Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования

Владимирский государственный университет

Кафедра информационных систем
и информационного менеджмента

ПРОГРАММИРОВАНИЕ ДЛЯ MICROSOFT .NET

Методические указания к лабораторным работам
по дисциплине «Программирование»

В двух частях

Часть 1

Составители
В. В. ВЕРШИНИН
С. В. ЧЕБЫКИН

Владимир 2010

УДК 004.4
ББК 32.973
П78

Рецензент
Кандидат технических наук
доцент кафедры информационных систем и информационного менеджмента Владимирского государственного университета
С.Ю. Кириллова

Печатается по решению редакционного совета
Владимирского государственного университета

П78 **Программирование** для Microsoft .NET : метод. указания к лабораторным работам по дисциплине «Программирование». В 2 ч. Ч. 1 / Владим. гос. ун-т ; сост. : В. В. Вершинин, С. В. Чебыкин. – Владимир : Изд-во Владим. гос. ун-та, 2010. – 56 с.

Содержат описание и задания к лабораторным работам по дисциплине «Программирование». Включают работы по изучению и освоению подходов к анализу и проектированию программных систем с использованием языка UML и программного средства Microsoft Visio.

Часть работ посвящена проектированию и разработке Web-приложений с использованием подходов и средств, предоставляемых платформой Microsoft .NET. Лабораторные работы выполняются с использованием среды разработки Microsoft Visual Studio 2008. Часть работ выполняется с подключением к системе управления базами данных Microsoft SQL Server 2003 или 2005.

Предназначены для студентов 3 – 4-х курсов специальности 230201 – информационные системы и технологии (системы поддержки принятия решений), и других, изучающих дисциплины соответствующего профиля.

Табл. 1. Ил. 10. Библиогр.: 4 назв.

УДК 004.4
ББК 32.973

ПРЕДИСЛОВИЕ

Необходимость и важность обеспечения высокого уровня разрабатываемых программных систем обусловливает использование современных технологий и подходов к их проектированию. Такой подход позволяет существенно повысить уровень программных разработок.

Использование новых технологий возможно только при условии их полного изучения и владения ими. Поэтому изучение существующих подходов позволяет сделать существенный прорыв в индустрии программного обеспечения и в подходах разработки программ.

Предлагаемый в данных методических указаниях к лабораторным работам по курсу «Программирование» материал позволит познакомить студентов с различными аспектами программной инженерии. С одной стороны, это изучение процедур анализа и проектирования программных систем, с другой стороны, их реализация на базе выбранных средств и технологий.

Первый цикл лабораторных работ (с первой по третью) дает совокупное представление о построении взаимоотношений программной системы и элементов внешней среды и позволяет научиться определять границы исследуемой системы и выделять её функции в виде прецедентов, знакомит с методами и средствами построения моделей для различных уровней проектирования ИС с использованием графического языка моделирования UML и средства Microsoft Visio.

Второй цикл работ (с четвертой по седьмую) дает основы Web-программирования. Знания, полученные в этой части работ, позволяют освоить подходы разработки как простых Web-приложений, так и более серьезных приложений, ориентированных на обработку и сохранение данных в базах данных.

Лабораторная работа № 1

АНАЛИЗ ПРЕЦЕДЕНТОВ РАБОТЫ С ПРОГРАММНОЙ СИСТЕМОЙ. МОДЕЛИРОВАНИЕ ВЗАИМООТНОШЕНИЙ ИС И ЭЛЕМЕНТОВ ВНЕШНЕЙ СРЕДЫ. РАЗРАБОТКА ДИАГРАММ ПРЕЦЕДЕНТОВ

1. Цель работы

Научиться моделировать взаимоотношения элементов внешней среды и элементов проектируемой программной системы через синтез прецедентов, их расширенное описание и включение в диаграмму UseCase.

2. Общие сведения

Никакие системы не существуют в изоляции. Как правило, они взаимодействуют с элементами внешней по отношению к ним (участниками взаимодействия, актерами) – людьми или программами, которые используют систему в своих целях, причем каждый актер (actor) ожидает, что она будет вести себя определенным вполне предсказуемым образом.

Диаграммы прецедентов позволяют визуализировать поведение системы, подсистемы или класса, чтобы пользователи могли понять, как их использовать, а разработчики - реализовать соответствующий элемент.

Прецеденты состоят из множества сценариев (последовательности шагов, описывающих взаимодействие пользователя и системы), объединенных вместе некоторой общей целью пользователя. Относительно сложная система содержит несколько десятков прецедентов, каждый из которых может разворачиваться в несколько десятков сценариев. Для любого прецедента можно выделить основные сценарии, описывающие важнейшие последовательности, и вспомогательные, описывающие альтернативные последовательности. У каждого прецедента есть предусловие и постусловие.

В качестве примера объекта информатизации в рамках настоящего издания мы выберем библиотеку. В данном случае актером, элементом, взаимодействующим с будущей ИС «Библиотека», будет читатель. Как правило, у читателя, зашедшего в библиотеку, есть два сценария пове-

дения – поиск и выбор нужной ему литературы и передача прочитанных им книг обратно в распоряжение библиотеки.

В нашем случае мы рассмотрим прецедент «Выбирать литературу» который охватывает два подсценария: успешного нахождения нужной литературы и неудачного поиска. Выбранный прецедент подробно описан ниже.

Название: «Выбирать литературу».
Предусловие: читатель зарегистрирован в ИС «Библиотека», т.е. имеет присвоенный ему номер читательского билета, он авторизован в системе.
Действующее лицо: читатель.
Основной поток: *Выбирать литературу.*
Читатель открывает форму (окно приложения), отображающую каталог литературы.
Пользуется средствами поиска по каталогу (вводит автора или название издания в соответствующие поля).
Отмечает необходимую литературу и нажимает кнопку «Выписать требование» - данные найденной книги или журнала (прецедент «Выписать требование»).
Система сохраняет требование в БД и перенаправляет его библиотекарю.
Альтернативный поток: книга в каталоге отсутствует.
На 2-м шаге читатель не находит требующуюся книгу или журнал. В этом случае он
либо начинает поиск другой литературы, либо корректирует условия поиска.
Постусловие: если книга или журнал найдены, должно быть выписано требование на литературу – запрос на выдачу найденной литературы.

Стоит заметить, что словесное описание любого прецедента, даже самого элементарного, должно включать в себя как минимум 3 раздела – предварительное условие, основной поток, постусловие. Однако в общем случае описание прецедента может включать в себя один или несколько альтернативных потоков, связанных с различными шагами основного потока, ссылки на другие прецеденты, указание конкретных актеров, вовлеченных в прецедент (раздел “Действующие лица/актеры”). Последнее связано с тем, что прецедент могут инициировать несколько актеров и в зависимости от конкретного актера прецедент может включать в себя специфические шаги основного потока или даже специфические альтернативные потоки.

После словесного описания **каждого** сценария прецедентов можно

приступить к построению диаграммы прецедентов (Use case diagram). На ней показываются совокупность прецедентов и актеров, а также отношения между ними.

Рассмотрим основные элементы диаграммы прецедентов. На диаграмме (рис. 1) представлены упомянутые выше прецеденты «Choose» и «Write Requirement».

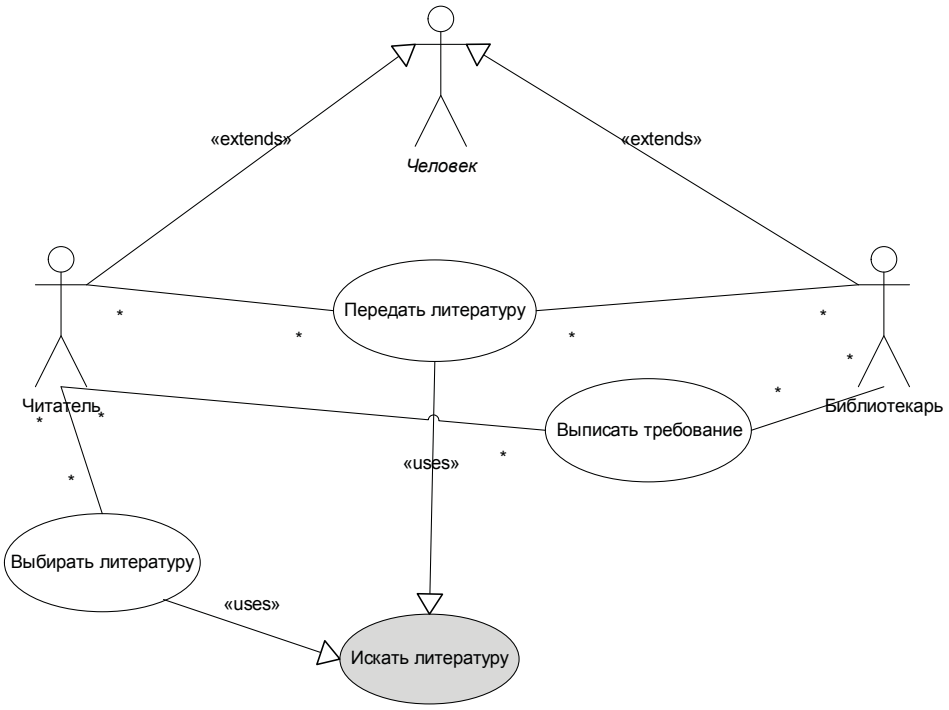

Рис. 1. Диаграмма прецедентов

Графически прецеденты изображаются в виде эллипсов. Прецеденты бывают двух типов: бизнес-прецеденты (описывают функциональность на верхнем уровне и предназначены для заказчика программной системы) и системные прецеденты (описывают функциональность на нижнем уровне, строятся для разработчика программной системы).

У любого прецедента должно быть имя, уникальное в рамках пакета. Имя может занимать несколько строк. На практике для именования прецедентов используют короткие глагольные фразы в активной форме, обозначающие некоторое поведение. Прецеденты можно организовать, определив между ними отношения обобщения, использования и расширения.

Отношение *обобщения* между прецедентами аналогично отношениям обобщения между классами. Это означает, что прецедент-потомок наследует поведение и семантику своего родителя, может замещать его или дополнять его поведение, а кроме того, может быть подставлен всюду,

где появляется его родитель. Отношение обобщения следует использовать при описании изменения некоторого нормального поведения.

Отношение *использования* (uses) имеет место, когда существует какой-либо фрагмент поведения системы, который повторяется более чем в одном прецеденте. Например, для обоих прецедентов «Выбрать литературу» и «Искать литературу» системе, равно как и ее пользователям – библиотекарю и читателю, необходимо найти нужную литературу. Описание поиска журналов или книг можно представить как отдельный сценарий. Поэтому выделяем отдельный прецедент «Искать литературу».

Отношение *расширения* (extend) по своей сути аналогично отношению обобщения. При построении модели расширяющий прецедент может дополнять поведение базового прецедента, но в последнем должны быть определены точки расширения (только для UML). При этом расширяющий прецедент может дополнять поведение базового только в этих точках расширения. Отношение расширения используется при более точном описании изменения некоторого нормального поведения.

На рис. 2, *б* представлен пример применения отношения расширения для фрагмента предметной области «ИС супермаркета». Прецедент «Оплатить покупку с дисконтной картой» расширяет прецедент «Оплатить покупку» дополнительными действиями – вводом данных о дисконтной карте; точка расширения – момент, когда необходимо ввести информацию о карте, – указано в прецеденте «Оплатить покупку».

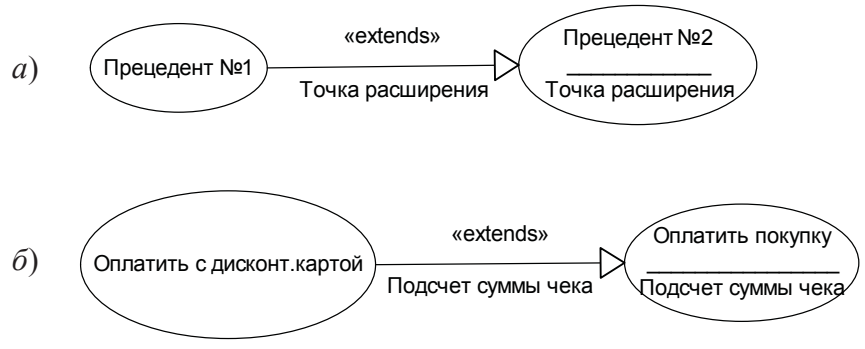

Рис. 2. Отношение расширения: а - общая схема, б – пример отношения расширения

Актер представляет собой некоторую роль, которую играет пользователь системы. Графически актеры изображаются в виде «человечков» (рис. 3). Актеры не обязательно должны быть людьми. В качестве актера может выступать любой внешний объект, пользующийся информацией от разрабатываемой системы.

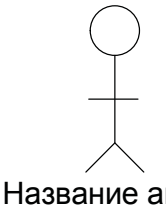

Рис. 3. Графическое изображение актера

На рис. 1 показаны три актера: «Человек», «Библиотекарь» и «Читатель». Между актерами также могут возникать различные отношения (обобщение, агрегация, зависимость). Например, «Читатель» и «Библиотекарь» обобщаются актером «Человек».

В библиотеке может быть много читателей и библиотекарей, но так как по отношению к системе они играют одни и те же роли, нам нет необходимости рассматривать конкретных читателей и библиотекарей в отдельности. Один и тот же человек может иметь несколько ролей. Например, библиотекарь тоже может взять почитать книгу.

Актер может выполнять несколько прецедентов, прецедент может выполнять несколько актеров.

Диаграммы прецедентов имеют большое значение для визуализации, специфицирования и документирования поведения элемента. Они облегчают понимание систем, подсистем или классов, представляя взгляд извне на то, как данные элементы могут быть использованы в соответствующем контексте. Кроме того, такие диаграммы важны для тестирования исполняемых систем и для понимания их внутреннего устройства.

3. Порядок выполнения работы

1. Перед выполнением предлагаемых лабораторных работ рекомендуется ознакомиться со следующими дополнительными источниками:

- Леоненков А.В. Нотация и семантика языка UML (лекции № 1 – 9) http://www.intuit.ru/department/pl/umlbasics/
- Грекул В.И. Проектирование информационных систем (лекции № 1 – 3, 11, 12) http://www.intuit.ru/department/se/devis/

2. Для выполнения данной лабораторной работы здесь и далее необходимо программное средство Mcrosoft Office Visio 2003 и более новые версии (для выполнения практикума потребуется набор графических шаблонов «UML Model Diagram (Metric Units)»).

3. Выбрать и подробно исследовать объект информатизации. Кратко (порядка одного абзаца) описать выбранный ОИ. Определить границы исследуемого объекта, актеров. Описать 3 – 4 прецедента по образцу, представленному в п. 2 данной работы. Построить диаграмму прецедентов, используя в Microsoft Office Visio набор графических примитивов UML Use Case (Metric).

4. Содержание отчета

1. Цель работы.
2. Вариант индивидуального задания.
3. Словесное описание объекта информатизации и всех выделенных прецедентов.
4. Диаграмма прецедентов.
5. Выводы по работе.

5. Контрольные вопросы

1. Что такое прецедент?
2. Каковы основные элементы, выносимые на диаграмму UseCase?
3. Что включает в себя расширенное описание прецедента?
4. Пояснить суть отношений между прецедентами (включение и расширение).

6. Варианты индивидуальных заданий

Каждый вариант индивидуального задания представляет собой описание некоторого объекта информатизации (ОИ) – предприятия, отдела или бизнес-процесса, который оперирует со специфичной информацией, характерной для данного объекта информатизации. Задача студента заключается:

- в определении границ ОИ и его взаимодействия с элементами внешней для него среды; результат – текстовое описание прецедентов и построение диаграммы прецедентов;
- определении структуры ОИ и протекающих в нем процессов обработки информации, анализе структуры ОИ и выделении атрибутов и поведения компонентов исследуемого объекта; результат – построение диаграммы классов;
- анализе поведения структурных элементов ОИ и их взаимодействия; результат – построение диаграмм деятельности.

В случае если на этапах анализа и проектирования выяснится, что каких-либо данных не хватает для логичного и законченного представления системы, студент вправе сам дополнить исследуемый объект необходимыми компонентами/атрибутами/поведением.

Итог работы – сводный отчет, включающий в себя результаты работы над проектом ИС по каждому этапу. Объекты информатизации перечислены ниже.

1. Университет.

2. Склад.
3. Производственное предприятие.
4. Сеть магазинов.
5. Сеть авторемонтных мастерских.
6. Деканат.
7. Поликлиника.
8. Телефонная станция.
9. Управление городским транспортом.
10. Авиакомпания.
11. Интернет-магазин.
12. Фермерское хозяйство.
13. Автотранспортное предприятие.
14. Отдел кадров.
15. ГИБДД.

В качестве объекта информатизации студент может выбрать произвольный бизнес-процесс, предприятие или отдел. Работа студента оценивается по следующим принципам и критериям:

- адекватность и достоверность представления объекта информатизации в моделях; поскольку работа ведется над реально существующими объектами, перед проведением анализа выбранный ОИ должен быть подробно изучен;
- соблюдение требований методологий проектирования ИС;
- соблюдение сроков выполнения этапов лабораторного практикума.

Лабораторная работа № 2
АНАЛИЗ ОБЪЕКТА ИНФОРМАТИЗАЦИИ И МОДЕЛИРОВАНИЕ СТРУКТУРЫ ПРОГРАММНОЙ СИСТЕМЫ НА ВЕРХНЕМ УРОВНЕ. РАЗРАБОТКА ДИАГРАММ КЛАССОВ

1. Цель работы

Выполнить анализ объекта информатизации и смоделировать структуры будущей программной системы на верхнем уровне в виде конечных диаграмм классов.

2. Общие сведения

Перед выполнением предлагаемых лабораторных работ рекомендуется ознакомиться со следующими источниками:

- Леоненков А.В. Нотация и семантика языка UML (лекции № 1 – 9) http://www.intuit.ru/department/pl/umlbasics/

• Грекул В.И. Проектирование информационных систем (лекции № 1 – 3, 11, 12). http://www.intuit.ru/department/se/devis/ или с литературными источниками из библиографического списка методических указаний.

Необходимые программные средства:

Mcrosoft Office Visio 2003 и более новые версии (для выполнения лабораторных работ потребуется набор графических шаблонов «UML Model Diagram (Metric Units)»).

Моделирование структуры ИС на верхнем уровне

Моделирование предметной области позволяет сконцентрировать внимание на наиболее важных деталях проекта, особенно при разработке крупных программных систем, когда в проект вовлечено большое число людей. Язык визуального моделирования позволяет более наглядно показать схему будущего программного продукта. Помимо этого он предоставляет возможность создавать техническую документацию и каркас кода будущего программного продукта. Основные принципы моделирования структуры ИС рассмотрим на знакомом по предыдущей лабораторной работе примере библиотеки.

У каждого читателя есть читательский билет, который однозначно идентифицирует посетителя библиотеки. Придя в библиотеку, читатель ищет по каталогу нужные ему книги и журналы. Далее он заполняет требование (список литературы), потом, предварительно проверив их наличие, библиотекарь выдает книги и журналы читателю.

Диаграмма классов занимает центральное место при проектировании ИС. Фактически любая методология проектирования, ориентированная на объект, включает в себя некоторую разновидность диаграммы классов.

На *диаграмме классов* (Class Diagram) показывают классы, интерфейсы, объекты и кооперации, а также их отношения. Выделяют два вида отношений: ассоциации (читатель может взять книгу в библиотеке) и подтипы (библиотекарь может передать литературу и выписать требование).

Рассмотрим основные элементы диаграммы классов.

Класс (Class) - это описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Класс реализует один или несколько интерфейсов. Графически класс изображается в виде прямоугольника, в котором обычно записаны его имя, атрибуты и операции (рис. 4).

Ассоциация (Association) - структурное отношение, описывающее совокупность связей; связь – это соединение между объектами (рис. 5). Графически ассоциация изображается в виде прямой. Каждая ассоциация имеет два конца. Конец ассоциации может быть явно помечен некоторой меткой – ролью. Если метка отсутствует, то концу ассоциации присваивается имя целевого класса.

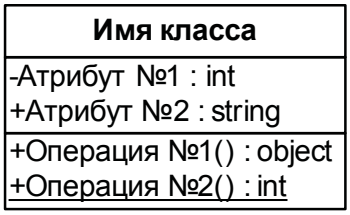

Рис. 4. Изображение класса в UML

Разновидностью ассоциации является *агрегирование* (Aggregation). В двух словах, агрегация – это когда класс в своей структуре использует часть другого класса. Например, можно сказать, что процессор или жесткий диск являются частью компьютера.

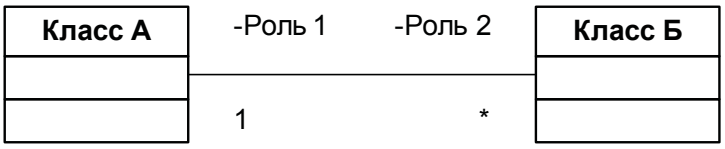

Рис. 5. Отношение ассоциации между классами

Существует более сильная разновидность агрегации – *композиция* (Composition). При композиции класс является целиком частью другого класса. Например, запись в требовании целиком является частью требования.

Конец ассоциации обладает *кратностью* (Multiplicity), которая показывает, сколько объектов может участвовать в отношении. Выделяют 4 основных варианта кратности:

- «1» - только один;
- «*» - много (ноль или более);
- «0..1» - необязательная (ноль или один);
- «m..n» - заданное число.

Например, для ассоциации между Каталогом (Catalogue) и Литературным изданием (Requirement), символ «*» показывает, что в одном каталоге может (и должно) быть несколько книг, а «1» – что вся литература может находиться лишь в одном каталоге.

Существует еще один способ связи классов – *обобщение* (Generalization) Обобщение - это отношение между общей сущностью (суперклассом, или родителем) и ее конкретным воплощением (субклассом, или потомком). Обобщения иногда называют отношениями типа «является», имея в виду, что одна сущность (например класс Книга (Book) или Журнал (Magazine)) является частным выражением другой, более общей (скажем, класса Литературное издание (Literature)).

Класс может иметь одного или нескольких родителей или не иметь их вовсе. Класс, у которого нет родителей, но есть потомки, называется базовым (base), или корневым (root), а тот, у которого нет потомков, – листовым (leaf). О классе, у которого есть только один родитель, говорят, что он использует одиночное наследование (Single inheritance); если родителей несколько, речь идет о множественном наследовании (Multiple inheritance).

Чтобы показать, что один из классов реализует поведение, специфицированное в другом классе, используют *реализацию*.

Для объединения часто повторяющихся групп блоков используют *пакеты* (Packages). В пакет можно поместить структурные, поведенческие и даже другие группирующие сущности. В отличие от компонентов, существующих во время работы программы, пакеты носят чисто концептуальный характер, то есть существуют только во время разработки.

С целью повышения информативности диаграммы можно использовать заметки (Note).

3. Порядок выполнения работы

Провести анализ предметной области на основе прецедентной мо-

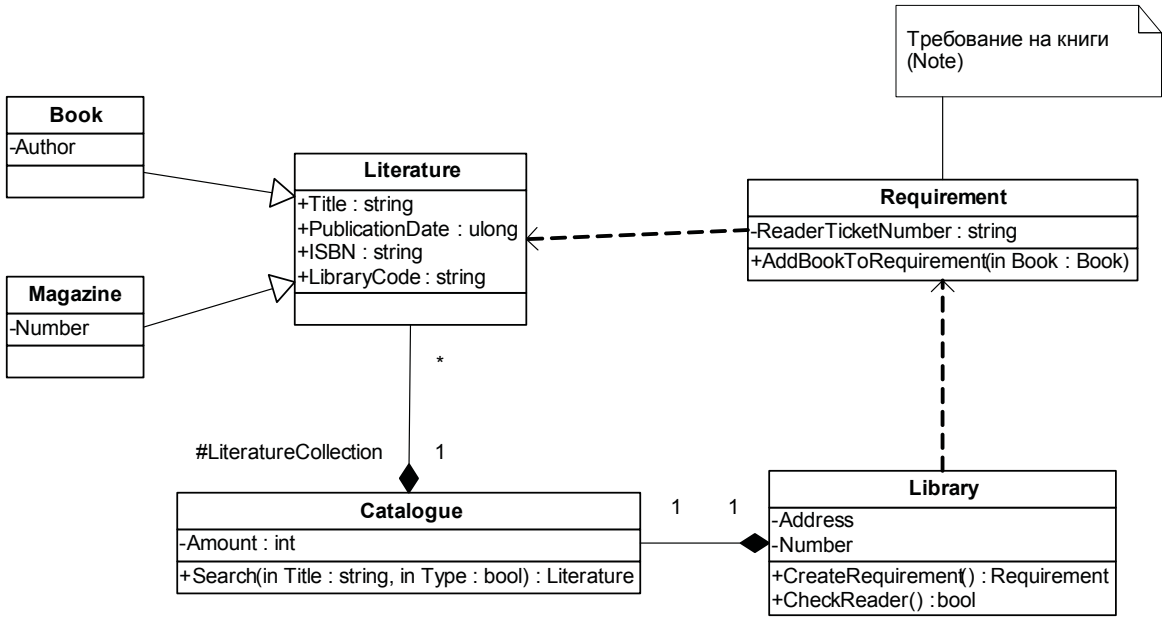

Рис. 6. Фрагмент диаграммы классов ИС «Библиотека»

дели, построенной в предыдущей лабораторной работе. Выделить основные сущности предметной области, их атрибуты и поведение, определить взаимосвязи между сущностями. Построить диаграмму классов анализируемой предметной области, используя для примера рис. 6.

4. Содержание отчета

1. Цель работы.
2. Вариант индивидуального задания.
3. Перечень функций системы, полученный в результате анализа диаграммы прецедентов.
4. Словесное описание сущностей системы, полученное в результате анализа диаграммы прецедентов (включая описание атрибутов классов и функций, реализующих его поведение).
5. Описание взаимоотношений между классами.
6. Диаграмма классов.
7. Выводы по работе.

5. Контрольные вопросы

1. Понятие и назначение диаграммы классов.
2. Перечислить основные элементы, выносимые на диаграмму классов. Пояснить назначение каждого их них.
3. Перечислить виды отношений между классами. Раскрыть суть каждого из них.
4. Описать порядок выделения классов предметной области и их вынесение на диаграмму классов.

6. Варианты индивидуальных заданий

Задания к лабораторной работе даны в методических указаниях к лабораторной работе № 1 и являются ее продолжением в данной работе.

Лабораторная работа № 3

ДИНАМИЧЕСКОЕ МОДЕЛИРОВАНИЕ РАБОТЫ СИСТЕМЫ

1. Цель работы

Изучить методы анализа поведения компонентов программной системы путем синтеза диаграмм, описывающих поведение моделируемой ИС с использованием нотации UML.

2. Общие сведения

Для большинства разрабатываемых систем, кроме самых простых и тривиальных, статических представлений совершенно недостаточно для моделирования процессов функционирования подобных систем как в целом, так и их отдельных подсистем и элементов.

Чтобы понять это, достаточно рассмотреть любой реальный объект окружающего мира. Понятно, что кроме сведений о его структуре отдельный интерес представляет и то, как и чем этот объект представлен во время своей «жизни». Это важно понимать и оговаривать, когда над ним возможно выполнять различные действия в зависимости от того, в каком состоянии он находится. Например, можно ввести такие значимые состояния для объекта автомобиль, как «движется» и «стоит». Очевидно, что операция «ремонтировать автомобиль» неприменима, когда он находится в состоянии «движется».

Объекты программных систем, являясь отражением реальных объектов предметной области, также нуждаются в анализе и моделировании их поведения, а также взаимодействия во времени. Нотация UML поддерживает в своем составе специальный набор диаграмм, которые обеспечивают динамическое представление работы элементов системы и их состояния. Фактически для динамического представления системы в UML существует три вида диаграмм: диаграммы состояний (Statechart diagramm), диаграммы последовательности (Sequence diagramm) и диаграммы кооперaций (Collaboration diagramm).

Диаграмма состояний (Statechart diagram). Отражает внутренние состояния объекта в течение его жизненного цикла от момента его создания до разрушения. Изображается в виде графа. Каждый объект рассматривается как сущность, которая контактирует с окружающим миром при помощи обмена сообщениями. Например, если читатель хочет взять в библиотеке книгу, он должен сначала запросить каталог (найти его), таким образом он посылает запрос на нахождение книги. Состояния объекта связаны с некоторыми событиями. Определение состояния объекта зависит от самого объекта и уровня моделирования. Диаграмма состояний используются для моделирования динамики поведения класса, поэтому ее целесообразно создавать только для объектов, имеющих несколько состояний. Этой диаграмме не ставится в соответствие программный код.

Состояние (State) описывает период времени в течение жизни объекта некоторого класса. Оно может быть представлено тремя дополняющими друг друга способами: изменением значений переменных объекта; как время ожидания объектом некоторого события или времени, когда это событие произойдет; или же как период времени, в течение которого объект совершает некоторое продолжающееся действие.

Состояние может иметь имя, хотя довольно часто оно не имеет имени и описывается действиями, которые совершаются объектом в этом состоянии. Состояние может быть расширено: под названием состояния можно указать действия, выполняемые объектом в данном состоянии.

Событие (Event) – заслуживающий внимания случай, который произошел в данное время в данном месте. Оно не имеет длительности, то есть рассматривается как мгновенно произошедшее. Событие может иметь некоторый набор параметров, которые характеризуют его, и бывает исходящим и входящим.

Переход (Transition) объекта из одного состояния в другое отображается направленной стрелкой. Переход характеризуется входными событиями, ограничивающими условиями, аргументами, действиями и посылаемыми событиями. Ограничивающие условия показывают, при каких условиях совершается данный переход. Это не обязательный элемент, но он часто используется, особенно при автоматическом переходе, поскольку делает диаграмму более понятной.

История (History) – свойство объекта запоминать предыдущие состояния, которое отображается на диаграмме специальным значком, размещаемым внутри элемента «Состояние». Этот элемент не может использоваться без связи с состоянием, к которому относится история.

Начальное состояние – это состояние объекта, в котором он создается.

Конечное состояние – состояние, из которого объект не может вернуться в активное состояние.

Рассмотрим использование диаграмм состояний на примере состояний объекта «Заказ» в ИС «Интернет-магазин». Один из возможных бизнес-процессов работы ИС с заказом представлен ниже:

1. Пользователь (покупатель) оформил заказ, указав свои реквизиты, способ оплаты и доставки. Объект «Заказ» создан в системе (объект в памяти и/или запись в БД). Поскольку оплата заказа производится через внешнюю платежную систему, заказ не обрабатывается до момента получения подтверждения об оплате.

2. От платежной системы пришло сообщение об отказе в оплате или истек максимальный срок ожидания оплаты. Покупателю на ука-

занный им e-mail отправляется соответствующее уведомление, после чего переходим к п. 6.

3. От платежной системы пришло подтверждение об оплате. Объект «Заказ» помечается как оплаченный. Адрес, указанный покупателем при оформлении заказа, передается во внешнюю систему проверки почтовых адресов; заказ ожидает подтверждения о проверке.

4. От системы проверки адресов пришло сообщение об ошибке в адресе. Покупателю на указанный им e-mail отправляется соответствующее уведомление; платежной системе отправляется сообщение о возврате платежа. После этого переходим к п. 6.

5. От системы проверки адресов пришло подтверждение корректности адреса. Заказ помечается как доставляемый и переходит в состояние ожидания подтверждения о доставке. Покупателю отправляется уведомление о передаче заказа в службу доставки.

6. Заказ помечается как отмененный.

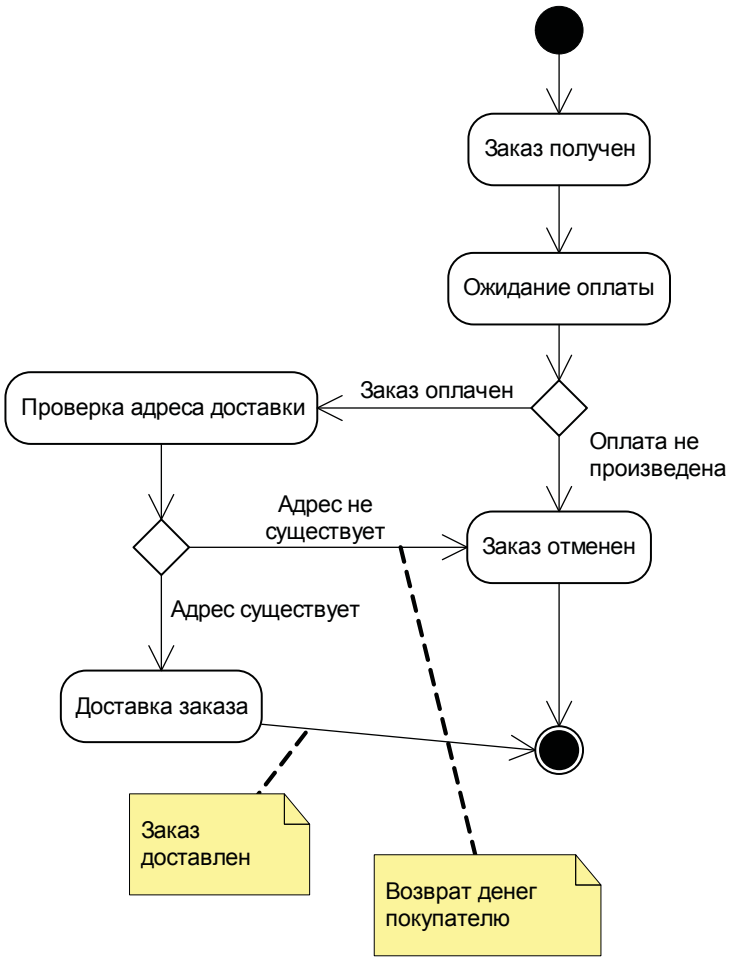

Рис. 7. Диаграмма состояний для объекта «Заказ»

Пример диаграммы состояний для объекта «Заказ» представлен на рис. 7.

Диаграмма последовательностей (Sequence diagramm). Диаграмма последовательностей является диаграммой взаимодействия, отражающей поток событий, происходящих при реализации одного из вариантов использования. На этой диаграмме изобража-

ются действующие лица, объекты, а также посылаемые и принимаемые ими сообщения. При построении в первую очередь принимается во внимание временная последовательность происходящих событий. На диаграмме последовательностей отображаются следующие элементы:

Линия жизни объекта (Lifeline) – отображает все, что происходит с объектом от его создания до разрушения.

Объект (Object) – в рамках диаграммы последовательности это экземпляр класса, сущность, шаблон.

Сообщение – поведение, связанное с передачей некоторой информации от одного объекта к другому, при котором передаваемая информация инициирует некоторое действие, например посылку сигнала, вызов некоторой операции, создание или уничтожение объекта. Сообщения бывают трех типов:

- *Сообщение (Message)* – горизонтальная стрелка от одного объекта к другому.
- *Сообщение с задержкой (Message with delivery time)* – наклонная стрелка от одного объекта к другому.
- *Сообщение объекта самому себе (Self message)* – направленная стрелка, начинающаяся и заканчивающаяся на одном и том же объекте.

Стимулирующее воздействие – взаимодействие между двумя объектами, которое передает информацию в предположении, что будет выполняться некоторое действие. Вызывает некоторую выполняемую операцию, а также создание или уничтожение объекта.

Активация – отображает период времени, в течение которого объект выполняет некоторое действие непосредственно или с помощью некоторой подчиненной процедуры. Это понятие характеризует продолжительность выполнения некоторого действия, которому в данный момент передано управление.

Время перехода используется для определения временных характеристик сообщения, которые могут использоваться в записи ограничений и условий. Функции времени могут включаться в логические выражения, истинность которых определяет последующий поток управления.

Пример диаграммы последовательностей для рассматриваемого нами примера «Заказ» представлен на рис. 8.

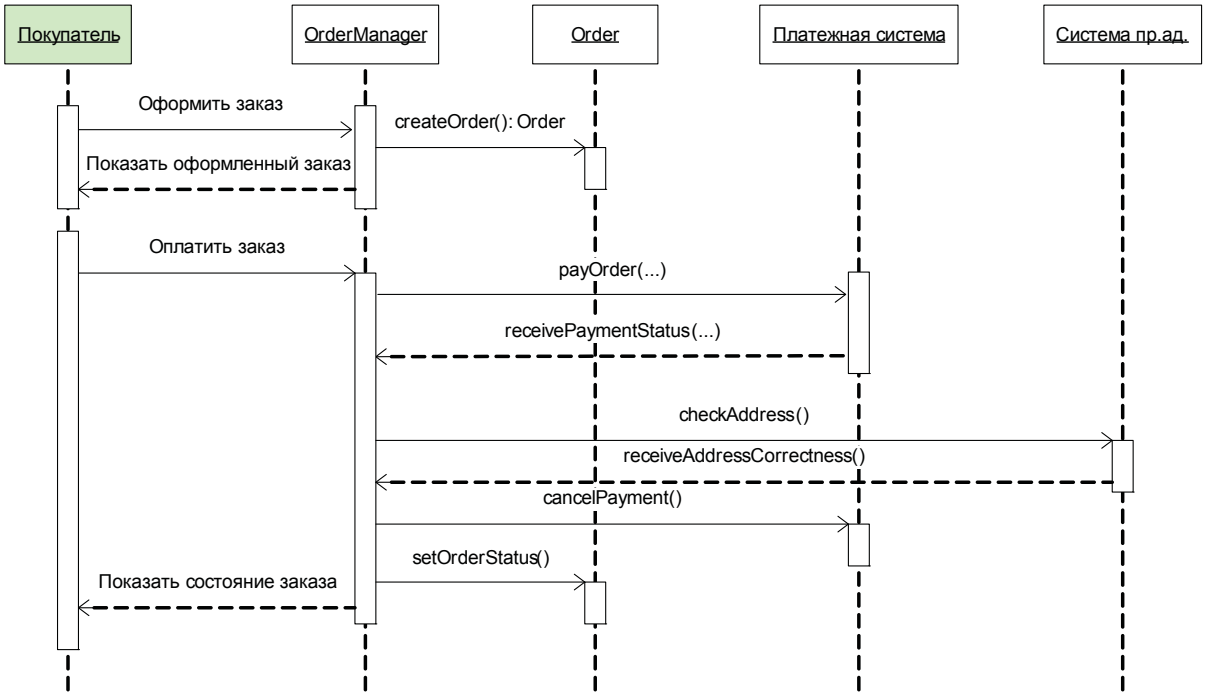

Рис. 8. Диаграмма последовательностей

На приведенной диаграмме последовательностей **Покупатель** является *актером*, а не объектом системы. В некоторых вариантах нотации UML изображается в виде человечка. **OrderManager** – класс ИС, реализующий подсистему работы с заказами. **Order** – заказ. **Платежная система** и **Система пр.ад.** (проверки адресов) – внешние ИС по отношению к проектируемой; в проектируемой ИС фигурируют как внешние Web-сервисы.

3. Порядок выполнения работы

1. Произвести анализ предметной области. Выделить сущности предметной области, определить их взаимосвязи, атрибуты и поведение.

2. Разработать и построить диаграмму состояний в соответствии с вариантом индивидуального задания, используя диаграмму классов.

3. Разработать диаграмму последовательностей при помощи ранее разработанных диаграмм прецедентов.

4. Содержание отчета

1. Цель работы.
2. Вариант индивидуального задания.
3. Описания состояний, переходов (включая условия перехода).
4. Описание взаимодействий.
5. Диаграммы состояний и последовательностей.
6. Выводы по работе.

5. Контрольные вопросы

1. Пояснить суть динамического моделирования программной системы. Указать назначение и цель моделирования.

2. Перечислить и охарактеризовать основные виды диаграмм, предназначенных для динамического моделирования системы.

3. Что такое линия жизни объекта?

6. Варианты индивидуальных заданий

Для выбранной ранее предметной области разработать диаграммы состояний, последовательности и кооперации.

Лабораторная работа № 4

НАЧАЛЬНОЕ ЗНАКОМСТВО С ПРОСТЕЙШИМИ WEB-ПРИЛОЖЕНИЯМИ (ASP.NET)

1. Цель работы

Изучить базовые принципы построения и функционирования простейших Web-приложений на платформе ASP.NET 2.0.

2. Общие сведения

2.1. *Технология ASP.NET*

ASP .NET — это элемент технологии .NET, используемый для написания клиент-серверных интернет-приложений. Она позволяет создавать динамические страницы HTML. ASP .NET возникла в результате объединения более старой технологии ASP (активные серверные страницы) и .NET Framework. Она содержит множество готовых элементов управления, применяя которые, можно быстро создавать интерактивные Web-сайты, которые также называются **Web-приложениями**.

Что такое динамические страницы HTML и чем они отличаются от

статических? Статическая страница содержит код на языке гипертекстовой разметки HTML. Когда автор страницы пишет ее, он определяет, как будет выглядеть страница для всех пользователей. Содержание страницы будет всегда одинаковым, независимо от того, кто и когда решит ее просмотреть. Языка HTML вполне достаточно для отображения информации, которая редко изменяется и не зависит от того, кто ее просматривает.

Динамическими принято называть Web-страницы, которые перед отправкой клиенту проходят цикл обработки на сервере (или целиком генерируются сервером). В самом простом случае это может быть некоторая программа, которая модифицирует запрашиваемые клиентом статические страницы, используя параметры полученного запроса и некоторое хранилище данных.

Существует несколько вариантов построения Web-приложений:

1. Вариант на основе «чистого» HTML (приводится в качестве общей демонстрации):

```
<html>
<body>
  <form>
    <input type="text" name="op1"/>
    +
    <input type="text" name="op2"/>
    <input type="submit" value=" = " />
 </form>
</body>
</html>
```

Приведенный выше пример содержит форму, в которой находятся 2 текстовых поля ввода и одна кнопка. По нажатию кнопки все данные из элементов формы (тэг <form>…</form> и все его содержимое) запросом POST протокола HTTP будут отправлены на сервер, однако обработаны они не будут. Другими словами, в ответ сервер вернет исходную HTML-страницу.

2. Вариант на основе CGI (Common Gateway Interface – общий шлюзовый интерфейс). Здесь запрос отправляется на сервер также с формы статической HTML-страницы, однако в ответ специальные программы на сервере (исполняемые программы, написанные на языках C/C++, скрипты на языке Perl) сгенерирует страницу с учетом полученных им

данных формы. Особенностями CGI является то, что они представляют низкоуровневый программный интерфейс; для обработки каждого запроса на сервере запускается свой процесс (следствие – коллизии, взаимные блокировки, нарушение целостности данных и т.п.); низкая скорость. Такой подход в основном используется на unix-платформах.

3. Вариант на основе ISAPI. В отличие от CGI вместо исполняемых файлов или скриптов на сервере используются DLL-библиотеки. Код DLL находится в памяти все время и для каждого запроса создает не процессы, а потоки (Threads) исполнения. Все потоки используют один и тот же программный код. ISAPI-приложение выполняется в процессе Web-сервера. Это позволяет повысить производительность и масштабируемость.

4. Скриптовые языки (PHP, ASP, отчасти JSP). Здесь код, исполняемый на сервере внедрен в текст страницы в виде специальных тэгов. Когда пользователь запрашивает страницу, на сервере выполняется код, внедренный в текст страницы. В задачу этого кода, как правило, входит извлечение каких-либо данных из БД (сообщений ветки форума, информации о погоде и т.п.) и представление этих данных в виде HTML. В результате пользователь получает страницу, структура (шаблон) которой неизменна, а содержимое генерируется сервером.

Этот вариант наиболее распространен, однако имеет существенный недостаток – HTML-код и код, выполняемый на сервере, «свалены в кучу», что сильно затрудняет работу как дизайнеров, так и программистов и делает неудобным сопровождение Web-приложения.

5. ASP.NET позволяет избавиться от недостатков, присущих скриптовым языкам. В ASP.NET HTML-код страницы и программный код, выполняемый на сервере (серверный код), выделены в разные файлы. Содержимое ASP.NET-страниц помимо тэгов HTML включает в себя так называемые **серверные элементы управления** – специальные тэги, которые имеют объектное представление в серверном коде.

Так, например, элемент Label на ASP.NET-странице выглядит следующим образом: «`<asp:Label id="Label1" runat="server" Text="" />`», а в серверном коде он же представлен в виде объекта класса Label с именем Label1, и к свойству Text у него можно обратиться следующим образом: «`Label1.Text = "Hello!";`».

Очевидно, что на сгенерированной сервером странице вместо тэга, приведенного выше, будет красоваться надпись “Hello!”

2.2. *Первый проект*

Первый проект создадим с использованием программной среды Microsoft Visual Web Developer 2005 Express edition (или ей подобных). При создании необходимо задать единый каталог проекта. После запуска среды выберите пункт меню File → New → Web Site. Появится диалоговое окно. Назначьте в нем имя проекта и выберите язык программирования C#.

По умолчанию проект создается в файловой системе. По желанию его можно создать на HTTP или FTP-сервере. Из файловой системы проект всегда можно скопировать на сервер нажатием одной кнопки в заголовке Solution Explorer.

В проекте будет создана страница default.aspx. Выберите ее, и появится окно редактирования с закладками Design и Source. Не меняя ничего, щелкните на кнопке со стрелкой, чтобы просмотреть страницу в браузере. Появится окно, в котором спрашивается, нужно ли добавить в файл web.config возможность отладки. Нажмите “OK”. Откроется браузер, показывающий страницу по адресу http://localhost:номер_порта/Website1/default.aspx. “Localhost” обозначает сервер, работающий на вашем компьютере. Встроенный сервер, используемый по умолчанию, сам назначает себе номер порта – для каждого проекта он разный. Сервер IIS обычно работает через порт 80 (или 8080, если порт 80 занят), и для него номер порта указывать не нужно. При этом ваша страница будет скомпилирована.

Пока что страница в браузере пустая. Но исходный код этой страницы не пустой. Среда автоматически сгенерировала код страницы.

```
<%@ Page Language="C#" AutoEventWireup="true";
CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC «-//W3C//DTD XHTML 1.0
Transitional//EN»;
«http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd»>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
```

```
<body>
    <form id="form1" runat="server">
    <div>
    </div>
    </form>
</body>
</html>
```

Разберем содержимое этой страницы:

<%@ Page Language=»C#» %> – тег <% всегда предназначается для интерпретации ASP-кода. Директива Page всегда присутствует на странице aspx. Ее атрибут Language – это указание, что в скриптах данной страницы будет использоваться C#, а могли бы VB, C++ или J#. CodeFile – имя файла с отделенным кодом (code-behind). Inherits – класс, определенный в том файле, от которого наследуется класс страницы.

Одновременно будет создан и файл Default.aspx.cs. Подход разделения кода называется “фоновый код”. Сама форма находится в файле Default.aspx, а в файле Default.aspx.cs находится класс страницы на языке C#. Таким образом, дизайн страницы может быть изменен не затрагивая кода страницы, что позволяет разделить ответственность за внешний вид и работу страницы между дизайнером и программистом.

`<form runat="server">` – тэг дает указание компилятору обрабатывать элементы управления страницы. Обратите внимание на то, что данный тэг имеет свойство runat, для которого установлено значение «server» (других значений не бывает). При использовании этого свойства элемент управления обрабатывается компилятором, а не передается браузеру «как есть».

Вставьте в Default.aspx между тэгами <form> и </form> тэг, задающий элемент управления: `<asp:Label id="Time" runat="server" Text="Сервер находится в Москве. Московское время: "/>`

Серверный элемент управления Label является средством размещения на странице текста, который может содержать тэги HTML. Изменяя значения свойств этого элемента управления в коде, можно динамически изменять текст на странице. В asp:Label компилятору сообщается, с каким объектом ведется работа (в рассматриваемом случае с элементом управления Label).

Далее задаются различные свойства элемента управления. В первую очередь определяются его имя id=»time» и атрибут «runat», а также текст.

В файле Default.aspx.cs должен содержаться такой текст:

```
using System;
......
public partial class _Default : System.Web.
UI.Page
{
 protected void Page_Load(object sender, EventArgs
e) {

    }
}
```

Ключевое слово partial появилось в C# 2.0 и позволяет разбить текст определения класса между разными файлами. Класс System.Web.UI.Page является базовым для всех страниц ASP .NET.

Вставьте в эту функцию строчку `Time.Text += DateTime.Now.ToString();`, что означает получение системного времени. Это значение присваивается свойству Text объекта time (time – элемент управления типа Label (метка)). Время на часах клиента и сервера может не совпадать, если они находятся в разных точках земного шара. Метод Page_Load похож на обычный обработчик события формы. Как можно легко догадаться, эта функция вызывается каждый раз, когда загружается форма.

Запустите страницу на просмотр кнопкой F5 или нажмите на кнопку со стрелкой на панели инструментов. В браузере должна открыться страница, на которой будет написано текущее время. Откройте исходный текст страницы. Никакого кода на C# или элементов управления ASP.NET там не будет:

```
<!DOCTYPE html PUBLIC «-//W3C//DTD XHTML 1.0
Transitional//EN»
«http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd»>
<html xmlns=”http://www.w3.org/1999/xhtml” >
<head><title>Время, вперед</title>
</head>
```

```
   <body>
    <formname="form1"method="post"action="Default.
aspx"
   id="form1">
   <div>
   <input type="hidden" name="__VIEWSTATE" id="__
VIEWSTATE"
   value="/wEPDwUJODExMDE5NzY5D2QWAgIDD2QWAgIBDw8W
Ah4EVGV4dAUSMDguMDYuMjAwNiA0OjU2OjQ3ZGRkkEMgqXmKC0
v9vwAwh999lefuIOw=" />
   </div>
   <div>
        <span id="Time">Сервер находится в Москве.
Московское время:
   08.06.2006 4:56:47</span>
       </div>
       </form>
   </body>
   </html>
```

Обновив страницу, можно увидеть новое значение времени на сервере.

3. Порядок выполнения работы

1. Создать Web-приложение, как это описано в п. 2.2.

2. С помощью панели инструментов Toolbox (меню View → Toolbox) добавьте в вашу Web-страницу элементы управления TextBox и Button.

3. С использованием вышеперечисленных элементов создайте Web-приложение "Калькулятор", реализующее операции сложения, вычитания, умножения и деления, а также проверку вводимых значений.

4. Содержание отчета

1. Тексты всех разработанных исходных кодов.
2. Таблицы с исходными данными.
3. Результаты выполнения запросов.
4. Выводы по работе.

5. Контрольные вопросы

1. Что такое Web-приложение и каковы его основные особенности?

2. Что такое серверный элемент управления? Чем он отличается от HYML элемента управления?

6. Варианты индивидуальных заданий

Для выбранной ранее предметной области разработать простейшее Web-приложение и реализовать базовый функционал.

Лабораторная работа № 5

РАБОТА С ДАННЫМИ В ASP.NET 2.0

1. Цель работы

Познакомиться с рядом средств ASP.NET 2.0 для представления и отображения данных.

2. Общие сведения

2.1. *Привязка к данным*

При создании Web-приложений часто решается задача отображения на клиентской стороне некоторых данных, которые на стороне сервера могут быть представлены либо в объектном (экземпляры классов с данными, списочные типы или коллекции), либо в реляционном виде (таблицы данных, представляющие собой выборки из БД).

Платформа ASP.NET предоставляет универсальный и достаточно простой способ отображения данных на стороне клиента. Данные (в каком бы виде они ни существовали на сервере) могут быть представлены на клиентской стороне с помощью специальных элементов управления. Задача этих элементов управления – отображение данных на Web-странице в удобной для пользователя форме в виде:

- таблицы (так, например, может быть представлен прайс-лист);
- множества одинаковых по структуре (но имеющих разное содержимое) HTML-фрагментов; в таком виде, например, представляются результаты поискового запроса в Гугле (и не только), страницы тем форумов, ленты новостей на крупных информационных порталах;
- иных способов представления данных.

Рассмотрим механизм отображения данных на элементарнейшем прототипе системы управления контентом сайта: в нашем примере реа-

лизована страница добавления новостей на сайт и отображения только что добавленных новостей. Например, сущность «Новость» на стороне сервера представлена следующим классом:

```
    /// <summary>
    /// Класс, описывающий бизнес-объект - элемент новости
    /// </summary>
    public class NewsItem {
        private string title;
        private string text;
        private string author;
        private int articleRating;

        public string Title {
            get { return title; }
            set { title = value; }
        }

        public string Text {
            get { return text; }
            set { text = value; }
        }

        public string Author {
            get { return author; }
            set { author = value; }
        }

        public int ArticleRating {
            get { return articleRating; }
            set { articleRating = value; }
        }

        public NewsItem() {
            Random rnd = new Random();
        this.ArticleRating = rnd.Next(5);
        }
    }
```

Как видно из примера, сущность «Новость» содержит 4 свойства – Title (заголовок), Text (текст новостной статьи), Author (e-mail автора) и ArticleRating (рейтинг статьи), который в данном примере может принимать значения от 0 до 5 и выставляется случайным образом.

Все множество новостей представлено типизированным динамическим списком «`protected List<NewsItem> newsList;`», где каждый элемент списка имеет приведенный выше тип.

Каким образом данные привязываются к серверным элементам управления? Некоторые серверные элементы управления: списки (DataList), таблицы (GridView), репитеры или повторители (Repeater) – имеют свойство **DataSource**, которое отвечает за привязку к данным. Тип этого свойства – **object**, то есть он может быть любым, но должен реализовывать интерфейс **IEnumerable**. Часто значениями этого свойства назначают коллекции.

Свойство DataSource может быть «привязано» к коллекциям, поддерживающим интерфейсы IEnumerable, ICollection или IListSource. Источником данных также могут быть XML-файлы, таблицы баз данных, массивы. Вызовом метода DataBind() данные непосредственно привязываются к серверному элементу управления. Метод Page. DataBind() вызывает привязку данных у всех элементов на странице.

Так, в рассматриваемом примере для отображения новостей используется серверный элемент управления **Repeater** (повторитель). Для привязки к повторителю всего множества новостей, представленных списком **newsList**, на сервере достаточно вызвать следующий код:

```
this.repNews.DataSource = newsList;
this.repNews.DataBind();
```

Как повторитель “выглядит” в коде рассматриваемого примера?

```
<%!-- Это, собственно, тэг повторителя. Ниже
рассмотрено его содержимое --%>
<asp:Repeater ID=”repNews” runat=”server”>

<%!-- Повторитель содержит HTML-шаблоны
(template) для следующих своих “частей”:
- шаблон заголовка (HeaderTemplate)
- шаблон повторяемой части (ItemTemplate)
- шаблон завершающей строки (Footer
Template)
Заголовок и завершающая строка будут
присутствовать на сгенерированной с использованием
```

повторителя странице однократно.

Повторяемая же часть будет повторена столько раз, сколько элементов в коллекции у связанного с повторителем бизнес-объекта: т.е. столько, сколько строк у DataTable, (в случае, если DataTable привязана к повторителю) или столько, сколько элементов у массива или списка...

В рассматриваемом нами примере мы “плодим” тэг <tr> обычной HTML-таблицы, наполняя его при этом данными из бизнес-объекта

```
            --%>
           <HeaderTemplate>
               <table width=”100%”>
               <tr>
        <td style=”background-color: #ebebeb”><h3>Лента
новостей:</h3></td>
               </tr>
           </HeaderTemplate>


           <ItemTemplate>
               <tr>
                 <td style=” background-color: #ebdada”>
                       <h4>
  <%!— Ниже, в рамках тэга <%# … %>- “серверный” C#-
код, вписанный непосредственно в код Web-страницы. Этот
код отвечает за отображение в данном конкретном месте
значение соответствующего поля конкретного элемента
коллекции. В данном случае будет отображено значение
поля Title объекта класса NewsItem, находящегося в
“привязанной” к повторителю коллекции.
  --%>

             <%#   DataBinder.Eval(Container.DataItem,
“Title”) %>
                                 </h4>
                                 <div>
             <%#   DataBinder.Eval(Container.DataItem,
“Text”) %>
                                 </div>
                                <div>Рейтинг статьи: 
              <%#  DataBinder.Eval(Container.DataItem,
```

```
"ArticleRating") %>
                                </div>
                                <div     style="text-align:
right">
      <%# DataBinder.Eval(Container.DataItem, "Author")
%>
                                </div>
                    </td>
                </tr>
            </ItemTemplate>

            <FooterTemplate>
                        <tr><td style="background-color:
#daebeb">Здесь могла быть Ваша реклама!</td></tr>
                </table>
            </FooterTemplate>
        </asp:Repeater>
```

Как непосредственно осуществляется привязывание бизнес-объектов к серверным элементам управления, отображающим данные из этих бизнес-объектов, продемонстрировано выше, в комментариях к коду повторителя.

Компоненты GridView и DataList имеют практически полностью аналогичную структуру, однако имеют значительно больше свойств и опций, позволяющих настраивать их вид согласно дизайну разрабатываемого сайта.

2.2. *Проверка вводимых данных*

Вводимые на Web-страницах данные как правило записываются в базы данных, а следовательно, и тип информации в них должен соответствовать типу и длине данных в соответствующих полях таблиц баз данных. Кроме того, иногда нужно вводить взаимосвязанные данные, например, пароль во время регистрации нужно вводить 2 раза и он в обоих полях должен совпадать. Или вводимые данные должны соответствовать определенному формату.

В рассматриваемом нами примере, прежде чем работать с данными, нужно убедиться, что:

- в поле «Заголовок новости» введен текст;
- текст в поле «E-mail писаря» имеет форму электронного адреса (с @ и с точкой).

Проверка может происходить и на стороне клиента, и на сервере. С одной стороны, при проверке (валидации) на стороне клиента в стра-

ницу встраивается код на Javascript. Если данные в форме не проходят проверку, страница просто не будет отправлена на сервер. Это позволит избежать лишнего траффика и не загружать канал связи с сервером. С другой стороны, валидация на стороне сервера более надежна. Javascript-код может быть легко посмотрен и изменен, наконец, Javascript можно просто выключить в настройках браузера. При валидации на стороне сервера данные проверяются программой на полноценном языке. Ее код пользователю неизвестен. В результате проверки генерируется новая страница с сообщениями об ошибках. Самая разумная стратегия – применять комбинацию этих методов. Предварительная проверка у клиента защитит от опечаток, а серьезная проверка на сервере – от злонамеренного взлома.

Существует целый ряд серверных элементов управления, которые не занимаются выводом информации, а проверяют данные, введенные пользователем. ASP.NET 2.0 сам определяет тип браузера и генерирует наиболее подходящий для данного случая код. Если браузер поддерживает Javascript-код, который он может послать, то валидация или ее часть происходит на стороне клиента. Если браузер не поддерживает Javascript, то вся валидация происходит на сервере.

Получить доступ к валидаторам просто – они находятся в панели Toolbox на вкладке «Validation». Классы валидаторов образуют иерархию, во главе которой стоит абстрактный класс BaseValidator.

Базовый класс валидаторов – сам наследник класса Label, так что по существу все валидаторы – метки, текст в которых становится видимым, когда не выполняются заданные нами условия проверки. По умолчанию текст в валидаторах красный (вспомните школу и замечания учительницы в тетради). Все валидаторы имеют свойство ControlToValidate. Оно задает тот элемент управления, содержимое которого проверяется данным валидатором. Этот элемент должен находиться в одной форме с валидатором. Основные свойства валидаторов представлены в таблице.

Свойство	Описание
Display	Предоставлять ли место статически или динамически
EnableClientScript	Разрешать ли генерировать клиентский javascript-код
ErrorMessage	Текст сообщения об ошибке
IsValid	Прошел ли валидацию связанный с валидатором элемент управления

Проверка данных всегда инициируется каким-либо событием. Обычно это щелчок на кнопках Button, ImageButton, LinkButton, у которых свойство CausesValidation по умолчанию установлено в True. Можно убрать это свойство для некоторых кнопок, которым оно не нужно, например для кнопки Cancel. В нашем примере на Web-странице имеется несколько валидаторов:

```
<asp:RequiredFieldValidator ID="rfvTitle" runat="server" ErrorMessage="RequiredFieldValidator" ControlToValidate="txtTitle"> Заголовок не может быть пустым! </asp:RequiredFieldValidator>
```

Класс RequiredFieldValidator проверяет, было ли изменено значение в связанном с ним элементе управления. Если, как в данном случае, это элемент ввода текста, требуется, чтобы пользователь ввел туда что-нибудь. Если выбор не был сделан, но кнопка (см. пример) была нажата, валидация проваливается и выводится текст, заданный в ErrorMessage или в Text. Валидаторы отображают текст, указанный внутри тэга.

Для проверки корректности ввода электронного адреса используется класс RegularExpressionValidator:

```
<asp:RegularExpressionValidator ID="revAuthor" runat="server" ErrorMessage="RegularExpressionValidator" ControlToValidate="txtAuthor" ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"> Нужно ввести e-mail </asp:RegularExpressionValidator>
```

Класс ValidationExpression – регулярное выражение, на соответствие которому проходит проверку значение текстового поля. В Visual Studio 2005 предоставляет несколько готовых шаблонов регулярных выражений, которые можно выбрать в окне свойств: телефонных номеров разных стран, адресов, и, самое полезное, шаблоны электронной почты и адреса в Интернете.

С одним элементом управления может быть связано несколько валидаторов. Например, электронный адрес проверяется и на соответствие шаблону, и на обязательное заполнение.

Свойство Page.IsValid позволяет определить, прошла ли вся страница валидацию. Для браузеров, которые поддерживают DHTML, проверка происходит на стороне клиента. Для этого автоматически генерируется JavaScript-код. Таким образом экономятся ресурсы сервера и трафик, который бы пришлось потратить на передачу неправильных данных.

3. Порядок выполнения работы

1. Ознакомиться с примером, который предоставлен Вам преподавателем.

2. Исследовать следующие вопросы, касающиеся представленного примера:

- как данные представлены на стороне сервера;
- каким образом данные представляются на стороне клиента;
- каким образом клиентское представление связывается с серверным;
- как осуществляется проверка вводимых пользователем данных;
- как используется и работает компонент **Repeater**.

3. Расширить представленный пример, используя вместо компонента **Repeater** компоненты **DataList** или **GridView**.

4. Содержание отчета

1. Цель работы.

2. Вариант индивидуального задания.

3. Описание механизма работы разработанного приложения с хранилищами данных и комментариями по тексту программы.

4. Выводы по работе.

5. Контрольные вопросы

1. Каким образом осуществляется связывание данных и представления?

2. Какие типы объектов могут быть присвоены свойству DataSource?

3. Как (и на какой стороне – на сервере или на клиенте) осуществляется валидация данных?

4. Каким образом валидация осуществляется на клиенте?

5. Есть две кнопки – OK и Cancel. Мы хотим, чтобы по нажатию OK валидация выполнялась, а по нажатию Cancel – нет. Как будет выглядеть валидатор для кнопки OK?

6. Как узнать (на стороне сервера), прошла ли страница валидацию?

7. Как валидатор «привязывается» к элементу управления, который он проверяет?

6. Варианты индивидуальных заданий

Для выбранной ранее предметной области разработать простейшее Web-приложение и реализовать базовый функционал.

Лабораторная работа № 6

ДОСТУП К ДАННЫМ В .NET. ТЕХНОЛОГИЯ ADO.NET

1. Цель работы

Познакомиться с технологией доступа к данным ADO.NET.

2. Общие сведения

В настоящей лабораторной работе технология ADO.NET рассматривается на примере использования ASP.NET-приложением данных из БД под управлением СУБД Microsoft SQL Server 2005 Express, поставляемой вместе с Visual C# Express 2005 и Visual Web Developer Express 2005.

2.1. *Создание базы данных*

СУБД Microsoft SQL Server в настоящей работе подробно не рассматривается, поскольку в рамках одной работы невозможно даже поверхностно ознакомиться и с сотой долей возможностей СУБД. Стоит лишь отметить, что MS SQL Server – многоцелевая промышленная СУБД, используемая в приложениях самого разного назначения (от настольных до распределенных кластерных систем).

В настоящей работе рассматривается бесплатно распространяемая Express-версия SQL Server, имеющая ограничения по аппаратной конфигурации сервера (может использоваться на системах с 1 процессором, и иметь максимум 4 Гб оперативной памяти). SQL Server Express подходит для настольных и малых “офисных” систем.

Microsoft Visual Studio (любая редакция, в том числе Express) предоставляет в себя набор средств, необходимых для создания и выполнения несложных типовых операций с БД. Рассмотрим их ниже.

Для создания БД в среде разработки Visual Studio (Visual C# или Web Developer) нужно:

1. Убедиться, что SQL Server Express установлен и запущен (выбрать в контекстном меню значка “Мой Компьютер” пункт “Управление”, узел “Службы и приложения\Службы”, убедиться, что служба SQL Server (SQLEXPRESS) – собственно сервер – присутствует в списке и запущена).

2. Открыть панель Server Explorer (меню View → Server Explorer).

3. В контекстном меню узла Data Connections выбрать пункт Create New SQL Server Database. Для того чтобы создать соединение с уже существующей БД, в меню присутствует пункт Add Connection.

4. В открывшемся окне в поле Server name указать имя (сетевой адрес) сервера: LOCALHOST\SQLEXPRESS, а в поле New Database name – имя создаваемой Вами БД и нажать ОК. База с указанным Вами именем создана и доступна.

5. После этого в панели Server Explorer можно будет увидеть только что созданную БД и все ее элементы, представленные в виде дерева.

6. Для добавления в БД таблиц в контекстном меню узла Tables следует выбрать пункт Add New Table.

7. Для просмотра и «ручного» редактирования содержимого созданной Вами таблицы (таблиц) можно воспользоваться ее контекстным меню.

В качестве примера создадим базу данных **Test**, содержащую единственную таблицу **tb_News** (таблица для хранения новостей) со следующими полями:

1. **ID**, имеющее тип int.
2. **Text**, имеющее тип nvarchar(max).
3. **Author**, тип nvarchar(50).
4. **Rating**, тип int.
5. **Date**, тип DateTime.

Для колонки ID в редакторе ее свойств установим значение параметра Identity Specification\(Is Identity) в Yes. Это будет означать, что при каждой вставке новой строки в данную таблицу значение поля ID будет автоматически увеличиваться.

2.2. *Доступ к данным SQL Server в C# с использованием ADO.NET*

Технология ADO.NET фактически представляет собой множество пространств имен и классов для организации соединения с СУБД и управления данными. ADO.NET может использоваться в любых приложениях – от консольных до ASP.NET без каких-либо ограничений.

ADO.NET реализует **единый принцип доступа к данным** вне зависимости от используемой СУБД. Так, в качестве СУБД (и в качестве источника данных вообще) могут использоваться как MS SQL Server, так и Oracle, и MySQL и так далее, а также COM-компоненты, предоставляющие доступ к данным из Excel-файлов, таблиц MS Access и пр.

Принцип организации доступа к данным рассмотрим на примере MS SQL Server Express.

Для работы с данными необходимо подключить следующие пространства имен:

```
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
```

Принцип доступа к данным опирается на следующие понятия:

- соединение (Connection) – "канал" доступа к БД (и СУБД), по которому осуществляется все взаимодействие с базой данных; по сути соединение предоставляет доступ к источнику данных, которым является база данных. Для SQL Server используется класс **SqlConnection**;
- команда (Command) – объект, несущий в себе информацию о том, какие действия необходимо выполнить СУБД над данными; все взаимодействие с БД – выборка и изменение данных, создание объектов БД – осуществляется посредством команд. Для SQL Server следует использовать класс **SqlCommand**;
- методы передачи/сохранения или извлечения данных; таких методов реализовано много, в рамках настоящей работы мы рассмотрим простейшие из них. Для работы с SQL Server используются классы **SqlDataReader**, **SqlDataAdapter** и пр.;
- «потребитель» данных – объект на стороне клиента, в который данные должны быть помещены. В качетстве потребителей данных могут быть использованы уже знакомые Вам **DataTable**, а также **DataSet** и прочие объекты.

Рассмотрим, как осуществляется взаимодействие с СУБД.

Первым делом необходимо установить соединение с БД. Для определения параметров соединения используется строка соединения (**connection string**), которую в реальных приложениях, как правило, хранят в конфигурационном файле приложения.

Мы сделаем строку соединения атрибутом класса, в котором соединение будет использоваться:

```
private string connectionString = @"Initial
Catalog=<имя БД>;" +
        @"Data Source=<имя (адрес) сервера БД>;" +
        @"User ID=<имя пользователя>;" +
        @"Password=<пароль>";
```

Например, для созданной нами БД (п. 2.1) строка соединения будет выглядеть следующим образом:

```
«Data Source=LOCALHOST\SQLEXPRESS;Initial Catalog=Test;Integrated Security=True;Pooling=False»
```

Параметр «`Integrated Security=True;`» заменяет имя пользователя и пароль и указывает, что для получения доступа к SQL Server используется встроенная безопасность Windows. Фактически Вы «войдете» на сервер под теми же логином и паролем, под которыми Вы вошли в Windows.

Далее рассмотрим пример простейшей программы, прилагающийся к настоящей лабораторной работе. Следующий код создает и открывает соединение с БД, используя строку соединения.

```
SqlConnection connection;
try {
    connection = new SqlConnection(connectionString);
    connection.Open();
}
catch (SqlException ex) {
    Console.WriteLine("Ошибка при установлении соединения: ", ex.Message);
    Console.ReadKey();
    return;
}
```

При открытии соединения может возникнуть множество различных исключительных ситуаций (Exception): сервер СУБД недоступен, или на сервере отсутствует БД с указанным именем, или доступ для указанного пользователя запрещен.

Все исключительные ситуации, относящиеся к соединению с БД, имеют тип **SqlException** и перехватываются в предлагаемом примере.

Далее рассматриваются примеры вставки данных в таблицу и выборки данных из таблицы. Для удобства изучения программы эти примеры вынесены в отдельные функции. Функции принимают в качестве параметра уже открытое соединение с БД. Следующий код осуществляет вставку записи в таблицу **tb_News** (см. пример п. 2.1):

```
try {
    SqlCommand command1 = new SqlCommand("INSERT
INTO tb_News (Text, Author, Rating, Date) " +
      "VALUES ('Another one article', 'Somebody',
10, 09/30/2008)", connection);
        command1.ExecuteNonQuery();
}
catch (SqlException ex) {
   Console.WriteLine("Ошибка при вставке данных:
", ex.Message);
    Console.ReadKey();
}
```

Здесь создается экземпляр класса SqlCommand, в конструктор которому в качестве параметров передается SQL-запрос на вставку данных, а также созданное выше соединение.

Выполняется команда на сервере при вызове метода ExecuteNonQuery(). Важно отметить, что этот метод следует вызывать, когда нужно лишь передать команду SQL Server'у, и нет необходимости извлекать какие-либо данные из таблицы БД.

При выполнении команды также возможны исключительные ситуации, которые перехватываются в блоке catch { … }.

Для извлечения данных из таблицы у класса SqlCommand существуют другие методы. Представленный ниже пример иллюстрирует один из способов извлечения данных из БД.

```
SqlCommand command2 = new SqlCommand("SELECT *
FROM tb_News", connection);
SqlDataReader reader = command2.ExecuteReader();
while (reader.Read()) {
     Console.WriteLine("ID: {0}, Автор статьи:
{1}", reader[0], reader["Author"]);
}
reader.Close();
```

Здесь для извлечения данных используется класс SqlDataReader, который осуществляет построчное чтение **набора данных**, которые вернул запрос **SELECT * FROM tb_News**. При этом весь набор данных хранится на сервере БД. Вызов метода Read() инициирует передачу ровно одной строки данных в наше клиентское приложение и переходит на следующую строку набора данных.

К элементам строки можно обратиться как по индексу, так и по имени.

3. Порядок выполнения работы

1. Создать пример БД, как описано в п. 2.1, и изучить работу примера приложения, которое предоставлено преподавателем.
2. Изучить способ извлечения данных из БД с помощью класса SqlDataAdapter (см. документацию на этот класс в MSDN).
3. Модифицировать Web-сайт, разработанный Вами в предыдущей работе так, чтобы он использовал данные из БД.

4. Содержание отчета

1. Цель работы.
2. Вариант индивидуального задания.
3. Описание схемы БД.
4. Описание механизмов доступа к данным с использованием технологии доступа ADO.NET по коду программы.
5. Выводы по работе.

5. Контрольные вопросы

1. Основные особенности использования технологии ADO.NET и ее возможности.
2. Основные классы, используемые для организации слоя доступа к данным, их особенности.
3. Как извлекаются и обрабатываются данные на уровне приложения?

6. Варианты индивидуальных заданий

Для выбранной ранее предметной области и разработанного в предыдущей работе приложения организовать хранилище данных и доработать приложение так, чтобы доступ к нему производился через ADO.NET.

Лабораторная работа № 7

WEB-СЕРВИСЫ

1. Цель работы

Познакомиться с технологией создания и использования Web-сервисов.

2. Общие сведения

2.1. *Web-сервисы*

В процессе эволюции Интернета появилась необходимость в создании распределенных приложений. Приложения, установленные на компьютере, обычно используют библиотеки, размещенные на нем. Одну библиотеку могут использовать несколько программ. Можно ли размещать аналоги библиотек в Интернете, чтобы разные сайты могли их вызывать? Оказывается, да.

Предприятия на своих страницах предоставляют разнообразную информацию. Например, на своем сайте www.Ford.com компания «Форд» публикует информацию о моделях и ценах. Дилер этой компании хотел бы иметь эту информацию и на своем сайте. Web-служба позволяет сайту-потребителю получать информацию с сайта-поставщика. Сайт-потребитель показывает эту информацию на своих страницах. Код для генерации этой информации написан один раз, но может использоваться многими потребителями. Данные представлены в простом текстовом виде, поэтому ими можно пользоваться независимо от платформы.

Web-сервисы широко используются и в настольных приложениях, и в Интернет-приложениях. Они сами по себе являются не приложениями, а источниками данных для приложений. У них отсутствует пользовательский интерфейс. Web-сервисами необязательно пользоваться через сеть – они могут быть частью проекта, в котором используются.

Web-сервисы – это функциональность и данные, предоставляемые для использования внешними приложениями, которые работают с сервисами посредством стандартных протоколов и форматов данных. Web-сервисы полностью независимы от языка и платформы реализации. Технология Web-сервисов является краеугольным камнем программной модели Microsoft .NET. Это дальнейшее развитие компонентного программирования CORBA и DCOM. Однако для использования таких компонентов необходимо регистрировать их в системе потребителя. Для Web-служб этого не требуется. Компоненты хорошо работают в локальных сетях. Протокол HTTP не приспособлен для вызова удаленных процедур (RPC). Даже в одной организации часто используются разные операционные системы, которые могут взаимодействовать только через HTTP.

Было предпринято несколько попыток создания языка коммуникации между разнородными системами DCOM, CORBA, RMI, IIOP. Они

не получили всеобщего распространения, так как каждый из них продвигался разными производителями и поэтому был привязан к технологиям конкретного изготовителя. Никто не хотел принимать чужой стандарт. Чтобы решить эту дилемму, несколько компаний договорились выработать независимый от производителя стандарт обмена сообщениями через HTTP. В мае 2000 года компании IBM, Microsoft, HP, Lotus, SAP, UserLand и другие обратились к W3C и выдвинули SOAP в качестве кандидата на такой стандарт. SOAP произвел революцию в области разработки приложений, объединив два стандарта Интернета – HTTP и XML.

2.2. SOAP

Для взаимодействия с Web-сервисами применяется протокол SOAP, основанный на XML. Можно было бы использовать просто XML, но это слишком свободный формат, в нем каждый документ фактически создает свой язык. SOAP — это соглашение о формате XML-документа, о наличии в нем определенных элементов и пространств имен.

SOAP позволяет публиковать и потреблять сложные структуры данных, например DataSet. В то же самое время его легко изучить. Текущая версия SOAP 1.2.

SOAP – это простой протокол обмена данными (Simple Object Access Protocol). SOAP создан для того, чтобы облегчать взаимодействие приложений через HTTP. Это особый независимый от платформы формат обмена сообщениями через Интернет. Сообщение SOAP – это обычный XML-документ. Стандарт SOAP разрабатывает консорциум W3C.

SOAP-сообщение состоит из конверта (envelope), заголовка (header) и тела (body). Элементы body и envelope должны присутствовать всегда, а header необязателен. Элемент envelope – корневой. Элемент header может содержать специфичную для данного приложения информацию. В документе, сгенерированном Web-сервисом, может присутствовать элемент ault, который описывает ошибку работы.

```
POST /WebSite5/WebService.asmx HTTP/1.1
Host: localhost
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
```

```
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap12="http://www.w3.org/2003/05/soap-
envelope">
     <soap12:Body>
            <HelloWorld  xmlns="http://localhost/
webservices" />
     </soap12:Body>
   </soap12:Envelope>
```

В ASP .NET можно легко как создать Web-службу, так и пользоваться готовой.

2.3. *Пользование Web-службой*

В Интернете существует множество готовых Web-служб. Сайты http://uddi.microsoft.com, http://www.webservicelist.com/ – каталоги различных сервисов. Чтобы получить информацию от Web-службы, нужно только послать HTTP-запрос, в теле которого находится SOAP-сообщение. Запрос к службе http://www.webservicex.net/globalweathet.asmx на получение прогноза погоды в Москве выглядит так:

```
   POST /globalweather.asmx HTTP/1.1
   Host: www.webservicex.net
   Content-Type: text/xml; charset=utf-8
   Content-Length: length
   SOAPAction:        "http://www.webserviceX.NET/
GetWeather"

   <?xml version="1.0" encoding="utf-8"?>
   <soap:Envelope           xmlns:xsi="http://www.
w3.org/2001/XMLSchema-
   instance"    xmlns:xsd="http://www.w3.org/2001/
XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/soap/
envelope/">
     <soap:Body>
        <GetWeather xmlns="http://www.webserviceX.
NET">
         <CityName>Moscow</CityName>
         <CountryName>Russian</CountryName>
       </GetWeather>
```

```
  </soap:Body>
</soap:Envelope>
```

Заголовок запроса отличается от запросов, которые обычно посылают браузеры, прежде всего полем Content-Type — text/xml; а не text/html; В теле запроса находится SOAP-сообщение. Сервис в ответ оправляет XML-документ:

```
<?xml version="1.0" encoding="utf-8" ?>
  <string xmlns="http://www.webserviceX.NET">
<?xml version="1.0" encoding="utf-16"?>
 <CurrentWeather>
<Location>Moscow / Vnukovo , Russia (UUWW) 55-39N 037-16E</Location>
<Time>Aug 07, 2006 - 04:30 AM EDT / 2006.08.07 0830 UTC</Time>
  <Wind> from the E (080 degrees) at 11 MPH (10 KT):0</Wind>
  <Visibility> greater than 7 mile(s):0</Visibility>
  <SkyConditions> overcast</SkyConditions>
  <Temperature> 66 F (19 C)</Temperature>
  <DewPoint> 55 F (13 C)</DewPoint>
  <RelativeHumidity> 68%</RelativeHumidity>
  <Pressure> 29.85 in. Hg (1011 hPa)</Pressure>
  <Status>Success</Status>
</CurrentWeather></string>
```

Чтобы использовать Web-сервис, первым делом в проекте надо создать Web-ссылку на удаленный объект – Web-сервис. Выберите в меню Website пункт Add Web Reference. Появится диалоговое окно (рис. 9).

Введите URL Web-сервиса с параметром wsdl в текстовое поле www.webservicex.net/globalweather.asmx?WSDL. В файл web.config добавляется настройка приложения:

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
   <appSettings>
 <add key="weather.webservicex.www.globalweather"
value="http://www.webservicex.net/globalweather.asmx"/>
```

```
  </appSettings>
 <connectionStrings>
```

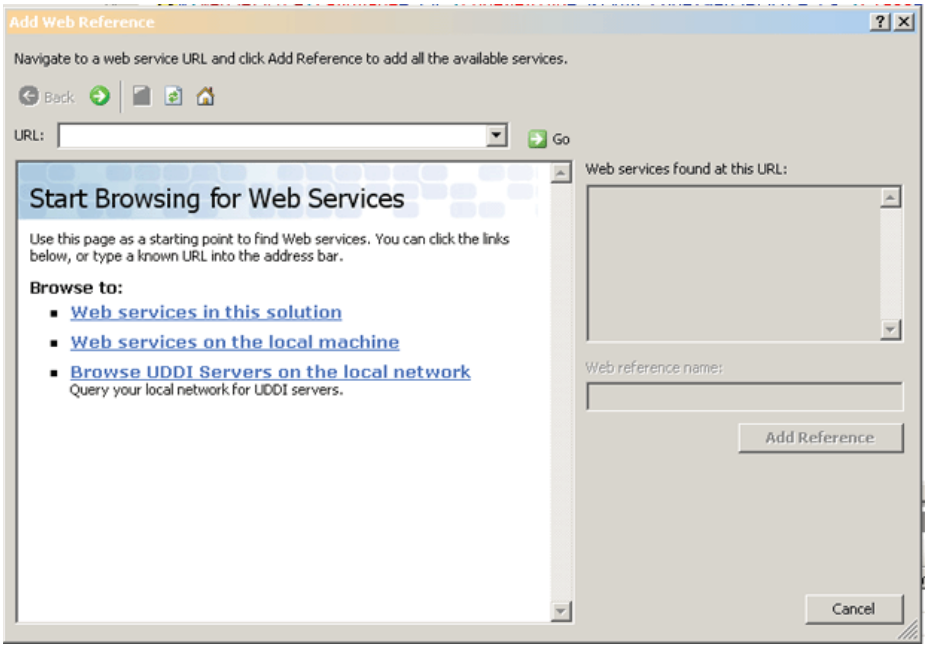

Рис. 9. Диалоговое окно Add Web Reference

Чтобы определить доступные по этому адресу Web-сервисы, используется алгоритм DISCO. При этом создается файл globalweather.wsdl. WSDL (Web Service Discovery Language) – это язык описания Web-сервисов. Это еще один тип XML-документов.

```
   <?xml version="1.0" encoding="utf-8"?>
   <wsdl:definitions     xmlns:soap="http://schemas.
xmlsoap.org/wsdl/soap/"
   xmlns:tm="http://microsoft.com/wsdl/mime/
textMatching/"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/
encoding/"
   xmlns:mime="http://schemas.xmlsoap.org/wsdl/
mime/"
   xmlns:tns="http://www.webserviceX.NET"
   xmlns:s="http://www.w3.org/2001/XMLSchema"
   xmlns:http="http://schemas.xmlsoap.org/wsdl/
http/"
   targetNamespace="http://www.webserviceX.NET"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
```

```
    <wsdl:types>
         <s:schema elementFormDefault="qualified"
targetNamespace="http://www.webserviceX.NET">
        <s:element name="GetWeather">
```

Создание Web-ссылки добавляет в конфигурационный файл еще одну запись:

```
  <configuration xmlns="http://schemas.microsoft.
com/.NetConfiguration/v2.0">
     <appSettings>
              <add key="weather.webservicex.www.
globalweather"
  value="http://www.webservicex.net/
globalweather.asmx"/>
     </appSettings>
```

Чтобы облегчить работу с Web-сервисами, используют прокси-классы. Они предоставляют разработчикам удобные функции и берут на себя преобразование их параметров в элементы XML, после чего посылают запрос Web-сервису через Интернет. Утилита wsdl в составе .NET Framework поможет преобразовать Web-сервис в прокси-класс: wsdl globalweather.wsdl.

Прокси-класс необходимо поместить в папку App_Code, после чего объекты этого класса можно создавать в коде любой страницы. Программу wsdl можно запустить и удаленно: wsdl http://www.webservicex.net/globalweather.asmx?wsdl.

В созданном файле объявлен класс GlobalWeather, наследник System.Web.Services.Protocols.SoapHttpClientProtocol. Функции этого класса предназначены как для синхронного вызова, так и для асинхронного. Например, синхронная функция GetWeather запрашивает строковые параметры с названием города и страны и возвращает строку с XML-документом. В сервисе есть и другая функция, с помощью которой можно узнать доступные города в каждой стране:

```
  [System.Web.Services.Protocols.
SoapDocumentMethodAttribute(
  "http://www.webserviceX.NET/GetWeather",
  RequestNamespace="http://www.webserviceX.NET",
  ResponseNamespace="http://www.webserviceX.NET",
  Use=System.Web.Services.Description.
```

```
SoapBindingUse.Literal,
   ParameterStyle=System.Web.Services.Protocols.
SoapParameterStyle.Wrapped)]
   public string GetWeather(string CityName, string
CountryName)
   {
        object[] results = this.Invoke(”GetWeather”,
new object[]
   { CityName, CountryName });

   return ((string)(results[0]));
   }
```

Этот текст можно использовать на странице:

```
       <form id=”form1” runat=”server”>
         <br />
           <asp:Button ID=”Button1” runat=”server”
Text=”А теперь прогноз погоды”
   OnClick=”Button1_Click” />
         <br />
         <br />
            <asp:Label ID=”lblTemp” runat=”server”
Text=”Температура в Москве ”></asp:Label>
       </form>

   protected void Button1_Click(object sender,
EventArgs e)
   {
   // создание экземпляра прокси-класса
        GlobalWeather gw = new GlobalWeather();

   // запрос функции web-сервиса
        string xmlstring = gw.GetWeather(”Moscow”,
”Russia”);
        XmlDocument doc = new XmlDocument();

   // загрузка ответа в документ
        doc.LoadXml(xmlstring);
```

```
// doc.ChildNodes.Item(0) — это XML-заголовок
// doc.ChildNodes.Item(1) — тело
     XmlNode child = doc.ChildNodes.Item(1);
     XmlElement el = child[”Temperature”];
     lblTemp.Text += el.InnerText;
}
```

2.4. *Создание Web-сервиса*

Создание Web-сервиса немногим отличается от создания обычной страницы. Есть два варианта: можно создать отдельный проект или вставить Web-сервис в существующий проект. В первом случае другие проекты должны создавать Web-ссылку, чтобы обращаться к сервисам этого проекта. Файл Web-сервиса имеет расширение asmx. Файл Web-сервиса должен начинаться с директивы WebService. Класс Web-сервиса может быть потомком класса System.Web.Services.WebService.

Если при объявлении Web-сервиса вы породили его от класса System.Web.Services.WebService, то автоматически получаете доступ к глобальным объектам приложения ASP .NET Application, Context, Session, Server и User. Если же вы создавали класс Web-сервиса как-то иначе – ничего страшного. Вы все равно можете получить доступ к вышеперечисленным свойствам с помощью соответствующих свойств статического HttpContext.Current.

Методы, которые сервис предоставляет для вызова с помощью SOAP-запросов, должны иметь атрибут WebMethod. У атрибута WebMethod существует 6 свойств, влияющих на работу метода.

Создадим новое приложение в VS .NET и добавим к нему файл Web-сервиса Customers.asmx. Файл nw.asmx содержит единственную строку – директиву WebService, которая утверждает, что этот файл – Web-сервис. Этот файл меняться не будет:

```
<%@ WebService Language=”C#” CodeBehind=”~/App_
Code/Customers.cs”  Class=” Customers ” %>
```

У директивы WebService всего 4 возможных атрибута. CodeBehind, который раньше был атрибутом и директивы Page, определяет физический путь к файлу отделенного кода. Атрибут Class обязателен и определяет класс, который реализует функциональность Web-сервиса. Debug и Language аналогичны тем же атрибутам директивы Page:

Файл с расширением .asmx – точка входа создаваемого Web-сервиса. Класс System.Web.Services.WebService, который обычно на-

следуется от класса сервиса, предоставляет доступ к глобальным объектам Application и ViewState.

Весь код Web-сервиса будет располагаться в codebehind-файле Service.asmx.cs. Изначально этот файл (созданный в Visual Studio .NET) имеет следующий вид:

```
using System;
using System.Web;
using System.Collections;
using System.Web.Services;
using System.Web.Services.Protocols;

/// <summary>
/// Summary description for WebService
/// </summary>
[WebService(Namespace = ”http://tempuri.org/”)]
[WebServiceBinding(ConformsTo = WsiProfiles.
BasicProfile1_1)]
public class MyWebService : System.Web.Services.
WebService {
    public WebService () {
        //Uncomment the following line if using
designed components
      //InitializeComponent();
       }

      [WebMethod]
      public string HelloWorld() {
           return ”Hello World”;
       }
}
```

Атрибут WebServiceBinding удостоверяет соответствие откликов Web-сервиса WS-I Basic Profile 1.0 release требованиям организации WS-I (Web Services Interoperability organization), которая занимается вопросами межплатформенной совместимости Web-сервисов.

Метод HelloWorld создан Visual Studio в качестве примера начинающим разработчикам.

Web-сервис может состоять из множества классов. Однако только

один класс в Web-сервисе может иметь методы, помеченные атрибутом WebMethod (которые можно вызывать через SOAP-запросы).

Когда страница Web-сервиса запрашивается в браузере, он возвращает автоматически генерируемую страницу помощи по данному сервису. Откроем в браузере страницу WebService.asmx. В браузере появится страница (рис. 10).

ASP .NET для отображения Web-сервиса использует файл шаблона DefaultWsdlHelpGenerator.aspx, расположенный в папке <%SYSTEM_ROOT%\Microsoft.NET\Framework\<version>\CONFIG. На выводимой странице Web-сервиса есть название Web-сервиса, ссылка на описание сервиса и список Web-методов, объявленных в Web-сервисе. Остальная часть страницы посвящена тому, что необходимо изменить пространство имен по умолчанию для Web-сервиса http://tempuri.org/ на собственное. Поступим, как рекомендуют, – изменим параметр Namespace атрибута WebService класса Web-сервиса: `[WebService(Namespace = "http://lab3.istx05.org/")]`

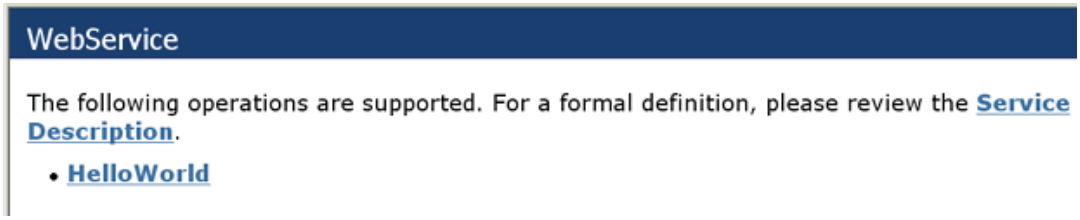

Рис. 10. Внешний вид окна WebService

Кроме того, в атрибуте WebService можно задать свойства Name и Description, и они появятся на странице помощи по Web-сервису.

Итак, перейдем к странице описания Web-метода SayHello (просто кликните по ссылке SayHello на странице описания Web-сервиса). На ней также присутствует название Web-сервиса, ссылка на первую страницу Web-сервиса, название Web-метода. Кроме этого на странице расположены кнопка Invoke, предназначенная для вызова Web-метода через GET-запрос (данное правило не выполняется, если Web-метод не может быть вызван таким образом), и примеры запросов для вызова данного Web-метода с помощью SOAP, HTTP POST (если такой вызов возможен) и HTTP GET (если такой вызов возможен). Также представлены примеры ответов вызова Web-метода. Протестируем нашу работу с помощью страницы Web-сервиса:

```
<?xml version="1.0" encoding="utf-8" ?>
<string                    xmlns="http://localhost/
```

```
webservices”>Hello World</string>
```

От такого Web-сервиса нет особенной пользы, поэтому создадим новый сервис `FirstWebService` и вместо метода HelloWorld напишем метод, который создает и возвращает объект класса DataSet:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Data;

namespace SampleWebService {
    /// <summary>
    /// Summary description for Service1
    /// </summary>
    [WebService(Namespace = “http://lab3.istx05.org/”)]
            [WebServiceBinding(ConformsTo = WsiProfiles.
BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
   public class FirstWebService : System.Web.Services.
WebService {
        [WebMethod]
        public DataSet GetCustomersInfo() {
                  DataSet data = new DataSet(“Customers
Data”);
          DataTable customers = new DataTable(“Customers
Table”);
              customers.Columns.Add(“ID”, typeof(Guid));
         customers.Columns.Add(“Name”, typeof(String));
                   customers.Columns.Add(“CreationDate”,
typeof(Date Time));

                customers.Rows.Add(Guid.NewGuid(), “Иван
Иванов», DateTime.Now);
                customers.Rows.Add(Guid.NewGuid(), “Петр
Петров», DateTime.Today);
              data.Tables.Add(customers);
              return data;
        }
    }
}
```

Другие аргументы атрибута WebMethod:

- CacheDuration определяет промежуток времени в секундах, на

которое кэшируется Web-сервис. По умолчанию равно 0, это значит, что кэширование отключено;

• в этом объявлении значение, возвращаемое методом GetCustOrders, кэшируется на 10 минут;

• Description – описание метода, которое выводится на странице сервиса под ссылкой на страницу метода;

• EnableSession позволяет включить поддержку сессий. По умолчанию поддержка сессий в Web-сервисах отключена. Чтобы включить ее, определите Web-метод следующим образом: [WebMethod(EnableSession=true)];

• MessageName определяет уникальное имя метода. Позволяет использовать перегруженные функции с одним именем, но разными сигнатурами.

TransactionOption управляет поддержкой транзакций. По умолчанию она отключена. Web-сервисы ограниченно поддерживают транзакции, то есть Web-сервис может порождать транзакцию, но при этом не может быть участником другой транзакции. Возможные значения аргумента – Disabled, NotSupported, Supported, Required и RequiresNew. Если вызывается Web-метод с TransactionOption, установленным в Required или RequiresNew, а в нем вызывается другой Web-метод с такими же установками, каждый из этих методов инициирует свою транзакцию.

Метод GetCustomersInfo возвращает тип DataSet. Посмотрим, как он помещается в SOAP:

```
<?xml version=»1.0» encoding=»utf-8» ?>
- <DataSet xmlns="http://lab3.istx05.org/">
-   <xs:schema id="Customers_x0020_Data" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
-     <xs:element name="Customers_x0020_Data" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
- <xs:complexType>
- <xs:choice minOccurs="0" maxOccurs="unbounded">
- <xs:element name="Customers_x0020_Table">
- <xs:complexType>
- <xs:sequence>
<xs:element name="ID" msdata:DataType="System.Guid, mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" type="xs:string" minOccurs="0" />
<xs:element name="Name" type="xs:string" minOccurs="0" />
<xs:element name="CreationDate" type="xs:dateTime" minOccurs="0" />
```

```
      </xs:sequence>
      </xs:complexType>
      </xs:element>
      </xs:choice>
      </xs:complexType>
      </xs:element>
      </xs:schema>
    - <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-
com:xml-msdata"       xmlns:diffgr="urn:schemas-microsoft-
com:xml-diffgram-v1">
    - <Customers_x0020_Data xmlns="">
    -  <Customers_x0020_Table  diffgr:id="Customers  Table1"
msdata:rowOrder="0" diffgr:hasChanges="inserted">
      <ID>4bcd2e13-04e1-4a4a-a885-36716615de38</ID>
      <Name>Иван Иванов</Name>
          <CreationDate>2008-10-16T01:31:00.6491699+04:00</
CreationDate>
      </Customers_x0020_Table>
    -  <Customers_x0020_Table  diffgr:id="Customers  Table2"
msdata:rowOrder="1" diffgr:hasChanges="inserted">
      <ID>8abc16cf-1bb7-4ca6-9860-5c83dbfe3a96</ID>
      <Name>Петр Петров</Name>
     <CreationDate>2008-10-16T00:00:00+04:00</CreationDate>
      </Customers_x0020_Table>
      </Customers_x0020_Data>
      </diffgr:diffgram>
      </DataSet>
```

Это почти готовая схема xsd. Из нее можно, например, сгенерировать класс бизнес-логики. Использовать созданный сервис можно и удаленно, и из приложения, в котором он находится. Стать потребителем этого сервиса могло бы и обычное Windows-приложение, написанное на C++, C# или Visual Basic. Web-сервисы тоже могут пользоваться услугами других Web-сервисов.

3. Порядок выполнения работы

1. Создать Web-сервис на основе примера в п. 2.4, который возвращал бы данные, извлеченные из БД (см. лаб. работу № 6).

2. Модифицировать Web-приложение, разработанное в предыдущей работе, таким образом, чтобы оно получало данные из сервиса, созданного в предыдущем пункте.

3. Ознакомиться со службой Aeroflot Web Services (http://webservices.aeroflot.ru/desc_flightnfo.asp). Создать Windows (Windows Forms)-

приложение, которое использовало бы одну из функций Web-сервиса компании ”Аэрофлот”.

4. Содержание отчета

1. Цель работы.
2. Вариант индивидуального задания.
3. Словесное описание сервиса.
4. Исходный код разработанного сервиса и примеры работы с ним.
5. Выводы по работе.

5. Контрольные вопросы

1. Что такое Web-сервис? Основные особенности.
2. Каковы основные элементы типового Web-сервиса?

6. Варианты индивидуальных заданий

Вариант работы взять из лабораторной работы № 1, предварительно согласовав его с преподавателем. Реализовать функции системы в виде Web-методов соответствующего сервиса.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. *Петцольд, Ч.* Программирование в тональности C# / Ч. Петцольд. – М. : Русская редакция, 2004. – 512 с. – ISBN 5-7502-0180-5.
2. *Просиз, Дж.* Программирование для Microsoft .NET / Дж. Просиз. – М. : Русская редакция, 2003. – 680 с. – ISBN 5-7502-0217-8.
3. *Вендров, А. М.* Проектирование программного обеспечения экономических информационных систем / А. М. Вендров. – М. : ФиС, 2005. – 352 с. – ISBN 5-279-02144-X.
4. *Буч, Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / Г. Буч. – М. : Бином, 2001. – 560 с. – ISBN 5-7989-0067-3.

ОГЛАВЛЕНИЕ

ПРОГРАММИРОВАНИЕ ДЛЯ MICROSOFT .NET

Методические указания к лабораторным работам
по дисциплине «Программирование»

Часть 1

Составители
ВЕРШИНИН Виталий Васильевич
ЧЕБЫКИН Сергей Владимирович

Ответственный за выпуск – зав. кафедрой профессор А.В. Костров

Подписано в печать 14.05.10.
Формат 60x84/16. Усл. печ. л. 3,25. Тираж 50 экз.
Заказ
Издательство
Владимирского государственного университета.
600000, Владимир, ул. Горького, 87.