст] : методические рекомендации по выполнению лабораторной работы / Юго-Зап. гос. ун-т; сост.: А.Г. Спеваков. – Курск, 2017. – 13 с.:

Содержат сведения по вопросам построения распределенных систем. Указывается порядок выполнения лабораторной работы, правила, содержание отчета.

Методические указания соответствуют требованиям программы, утвержденной учебно-методическим объединением по специальности.

Предназначены для студентов направления подготовки магистратуры 10.04.01 «Информационная безопасность».

Текст печатается в авторской редакции

Подписано в печать . Формат 60х84 1/16.

Усл.печ. л. 1,57. Уч.-изд. л. 1,42. Тираж 100 экз. Заказ. Бесплатно.

Юго-Западный государственный университет.

305040, г.Курск, ул. 50 лет Октября, 94.

Цель работы: Познакомиться с общими принципами организации распределенных ИС. Получить представление о компонентной объектной модели (COM). Научиться использовать механизм автоматизации для обработки данных с использованием внешних приложений.

Указания к выполнению лабораторной работы

Классическая компьютерная информационная система включает в себя следующие обязательные элементы:

* Информационная база, которая является основным хранилищем данных ИС. В подавляющем большинстве случаев она представляет собой *серверную базу данных*, которая управляется некоторой СУБД. Иные варианты (например, совместно используемый набор файлов) встречаются значительно реже.
* Клиентские приложения, которые используются для работы с данными и устанавливаются на ЭВМ пользователей информационной системы. Клиентские приложения не только выполняют просмотр и редактирование данных (так называемые *презентационные функции* информационной системы), но и обеспечивают решение более сложных задач: контроль корректности вводимой информации, статистическая обработка данных, формирование отчетов и т.п. (*функции ИС по прикладной обработке данных*).
* Средства обеспечения доступности данных информационной базы из клиентских приложений. К таким средствам, прежде всего, относятся средства сетевого доступа, которые обычно реализованы соответствующими средствами сетевых операционных систем (например, поддержка протоколов TCP/IP). Кроме того, сюда следует отнести средства взаимодействия пользовательского приложения и сервера баз данных. Некоторые из них были изучены в ходе выполнения предыдущих лабораторных работ (процессор баз данных BDE, технологии ODBC и ADO и т.п.).

При эксплуатации описанных информационных систем нередко возникают следующие проблемы:

* Необходимость использования ЭВМ с высокими требованиями к аппаратной части при организации рабочих мест пользователей. Клиентские приложения, которые реализуют не только презентационные, но и прикладные функции ИС, обычно требуют для своей работы довольно значительных вычислительных ресурсов. Кроме того, дополнительные ресурсы необходимы для установки и использования средств обеспечения доступа к данным.
* Необходимость приобретения (лицензирования) и установки специальных средств доступа к данным (например, компонентов BDE или поставщиков данных OLE DB) на каждом компьютере пользователя ИС.
* Сложность конфигурирования средств доступа к данным (например, организация BDE-псевдонимов или настройка источников данных ODBC на всех пользовательских ЭВМ).
* Возможные конфликты между средствами доступа к данным, установленным для работы с различными информационными системами.

Указанные проблемы могут существенно повысить стоимость эксплуатации ИС и значительно уменьшить (или вовсе устранить) экономический эффект от ее внедрения. Для преодоления этих проблем в последние годы все чаще рекомендуется проектировать сложные информационные системы на базе многоуровневой (многозвенной) архитектуры.

Рассмотренная классическая структура информационной системы соответствует двухуровневой модели клиент-сервер. Клиентами в данном случае являются приложения, установленные на ЭВМ пользователей, а сервером – очевидно, сервер базы данных (СУБД либо сетевая ОС – если база данных состоит из множества совместно используемых файлов). Клиенты напрямую обращаются к серверу (посредством служебных средств обеспечения доступа к данным) и единственной информационной услугой, которую они запрашивают, являются хранящиеся в информационной базе данные. Большая часть прикладной обработки данных выполняется клиентским приложением самостоятельно.

Использование трехуровневой архитектуры ИС позволяет разделить функции представления и прикладной обработки данных, а также упростить и унифицировать доступ к данным из клиентских приложений. Это достигается путем выделения части функций информационной системы в отдельный промежуточный слой. В этом случае структура ИС будет состоять не из трех, а из четырех элементов:

* Данные по-прежнему хранятся в информационной базе, в роли которой чаще всего выступает серверная БД.
* Функции прикладной обработки данных сосредоточены в так называемом *сервере приложений* (*Application Server*), который предоставляет клиентским пользовательским приложениям множество информационных услуг (сервисов) высокого уровня. Серверы приложений выступают в качестве промежуточного уровня между серверами данных и клиентскими приложениями. С одной стороны, они взаимодействуют с информационной базой ИС, запрашивая необходимые данные и отсылая запросы на модификацию данных. В этом случае они выступают в качестве клиентов. С другой стороны, они взаимодействуют с клиентскими пользовательскими приложениями, выдавая по их запросам результаты обработки данных или транслируя полученные запросы на модификацию данных в информационную базу. В этом случае они выступают в качестве серверов. Серверы приложений являются объектами высокого уровня, поэтому их реализация может не зависеть от специфики конкретной операционной системы или СУБД. Это обеспечивает целый ряд важных достоинств многоуровневой архитектуры ИС, в том числе:
  + Возможность подключения к серверу приложений клиентских приложений, работающих под управлением различных ОС.
  + Возможность простого и единообразного подключения клиентских приложений. Для их взаимодействия с сервером приложений может быть использована простая, но достаточно эффективная технология, не требующая установки сложного программного обеспечения на пользовательские ЭВМ. Все средства доступа к данным информационной базы (менеджеры драйверов, поставщик данных для конкретной СУБД, библиотеки, реализующие прикладной программный интерфейс и т. п.) в этом случае сосредоточены на уровне сервера приложений, а не на уровне клиентов.
* Клиентские приложения реализуют только *презентационные функции* ИС (просмотр и редактирования данных). Это соответствует концепции так называемого «тонкого» клиента, предъявляющего минимальные требования к аппаратной части пользовательских ЭВМ.
* Средства обеспечения доступности данных теперь распределены между клиентами и серверами приложений. Средства сетевого взаимодействия по-прежнему необходимы на всех уровнях, однако средства взаимодействия с базами данных необходимы только на уровне серверов приложений (см. выше).

Развитием трехуровневой архитектуры является так называемая *многоуровневая* (*n-уровневая*) *организация вычислений* (*Multi-tier computing*), когда информационная система состоит из большого количества удаленных друг от друга объектов (серверов), каждый из которых может предоставлять другим объектам (клиентам) разнообразные информационные услуги. При этом различные серверы могут работать под управлением различных ОС или использовать для хранения данных СУБД различных типов. Для синхронизированного и согласованного функционирования таких объектов в состав информационной системы также включают специальные служебные сервисы – системы мониторинга и управления распределенными транзакциями, агенты запуска и управления серверами и др.

В настоящее время наиболее популярны следующие технологии организации распределенных информационных систем:

* Технология COM и ее развитие (DCOM, COM+);
* Технология MIDAS (на базе компонентной объектной модели COM);
* Технология CORBA;
* Платформа .NET.

Сокращение COM означает Component Object Model, что можно перевести с английского языка как компонентная объектная модель. Технология была разработана фирмой Microsoft в начале 90-ых годов прошлого века для обеспечения взаимодействия между различными приложениями, запущенными на одном или даже на разных компьютерах1.

Сущность технологии COM заключается в программировании с использованием *компонентов* – подход, который знаком всем программистам, использующим в своей работе среду разработки Delphi или C++ Builder. Под компонентом в данном случае понимается законченный (и откомпилированный) объект со своими свойствами и методами, который может легко встраиваться в различные приложения и распространяться как отдельный продукт. Компоненты, созданные в соответствии со спецификацией COM, могут функционировать в различной языковой и операционной средах. Это значит, что если разработчик оформил некоторый набор функций как объект COM, то функциями этого объекта могут воспользоваться программисты самых разных языков программирования: C++, Delphi, Visual Basic и т. д. – достаточно, чтобы соответствующая среда разработки поддерживала технологию COM. По этой причине модель COM может являться базовой для создания распределенных информационных систем – составляющие ее объекты могут быть реализованы с использованием различных технологий и инструментов программирования, однако их взаимодействие может осуществляться в соответствии со спецификацией COM (посредством определенных интерфейсов и протоколов).

Принципы обращения к свойствам и методам COM-объекта из других приложений полностью соответствуют модели клиент-сервер. Компонент, в котором реализованы некоторые полезные свойства и методы, выступает в качестве COM-сервера, а обращающиеся к нему приложения – в качестве клиентов. Одно и то же приложение, очевидно, может выступать и в качестве клиента, и в качестве COM-сервера, в зависимости от характера взаимодействия с другими приложениями в каждый конкретный момент времени. Чтобы взаимодействие «клиент-сервер» между произвольной парой приложений состоялось успешно, необходимо выполнение следующих условий:

* Приложение-клиент должно «иметь представление» о тех сервисах (полезных свойствах и методах), которые предоставляет приложение-сервер. Если бы речь шла о приложениях, написанных на одном языке программирования, то для решения этой проблемы достаточно было бы распространять вместе с компонентом описания соответствующих ему классов. Например, для компонента, написанного на C++, описания его классов хранятся в заголовочных файлах с расширением “*.h*”. Поскольку технология COM является не зависимой от языка программирования, то для определения сервисных функций COM-объекта используется специальный язык – *язык описания интерфейса* (*IDL*, *Interface Description Language*2).
* Все COM-объекты должны иметь уникальные имена, по которым их можно отличать друг от друга и обращаться к ним. При этом уникальность должна носить глобальный характер, поскольку одни и те же COM-объекты могут использоваться десятками тысяч разработчиков в сотнях тысяч различных приложений. Для именования объектов COM используются так называемые глобальные уникальные идентификаторы (*Globally unique identifier*, *GUID*3). Глобальный уникальный идентификатор представляет собой 128-разрядное число, которое генерируется с использованием алгоритмов получения случайных чисел и специальной хеш-функции. Вероятность повторения глобального уникального идентификатора такой разрядности чрезвычайно мала. Таким образом, для того чтобы все COM-серверы были поименованы уникальными именами, разработчикам достаточно использовать один и тот же заранее определенный алгоритм генерации глобальных уникальных идентификаторов. В ОС Windows для генерирования GUID используются функции API **UuidCreate** и **CoCreateGuid**. Глобальный уникальный идентификатор принято записывать в следующем формате:

XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX,

например:

B502D1BE-9A57-11d0-8FDE-00C04FD9189D

* Операционная система должна хранить список доступных COM-объектов и информацию о месте их хранения (то есть путь к соответствующим исполняемым модулям, в которых реализованы сервисы этих объектов). В ОС Windows для этих целей используется специальный подраздел системного реестра: HKEY\_CLASSES\_ROOT\CLSID\, у которого в качестве вложенных подразделов выступают имена (глобальные уникальные идентификаторы) зарегистрированных в системе объектов, например:

{00000000-0E4D-0463-87B5-D411BEBE0010}

{00000001-4FEF-40D3-B3FA-E0531B897F98}

{00000010-0000-0010-8000-00AA006D2EA4}

{00000100-0000-0010-8000-00AA006D2EA4}

Каждый из указанных разделов содержит описание свойств соответствующего COM-объекта. Например, свойство **ProgID** содержит программный идентификатор объекта – кодовое наименование, которое присваивается разработчиком и которое, в отличие от глобального уникального идентификатора, несет некоторую смысловую нагрузку. Чтобы задать местонахождение исполняемого модуля для объекта COM используются свойства **InprocServer32** (если реализация объекта располагается в DLL-файле), **LocalServer32** (если объект реализован как EXE-файл) или **RemoteServer32** (для объектов, расположенных на удаленных ЭВМ в компьютерной сети).

* Операционная система, под управлением которой работает COM-сервер, должна иметь возможность единообразного получения адреса зарегистрированного COM-объекта, чтобы передавать соответствующую ссылку запросившим ее клиентам. Кроме того, для взаимодействия клиента с удаленными COM-серверами необходим специальный протокол сетевого взаимодействия, реализующий, в том числе, правила аутентификации и авторизации.

На основании указанного выше можно сделать вывод, что создание полноценных COM-объектов – это весьма сложная задача, требующая от разработчика глубоких знаний в области системного и объектно-ориентированного программирования4.

Среда разработки Borland C++ Builder предоставляют программисту возможность не только создавать собственные, но и эффективно использовать уже существующие COM-объекты в своих приложениях. В некоторых случаях взаимодействие создаваемого приложения с объектами COM реализуется незаметно («прозрачно») для разработчика, в других – требует от него осознанных действий.

Полезным примером практического и осознанного использования технологии COM является использование *механизма автоматизации* для взаимодействия с внешними приложениями. Механизм автоматизации (*Automation*5) позволяет разработчикам приложения привлекать для обработки данных функциональные возможности других приложений. Например, для формирования отчетов часто используются возможности Microsoft Excel или Microsoft Word.

Взаимодействие в данном случае осуществляется между *объектами* и *контроллерами* автоматизации. Объект автоматизации – это объект, созданный по технологии COM, в котором реализован доступ к полезным свойствам и методам некоторого приложения или его части. Контроллер автоматизации – это приложение, которое имеет доступ к свойствам и методам COM-объекта и использует их для обработки своих данных. В указанном ранее примере, приложение разработчика – это контроллер, а Microsoft Excel либо Microsoft Word – объекты автоматизации.

Рассмотрим общую схему работы контроллера автоматизации при взаимодействии с каким-либо приложением Microsoft Office:

1. Проверить, запущена ли копия приложения-сервера (объекта автоматизации).
2. В зависимости от результатов проверки (либо исходя из конкретной ситуации) запустить копию приложения-сервера либо подключиться к уже имеющейся копии.
3. Если необходимо, сделать окно приложения-сервера видимым.
4. Выполнить какие-то действия с приложением-сервером (например, создать или открыть документы, изменить их данные, сохранить документы и т. п.).
5. Закрыть приложение-сервер, если его копия была запущена данным контроллером, или отключиться от него, если контроллер подключился к уже имеющейся копии (или если работа с приложением будет продолжена).

Ниже приведен программный код для указанной схемы (в качестве объекта автоматизации выбрано приложение Microsoft Excel, которое запускается по нажатию на кнопку Button1 и закрывается по нажатию кнопки Button2):

*#include <comobj.hpp>*

*...*

*TForm1 \*Form1;*

*// глобальные переменные модуля*

*Variant Server;*

*bool ServerIsRunning;*

*...*

*void \_\_fastcall TForm1::Button1Click(TObject \*Sender)*

*{*

*AnsiString ServerProgID;*

IUnknown \*UnknownInterface;

HResult Res;

*// Свойство ProgID для приложения Microsoft Excel = "Excel.Application"*

ServerProgID = "Excel.Application";

*// Проверяем наличие запущенной копии приложения*

Res = GetActiveObject(ProgIDToClassID(ServerProgID),0,&UnknownInterface);

**if** SUCCEEDED(Res)

{

*// подключаемся к существующему экземпляру приложения-сервера*

Server = GetActiveOleObject(ServerProgID);

ServerIsRunning = **true**;

}

**else**

{

*// запускаем новый экземпляр приложения-сервера*

Server = CreateOleObject(ServerProgID);

ServerIsRunning = **false**;

}

*// Показываем окно приложения-сервера на экране*

Server.OlePropertySet("Visible", **true**);

**try**

{

//------------------------------------------

// *Здесь выполняются некоторые действия*

// *с объектами приложения Office*

// ------------------------------------------

}

**catch(**...**)**

{

// Сообщение об ошибке работы с MS Excel

}

Button1->Enabled = **false**;

Button2->Enabled = **true**;

}

//-------------------------------------------------------------

**void** \_\_**fastcall** TForm1::Button2Click(TObject \*Sender)

{

// отключаемся либо закрываем приложение-сервер

**if** (ServerIsRunning)

Server= Unassigned;

**else**

Server.OleProcedure("Quit");

Button2->Enabled = **false**;

Button1->Enabled = **true**;

}

Для обращения к свойствам и методам объектов автоматизации можно использовать следующие методы класса Variant:

* Variant OlePropertyGet(const String& name, TAutoArgsBase\* args = 0) – получить значение свойства *name* некоторого COM-объекта. В качестве второго параметра метода обычно передаются дополнительные аргументы;
* **void** OlePropertySet(const String& name, TAutoArgsBase& args) – установить значение свойства *name* некоторого COM-объекта. В качестве второго параметра метода обычно передается устанавливаемое значение;
* **void** OleProcedure (const String& name, TAutoArgsBase\* args = 0) – запустить метод (процедуру) с именем *name* для некоторого COM-объекта. В качестве второго параметра метода обычно передаются дополнительные параметры процедуры;
* Variant OleFunction (const String& name, TAutoArgsBase\* args = 0) – запустить метод (функцию) с именем *name* для некоторого COM-объекта. В качестве второго параметра метода обычно передаются дополнительные параметры функции;

Используя перечисленные методы, можно обращаться к тем объектам приложения Microsoft Excel (Workbooks, WorkSheets, Cells и т. п.), которые изучались в лабораторной работе № 2 курса «Информационные системы».

**Примеры**:

1. Открываем новую рабочую книгу

Server.OlePropertyGet("WorkBooks").OleProcedure("add");

1. Устанавливаем ширину колонки “B” равной 40 на первом рабочем листе:

Variant Sheet1 = Server.OlePropertyGet("WorkSheets",1);

Variant ColumnB =

Sheet1.OlePropertyGet("Cells",1,2).OlePropertyGet("EntireColumn");

ColumnB.OlePropertySet("ColumnWidth",40);

1. Устанавливаем значение ячейки c RC-адресом (Row, Column) равным значению переменной (или Variant-объекта) Data:

Sheet1.OlePropertyGet("Cells",Row,Column).OlePropertySet("Value",Data);

1. Устанавливаем жирный шрифт для ячейки c RC-адресом (Row, Column)

Sheet1.OlePropertyGet("Cells",Row,Column).

OlePropertyGet("Font").OlePropertySet("Bold",**true**);

1. Закрываем последнюю открытую рабочую книгу. Если в нее были внесены изменения, то пользователь будет предупрежден об их возможной потере.

Variant WorkBook = Server.OlePropertyGet("WorkBooks",

Server.OlePropertyGet("WorkBooks").OlePropertyGet("Count"));

WorkBook.OleProcedure("close");

1. Закрываем последнюю открытую рабочую книгу без сохранения изменений и без предупреждения пользователя.

Variant WorkBook = Server.OlePropertyGet("WorkBooks",

Server.OlePropertyGet("WorkBooks").OlePropertyGet("Count"));

WorkBook.OleProcedure("close",**false**);

Задания к лабораторной работе

1. Исследовать перечень COM-объектов, зарегистрированных на компьютере, на котором выполняется лабораторная работа. Для этого запустить редактор системного реестра (это можно, сделать, например, через команду **Выполнить** кнопки **Пуск**; имя приложения редактора реестра – *regedit.exe*). В редакторе реестра найти раздел HKEY\_CLASSES\_ROOT и подраздел CLSID. В отчет о выполнении лабораторной работы включить рисунок с частью системного реестра, отображающий перечень нескольких первых объектов COM.
2. Найти в списке зарегистрированных COM-объектов объекты со следующими глобальными уникальными идентификаторами:

{00000011-0000-0010-8000-00AA006D2EA4}

{00000100-0000-0010-8000-00AA006D2EA4}

{00024500-0000-0000-C000-000000000046}

В отчете о выполнении лабораторной работы указать, какие из указанных объектов были обнаружены. Для найденных объектов привести значения свойств **ProgID** и **InprocServer32**/**LocalServer32**/**RemoteServer32**.

1. С использованием механизма автоматизации доработать БД-приложение из лабораторной работы № 5 таким образом, чтобы у пользователя была возможность экспортировать в Microsoft Excel данные из основной таблицы приложения. Обратить внимание на удобство восприятия экспортированных данных (предусмотреть выделение заголовков столбцов, подбор ширины ячеек и т.п.).
2. В отчете отразить ход выполнения работы, результаты тестирования и сделанные выводы.

Контрольные вопросы и задания

1. Двухзвенная архитектура ИС.
2. Достоинства и недостатки двухзвенной архитектуры.
3. Особенности трехзвенной архитектуры ИС.
4. Распределенные (многозвенные) ИС.
5. Принципы технологии COM.
6. Понятие глобального уникального идентификатора.
7. Механизм автоматизации как средство межпроцессного взаимодействия.
8. Общая схема работы контроллера автоматизации.
9. Приведите примеры «прозрачного» использования механизма автоматизации при разработке приложений в Borland C++ Builder.
10. Функции и методы для взаимодействия с объектами автоматизации в Borland C++ Builder.