Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Л. В. Серебряная

ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. СОЗДАНИЕ ПРИЛОЖЕНИЯ В СРЕДЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО CASE-СРЕДСТВА

Рекомендовано УМО по образованию в области информатики и радиоэлектроники для специальности 1-40 01 01 «Программное обеспечение информационных технологий» в качестве учебно-методического пособия

Библиотека БГУИР

Минск БГУИР 2012

УДК [004.413+004.4′22](076)
ББК 32.973.26-018.2я73
С32

Р е ц е н з е н т ы:
кафедра интеллектуальных систем учреждения образования «Белорусский национальный технический университет»
(протокол №12 от 10.05.2012 г.);

старший научный сотрудник Объединенного института проблем информатики Национальной академии наук Беларуси,
кандидат физико-математических наук С. В. Чебаков;

ведущий научный сотрудник Объединенного института проблем информатики Национальной академии наук Беларуси,
доцент кафедры интеллектуальных информационных технологий учреждения образования «Белорусский государственный университет информатики и радиоэлектроники»,
кандидат технических наук В. И. Романов

Серебряная, Л. В.

С32 Технологии разработки программного обеспечения. Создание приложения в среде объектно-ориентированного CASE-средства: учеб.-метод. пособие / Л. В. Серебряная. – Минск : БГУИР, 2012. – 50 с. : ил.

ISBN 978-985-488-883-5.

В пособии рассмотрены современные технологии разработки программного обеспечения. Предложены восемь лабораторных работ, реализация которых позволяет создать программное средство, построив UML модель и сгенерировав исходный код в среде автоматизированного синтеза Rational XDE.

Рекомендовано для обучения по дисциплине «Технологии разработки программного обеспечения».

УДК [004.413+004.4′22](076)
ББК 32.973.26-018.2я73

ISBN 978-985-488-883-5

© Серебряная Л. В., 2012
© УО «Белорусский государственный университет информатики и радиоэлектроники», 2012

Библиотека БГУИР

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4
1. СОВРЕМЕННЫЕ ТЕХНОЛОГИИ СОЗДАНИЯ ПРОГРАММНЫХ СРЕДСТВ 5
1.1. Технология Rational Unified Process 5
1.2. Унифицированный язык моделирования UML 8
1.3. Основные окна и пункты меню программы Rational XDE 9
2. ЛАБОРАТОРНЫЕ РАБОТЫ 13
2.1. Лабораторная работа №1 13
2.2. Лабораторная работа №2 16
2.3. Лабораторная работа №3 19
2.4. Лабораторная работа №4 25
2.5. Лабораторная работа №5 29
2.6. Лабораторная работа №6 36
2.7. Лабораторная работа №7 49
2.8. Лабораторная работа №8 44
ЗАКЛЮЧЕНИЕ 48
ЛИТЕРАТУРА 49

Библиотека БГУИР

ВВЕДЕНИЕ

На пути к достижению комплексного подхода при разработке программных средств (ПС) широкое применение получили CASE-продукты (Computer Aided Software Engineering), обеспечивающие поддержку многочисленных технологий проектирования информационных систем, охватывая всевозможные средства автоматизации и весь жизненный цикл (ЖЦ) программного обеспечения (ПО). Диапазон CASE-средств очень велик, и сегодня практически каждое из них располагает мощной инструментальной базой.

CASE-технология включает в себя методологию анализа, проектирования, разработки и сопровождения сложных систем ПО, поддержанную комплексом взаимосвязанных средств автоматизации. Главная цель CASE-подхода – разделить и максимально автоматизировать все этапы разработки ПС. Основные преимущества применения CASE-средств:

– улучшение качества ПО за счет автоматического контроля проекта;

– возможность быстрого создания прототипа будущей системы, что позволяет уже на ранних стадиях разработки оценить результат;

– ускорение процессов проектирования и программирования;

– освобождение разработчиков от выполнения рутинных операций;

– возможность повторного использования ранее созданных компонентов.

Методы объектно-ориентированного анализа и проектирования включают в себя язык моделирования и описание процессов моделирования. Язык моделирования UML (Unified Modeling Language) – это нотация, которая используется методом для описания проектов. Основная идея UML – это возможность моделировать ПО и другие системы как наборы взаимодействующих объектов. UML не привязан к какой-либо конкретной методологии или ЖЦ и может использоваться со всеми существующими методологиями. В настоящее время UML принят в качестве стандартного языка моделирования и получил широкую поддержку в индустрии ПО. Язык взят на вооружение самыми известными производителями ПО: IBM, Microsoft, Hewlett-Packard, Oracle. Большинство современных CASE-средств разрабатывается на основе UML.

На сегодняшний день практически все ведущие компании–разработчики технологий и программных продуктов располагают развитыми технологиями создания ПО. В пособии рассмотрена одна из признанных технологий, претендующих на роль мирового корпоративного стандарта, – Rational Unified Process (RUP), а также объектно-ориентированное CASE-средство Rational XDE. Главное отличие это средства от своего предшественника, программы Rational Rose, – полная интеграция с платформой Microsoft Visual Studio.NET, позволяющая в одной оболочке работать как с моделями программной системы, так и с кодом. За счет этого заметно сократилось время синхронизации модели и программного кода, в результате чего повысилась производительность рабочей станции.

Библиотека БГУИР

1. СОВРЕМЕННЫЕ ТЕХНОЛОГИИ СОЗДАНИЯ ПРОГРАММНЫХ СРЕДСТВ

1.1. Технология Rational Unified Process

RUP-технология в значительной степени соответствует стандартам и нормативным документам, связанным с ЖЦ ПО [1]. Ее основными принципами являются:

– итерационный и инкрементный подход к созданию ПО;

– планирование и управление проектом на основе функциональных требований к системе;

– построение системы на базе архитектуры ПО.

Первый принцип является определяющим. В соответствии с ним разработка системы выполняется в виде нескольких краткосрочных мини-проектов фиксированной длительности (от 2 до 6 недель), называемых итерациями. Каждая итерация включает в себя свои собственные этапы анализа требований, проектирования, реализации, тестирования, интеграции и завершается созданием работающей системы.

Итерационный цикл основывается на постоянном расширении и дополнении системы в процессе нескольких итераций с периодической обратной связью и адаптацией добавляемых модулей к существующему ядру системы. Система постоянно разрастается, поэтому такой подход называют итерационным и инкрементным. На рис. 1.1 показано общее представление RUP в двух измерениях. Горизонтальное измерение представляет время, отражает динамические аспекты процессов и оперирует такими понятиями, как стадии, итерации и контрольные точки. Вертикальное измерение отражает статические аспекты процессов и оперирует такими понятиями, как виды деятельности (технологические операции), рабочие продукты, исполнители и дисциплины (технологические процессы).

Согласно RUP ЖЦ ПО разбивается на отдельные циклы, в каждом из которых создается новое поколение продукта. Каждый цикл в свою очередь разбивается на четыре последовательные стадии [2]:

– начальная стадия;

– стадия разработки;

– стадия конструирования;

– стадия ввода в действие.

Каждая стадия завершается в четко определенной контрольной точке. В этот момент времени должны достигаться важные результаты и приниматься критически важные решения о дальнейшей разработке.

Начальная стадия может принимать множество разных форм. Для крупных проектов она связана с всесторонним изучением всех возможностей реализации проекта. В это же время разрабатывается бизнес-план проекта: определяется, сколько приблизительно он будет стоить и какой доход принесет. Кро-

Библиотека БГУИР

ме того, выполняется начальный анализ для оценки размеров проекта. Результатами начальной стадии являются:

– общее описание системы: основные требования к проекту, его характеристики и ограничения;

– начальная модель вариантов использования;

– начальный проектный глоссарий;

– начальный бизнес-план;

– план проекта, отражающий стадии и итерации;

– один или несколько прототипов.

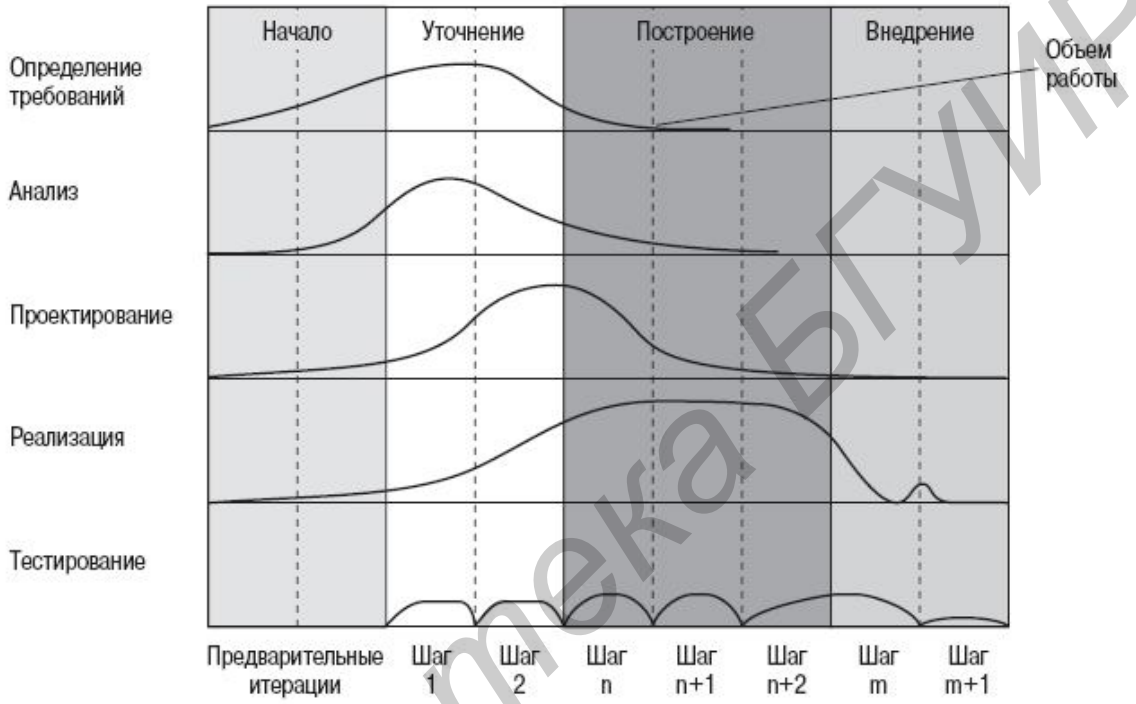

Рис. 1.1. Общее представление RUP

На стадии разработки выявляются более детальные требования к системе, выполняется высокоуровневый анализ предметной области и проектирование для построения базовой архитектуры системы, создается план конструирования и устраняются наиболее рискованные элементы проекта. Результатами стадии разработки являются:

– завершенная модель вариантов использования, определяющая функциональные требования к системе;

– перечень дополнительных требований, включая требования нефункционального характера и требования, не связанные с конкретными вариантами использования;

– описание базовой архитектуры будущей системы;

– работающий прототип;

– уточненный бизнес-план;

Библиотека БГУИР

– план разработки всего проекта, отражающий итерации и критерии оценки для каждой итерации.

Самым важным результатом стадии разработки является описание базовой архитектуры будущей системы. Эта архитектура включает в себя:

– модель предметной области, которая отражает понимание бизнеса и служит отправным пунктом для формирования основных классов предметной области;

– технологическую платформу, определяющую основные элементы технологии реализации системы и их взаимодействие.

Созданная архитектура является основой всей дальнейшей разработки. В будущем неизбежны незначительные изменения в деталях архитектуры, однако серьезные изменения маловероятны. Стадия разработки занимает около пятой части общей продолжительности проекта. Основными признаками ее завершения являются следующие:

– разработчики в состоянии оценить, сколько времени потребуется на реализацию каждого варианта использования;

– идентифицированы все наиболее серьезные риски и степень понимания наиболее важных из них такова, что известно, как справиться с ними.

Стадия конструирования напрямую связана с проработкой итераций. На стадии конструирования они являются одновременно инкрементными и повторяющимися. Инкрементность связана с добавлением новых конструкций к вариантам использования, реализованным во время предыдущих итераций. Повторяемость относится к разрабатываемому коду: на каждой итерации некоторая часть существующего кода переписывается с целью сделать его более гибким. Результатом стадии конструирования является продукт, готовый к передаче конечным пользователям и содержащий следующее:

– ПО, интегрированное на требуемых платформах;

– руководства пользователя;

– описание текущей реализации.

Стадия ввода в действие связана с передачей готового продукта в распоряжение пользователей. Она включает в себя:

– бета-тестирование, позволяющее убедиться, что новая система соответствует ожиданиям пользователей;

– параллельное функционирование с существующей системой, которая подлежит постепенной замене;

– конвертирование баз данных;

– оптимизацию производительности;

– обучение пользователей и специалистов службы сопровождения.

Статический аспект RUP представлен четырьмя основными элементами: роли; виды деятельности; рабочие продукты; дисциплины.

Роль определяет поведение и ответственность личности или группы личностей, составляющих проектную команду. Одна личность может играть в проекте много различных ролей.

Библиотека БГУИР

Под видом деятельности конкретного исполнителя понимается единица выполняемой им работы. Вид деятельности соответствует понятию технологической операции. Он имеет четко определенную цель, обычно выражаемую в терминах получения или модификации некоторых рабочих продуктов, таких, как модель, элемент модели, документ, исходный код или план. Каждый вид деятельности связан с конкретной ролью. Продолжительность вида деятельности составляет от нескольких часов до нескольких дней, он обычно выполняется одним исполнителем и порождает только один или весьма небольшое количество рабочих продуктов. Любой вид деятельности должен являться элементом процесса планирования. Примерами видов деятельности могут быть планирование итерации, определение вариантов использования и действующих лиц, выполнение теста на производительность. Каждый вид деятельности сопровождается набором руководств, представляющих собой методики выполнения технологических операций.

Дисциплина соответствует понятию технологического процесса и представляет собой последовательность действий, приводящую к получению значимого результата.

В рамках RUP определены шесть основных дисциплин: построение бизнес-моделей; определение требований; анализ и проектирование; реализация; тестирование; развертывание.

Имеется три вспомогательные дисциплины: управление конфигурацией и изменениями; управление проектом; создание инфраструктуры.

1.2. Унифицированный язык моделирования UML

Создание UML началось в 1994 г., в 1995 г. появилась первая спецификация языка, а в настоящее время UML является общепризнанным стандартом моделирования. В UML-модели есть два аспекта [3]:

– статическая структура – описывает, какие типы объектов важны для моделирования системы и как они взаимосвязаны;

– динамическое поведение – описывает ЖЦ этих объектов и то, как они взаимодействуют друг с другом для обеспечения требуемой функциональности системы.

Основные цели создания унифицированного языка моделирования:

1. Предоставить пользователям готовый к применению выразительный язык визуального моделирования, позволяющий разрабатывать осмысленные модели и обмениваться ими.

2. Предусмотреть механизмы для расширения базовых концепций.

3. Обеспечить независимость UML от конкретных языков программирования и процессов разработки.

4. Создать формальную основу для понимания языка моделирования.

5. Стимулировать рост рынка объектно-ориентированных инструментальных средств.

6. Интегрировать лучший практический опыт.

Библиотека БГУИР

Семантика языка UML представляет собой некоторую метамодель, которая определяет абстрактный синтаксис и семантику понятий объектного моделирования на языке UML. Семантика определяется для двух видов объектных моделей: структурных моделей и моделей поведения. Структурные модели, известные также как статические модели, описывают структуру сущностей или компонентов некоторой системы, включая их классы, интерфейсы, атрибуты и отношения. Модели поведения, называемые иногда динамическими моделями, описывают поведение или функционирование объектов системы, включая их методы, взаимодействие и сотрудничество между ними, а также процесс изменения состояний отдельных компонентов и системы в целом.

Для решения столь широкого диапазона задач моделирования разработана достаточно полная семантика для всех компонентов графической нотации. Требования семантики языка UML конкретизируются при построении отдельных видов диаграмм.

1.3. Основные окна и пункты меню программы Rational XDE

Рассмотрим программное средство Rational XDE – один из признанных лидеров на мировом рынке CASE-продуктов. Программа потеряла часть своей универсальности по сравнению с Rational Rose [4, 5]. В основном потери касались возможности генерации программного кода практически для любых языков программирования. В Rational XDE для .NET возможна синхронизация модели и кода только для языков, которые поддерживаются Microsoft Visual Studio.NET. При этом Rational XDE позволяет создавать Free Form, в которых не отслеживается нотация UML и которые могут включать в себя значки из разных UML-диаграмм [6].

Язык C#, на котором генерируется код в среде Rational XDE, разработан компанией Microsoft для платформы .NET. Она представляет собой обширную библиотеку классов, инфраструктуру и инструментальные средства для создания межплатформенных, не зависящих от языка программирования приложений. На платформе .NET создана ASP.NET – технология активных серверных страниц. В приложении ASP.NET доступна вся библиотека .NET Framework, значительно ускоряющая и облегчающая разработку сложных сетевых программных систем [7].

На рис. 1.2 показано главное окно программы Rational XDE. На его внешнем виде отразилась интеграция с Microsoft Visual Studio.NET. В результате рабочий стол Visual Studio.NET изменился и на него добавились окна, отвечающие за моделирование программной системы. В центре экрана расположено окно документов, в котором можно открывать код, ресурсы и диаграммы UML, создаваемые в модели. Справа в окне Explorer добавилась закладка Model Explorer, позволяющая перемещаться по модели. Под ней – окно XDE Code Properties, которое показывает свойства выбранной диаграммы. Внизу – закладка Model Documentation, отражающая документацию модели. Слева, в окне Tool-

Библиотека БГУИР

box, добавилось большое количество инструментов, необходимых для работы с UML-моделями в отдельных разделах, которые появляются в момент активизации одной из диаграмм [7].

Основная работа с элементами модели осуществляется при помощи окон Model Explorer и рабочего стола диаграммы. При этом за перемещение по модели отвечает Model Explorer, а редактирование лучше выполнять на рабочем столе. Одним из отличий Rational XDE от предыдущей версии явилось использование окна Toolbox, содержащего дополнительные инструменты для работы над проектом. Строки инструментов (Toolbar) также остались, но их роль сократилась до управления основными режимами, а создание новых элементов теперь производится с использованием Toolbox.

Model Explorer позволяет осуществлять навигацию по элементам модели, представленным в иерархическом виде. Из контекстного меню можно добавлять, удалять и изменять как диаграммы UML, так и элементы диаграмм, синхронизировать исходный код и диаграммы, работать с шаблонами кода, т.е. полностью управлять созданием модели программной системы. Model Explorer представляет собой аналог стандартного обозревателя Windows.

В окне Model Documentation отображается текст документации, относящийся к модели, диаграмме или выделенному элементу на ней. Окно документации представляет собой текстовый редактор, позволяющий описывать цели создания, поведение и другую информацию. Документация в Rational XDE обновляется вместе с моделью. В случае если разработчик внес в исходный код комментарий, а после этого модель обновилась, то все комментарии сразу же будут отображены в окне документации.

Окно Toolbox сменило строку инструментов Toolbar. С помощью Toolbox выполняются основные функции по работе с элементами на диаграмме.

Аналогично тому, как это происходило в Rational Rose, в среде Rational XDE все действия над объектами выполняются посредством контекстного меню. Оно зависит от набора функций, доступных для применения к конкретному объекту. При установке Rational XDE добавляет свои пункты в главное меню Visual Studio .NET, изменяет некоторые пункты, установленные по умолчанию, а также добавляет свои строки инструментов. Это связано с тем, что после установки Rational XDE в среде Visual Studio .NET можно работать с графическими диаграммами, а не только с программным кодом и ресурсами проекта. Рассмотрим пункты меню, добавленные Rational XDE, и предоставляемые ими возможности по работе с моделями или их элементами.

Modeling. Этот раздел меню предназначен для добавления диаграмм и их элементов в проект. Меню *Modeling* показано на рис. 1.3. Пункты моделирования позволяют добавлять в проект UML-элементы или диаграммы; проверять корректность диаграмм; проверять из проекта ссылки на другие модели; исправлять ошибки во внешних ссылках; устанавливать пути доступа к файлам модели; использовать шаблоны при построении модели.

Diagram. Этот раздел меню предназначен для общего управления значками на диаграммах. Меню *Diagram* показано на рис. 1.4. Его пункты позволяют: ав-

Библиотека БГУИР

томатически расставить значки на текущей диаграмме; изменить на диаграмме положение выделенного элемента; переместить фокус просмотра на выделенный элемент диаграммы; добавить на диаграмму элементы, связанные с выделенным; активизировать окно настройки визуализации соединений; работать с надписями на стрелках соединителях; управлять перерисовкой диаграммы и найти отмеченный на ней элемент [7].

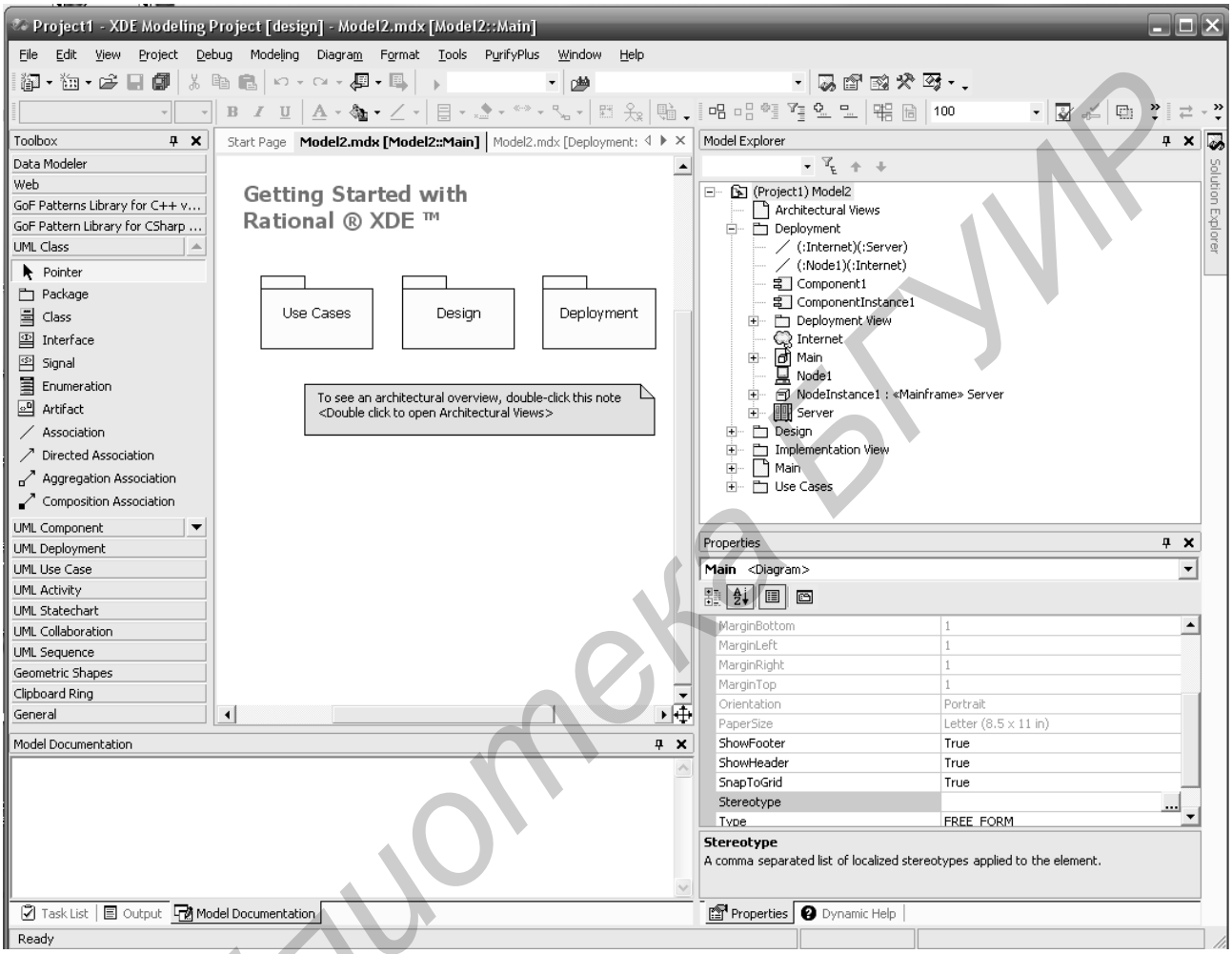

Рис. 1.2. Главное окно Rational XDE

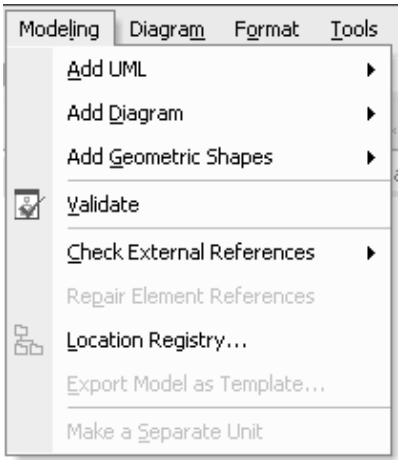

Рис. 1.3. Меню Modeling

Библиотека БГУИР

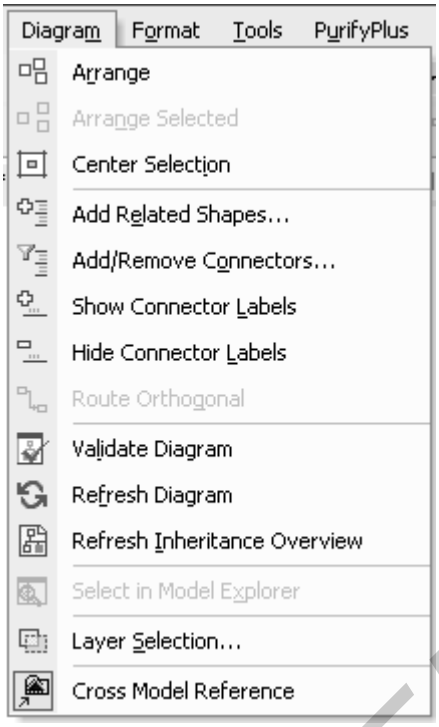

Рис. 1.4. Меню Diagram

Format. Этот раздел меню позволяет управлять форматированием диаграммы. Меню *Format* показано на рис. 1.5. Его пункты позволяют показать или скрыть список атрибутов, операций и сигналов в классе; показать или скрыть сигнатуры операций или сигналов; изменять стиль визуализации элементов диаграмм и их стереотипов; изменять стиль выделенной линии; поддерживать заданный размер элемента диаграммы; включить или отключить показ имени родительского контейнера для выделенного элемента; установить одинаковые настройки для всех выделенных элементов диаграммы.

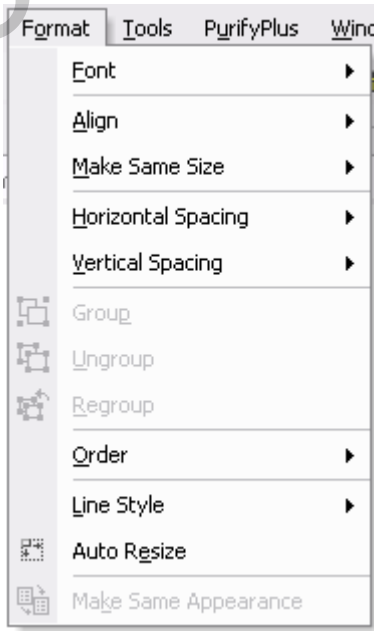

Рис. 1.5. Меню Format

Библиотека БГУИР

2. ЛАБОРАТОРНЫЕ РАБОТЫ

2.1. Лабораторная работа №1

АНАЛИЗ ТРЕБОВАНИЙ И ПОСТРОЕНИЕ ДИАГРАММЫ USE CASE

Цель работы:

- научиться строить диаграммы Use Case в среде автоматизированного синтеза Rational XDE;
- разработать диаграмму Use Case для проектируемой системы.

Задание:

описать функциональные требования к системе и представить сценарии поведения ее объектов с помощью диаграммы Use Case.

Порядок выполнения работы

1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.
3. Построить диаграмму Use Case по предложенной тематике.

Построение диаграммы Use Case

Моделирование системы начинается с анализа требований к ней, что напрямую связано с определением прецедентов в системе. Прецеденты и связи между ними могут быть представлены с помощью диаграммы Use Case.

Рассмотрим построение модели виртуального книжного магазина, начав с выделения актеров и их ролей в системе.

1. Покупатель книги – любой пользователь сети интернет, зарегистрировавшийся в магазине как покупатель.

2. Администратор магазина – работник, который проверяет наличие заказов пользователей, формирует их, если они есть на складе, и отправляет покупателям с посыльным.

3. Директор магазина – получает отчет о заказанных и отправленных потребителям книгах.

Тогда можно описать функции, которые должна выполнять программа.

1. Просмотр книжного каталога (все пользователи).
2. Регистрация пользователей.
3. Работа с корзиной покупателя (зарегистрированные пользователи).
4. Оформление заказа на покупку (зарегистрированные пользователи).
5. Просмотр статуса заказа (для зарегистрированных пользователей).
6. Изменение статуса заказа (для администратора).
7. Просмотр списка заказов (для руководителя).

Библиотека БГУИР

8. Редактирование книжного каталога (для администратора).

9. Изменение данных пользователя (для администратора и зарегистрированного пользователя).

Все диаграммы в Rational XDE создаются в модели. Ее построение начинается с создания проекта приложения, для чего необходимо выполнить следующие действия [7].

1. Выбрать пункт меню *File=>New.*

2. В предложенном окне выбрать *Blank Model* и создать пустую модель, после чего в нее можно добавлять диаграммы.

3. Из контекстного меню модели выбрать пункт *Add Diagram.*

4. Из подменю выбрать *Add Use Case*.

5. Переименовать добавленную диаграмму в *Прецеденты*.

В Rational XDE не включены специальные значки для бизнес-анализа, что отражает сокращенный процесс разработки .NET приложений, однако это не мешает полноценно использовать диаграмму прецедентов для определения требований к системе. Все инструменты диаграммы Use Case активны и находятся в окне *Toolbox*. Их набор приведен на рис. 2.1.

Значок *Access* позволяет показать зависимость одного элемента диаграммы от другого на уровне доступа.

Значок *Include* отражает прецедент, являющийся частью главного прецедента.

Значок *Import* показывает зависимость одного элемента диаграммы от другого, когда один элемент импортирует информацию из другого.

Значок *Extend* отражает расширение прецедента и используется, чтобы показать часть главного прецедента, который обрабатывается не всегда, а в определенных случаях или при определенных условиях.

Значком *Association* обозначают простые связи между элементами.

Рис. 2.1. Toolbox для диаграммы Use Case

Библиотека БГУИР

Значок *Direct Association* позволяет обозначать направленные связи между элементами. Эта связь более сильная, чем простая ассоциация, и позволяет точнее показать отношения между элементами на диаграмме. Например, отразить актера, который инициализирует прецедент. Чтобы преобразовать *Association* в *Direct Association*, необходимо выполнить следующее:

1. Выделить связь.

2. Из контекстного меню связи выбрать пункт *Properties Window*.

3. В окне свойств перейти к строке *End2Navigable* и изменить ее значение на *False*.

Значок *Constraint* применяется для указания ограничений, налагаемых на прецеденты.

Значок *Constraint Attachment* позволяет соединить элемент *Constraint* с любым элементом на диаграмме.

На основе ранее описанных ролей актеров и функций системы строится диаграмма прецедентов. Ее окончательный вариант приведен на рис. 2.2.

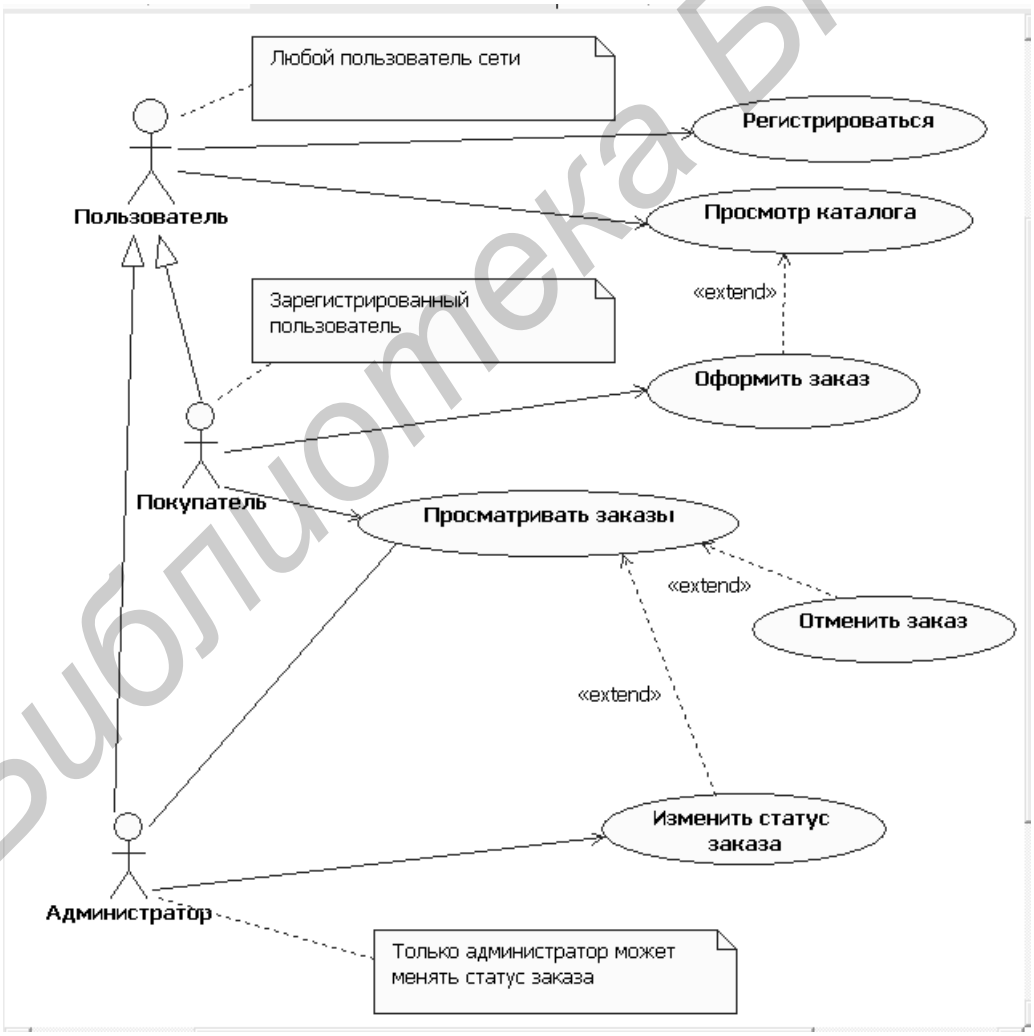

Рис. 2.2. Диаграмма Use Case (прецедентов)

Библиотека БГУИР

Здесь показаны три актера и их роли. Пользователь может зарегистрироваться и просматривать каталог. После регистрации он становится покупателем и может оформлять, просматривать и отменять заказы. Администратор также может просматривать заказы или менять их статус. На диаграмме использованы направленные ассоциации для отражения инициализации прецедентов актерами и простые ассоциации – в остальных случаях. Прецеденты оформления заказа, изменения статуса и удаления являются расширением соответствующих прецедентов, что отражено связью *Extend*.

Контрольные вопросы

1. Какой вариант не позволит создать новый элемент Actor на диаграмме прецедентов:

а) находясь в окне диаграммы Use Case, выбрать из контекстного меню Add UML => Actor;

б) щелкнуть по значку Actor в Toolbox, а затем на диаграмме;

в) находясь в окне Model Explorer, из контекстного меню диаграммы Use Case выбрать Add UML => Actor.

2. Есть прецедент, который инициализируется актером. Какой тип связи обычно используется для их соединения:

а) Association; б) Direct Association; в) Dependency.

3. Есть два прецедента, один из которых возникает при определенных обстоятельствах при выполнении другого. Какую связь нужно использовать для отражения такого взаимодействия:

а) Include; б) Dependency; в) Extend.

2.2. Лабораторная работа №2

АНАЛИЗ УСТРОЙСТВ СРЕДСТВАМИ ДИАГРАММЫ DEPLOYMENT

Цель работы:

- научиться строить диаграммы Deployment CASE- среде Rational XDE;
- разработать Deployment для проектируемой прикладной системы.

Задание:

с помощью диаграммы Deployment проанализировать и спроектировать аппаратную конфигурацию, на которой будут работать отдельные компоненты и Web-службы, а также описать их взаимодействие между собой.

Порядок выполнения работы

1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.
3. Построить диаграмму Deployment по предложенной тематике.

Библиотека БГУИР

Описание диаграммы Deployment

Диаграмма *Deployment* предназначена для анализа аппаратной части системы. Для распределенных систем, какими часто бывают Web-приложения, этот тип диаграмм очень важен, поскольку именно здесь определяется, на каком сервере сети будет работать конкретный компонент или Web-служба и с какими другими сетевыми устройствами будет осуществляться взаимодействие. Rational XDE позволяет наглядно показать топологию сети, поскольку в него включены стереотипы большинства распространенных сетевых устройств. В случае моделирования интернет-магазина особое внимание будет уделено коммуникациям серверов и компьютеров клиентов.

Для создания диаграммы необходимо из контекстного меню модели выбрать пункт *Add=>Deployment*.

Разработка диаграммы начинается с выбора архитектуры создаваемого приложения. Будут использоваться два сервера. Первый предназначен для работы *Internet Information Server (IIS)* и программного обеспечения *.NET Framework*. На втором будет расположена база данных. Для приложения планируется многоуровневая архитектура, т. е. логика работы с базой данных, логика приложения и логика представления будут разделены.

Инструменты диаграммы Deployment показаны на рис. 2.3.

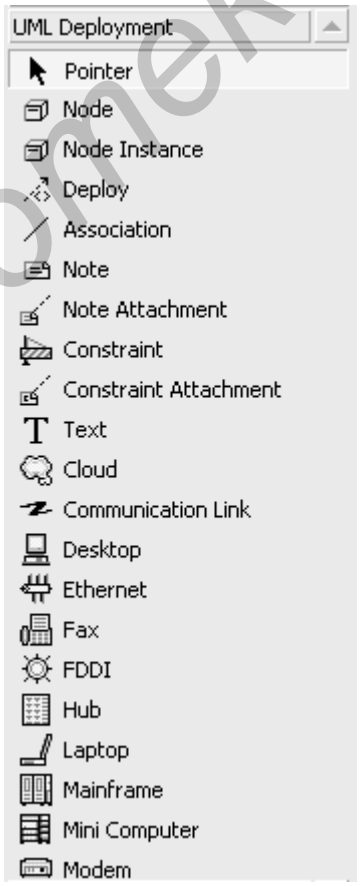

Рис. 2.3. Toolbox диаграммы Deployment

Библиотека БГУИР

Несмотря на обилие значков диаграммы Deployment, фактически это лишь представление двух видов значков с различными стереотипами и их связей. Главные устройства – это *Node* (узел) и *Node Instance* (реализация узла). На данном этапе проектирования можно использовать только значки с типом Node, поскольку пока определяется только возможная конфигурация сети, а не ее конкретная реализация. Кроме того, в системе планируется использование компьютеров клиентов. Для их определения также можно использовать три элемента с типом Node: компьютеры клиента, руководителя и администратора. Планируется подключение к серверу клиентских машин посредством связи через интернет. Все элементы на диаграмме свяжем соединением типа Association.

Для того чтобы диаграмма выглядела более выразительной и легче читалась, можно изменить стереотипы узлов таким образом, чтобы они максимально соответствовали своему назначению. Удобнее всего это выполнить с помощью окна *Properties Window*, используя свойство *Stereotype*. Тогда для представления PC клиента и администратора выбираем элемент *Desktop*; для изображения PC руководителя – элемент *Laptop*; серверы отображаются с помощью элемента *Tower*. Чтобы показать, что серверы и компьютеры администратора и руководителя находятся во внутренней сети предприятия, выбираем элемент *Hub*; для изображения сети интернет используем значок *Cloud* и разместим его между компьютером клиента и сервером. В результате диаграмма *Deployment* примет вид, показанный на рис. 2.4.

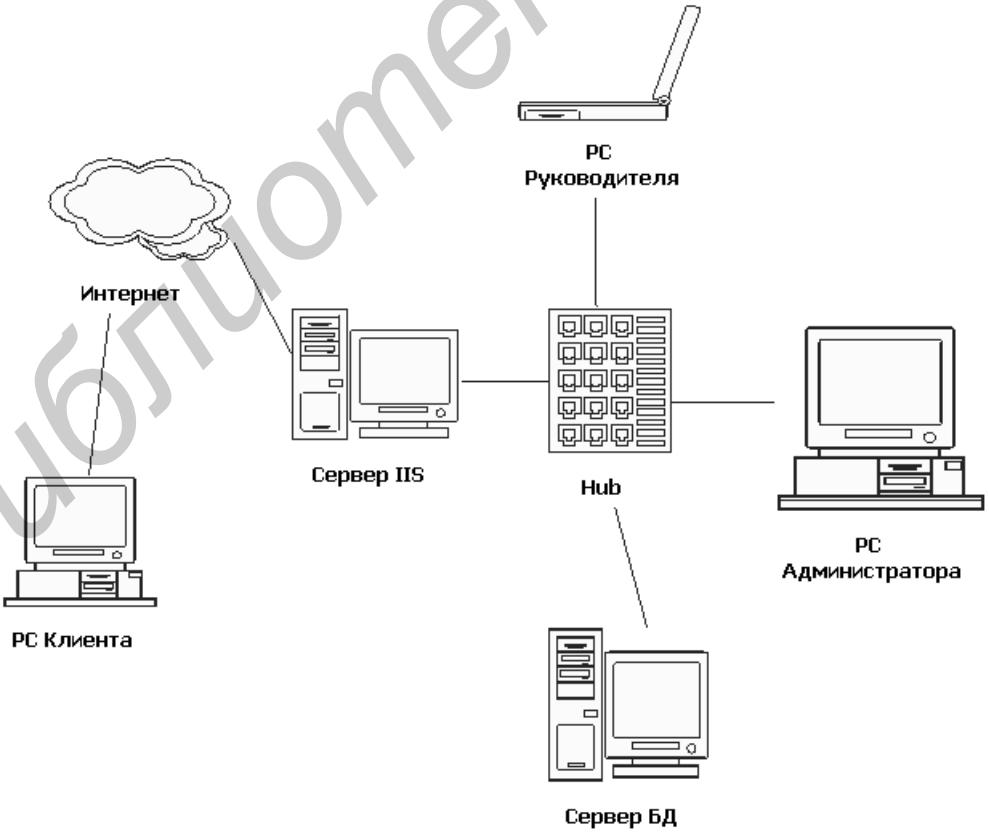

Рис. 2.4. Диаграмма Deployment

Библиотека БГУИР

Контрольные вопросы

1. В чем заключается принципиальное отличие диаграммы Deployment от остальных диаграмм UML?

2. Почему в большинстве случаев элементы на диаграмме Deployment соединены связью Association, не имеющей направления?

3. Почему для большинства систем разрабатывается только одна диаграмма Deployment?

2.3. Лабораторная работа №3

СОЗДАНИЕ МОДЕЛИ ПОВЕДЕНИЯ СИСТЕМЫ ПРИ ПОМОЩИ ДИАГРАММ STATECHART И ACTIVITY

Цель работы:

- научиться строить диаграммы Statechart и Activity в среде автоматизированного синтеза Rational XDE;
- разработать диаграммы Statechart и Activity для проектируемой прикладной системы.

Задание:

1. С помощью диаграммы Statechart описать состояния объектов системы и условия переходов между ними. Добавить в модель классы, на основе которых будут создаваться исследуемые объекты.

2. С помощью диаграммы Activity промоделировать действия объектов проектируемой системы.

Порядок выполнения работы

1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.
3. Построить диаграммы Statechart и Activity по предложенной тематике.

Описание диаграммы Statechart

Объектно-ориентированные системы в каждый момент времени находятся в определенном состоянии, зависящем от состояния каждого входящего в нее объекта, поэтому при проектировании важно уметь описывать состояние системы через состояния входящих в нее объектов.

Согласно теории конечных автоматов любую сложную машину можно разложить на простые автоматы, имеющие определенные состояния, поэтому в объектно-ориентированных программных системах этот подход действительно оправдан. Кроме моделирования поведения самих объектов, диаграмма состоя-

Библиотека БГУИР

ний может применяться для конкретизации прецедентов, что отражает взгляд на поведение объектов со стороны. Будем использовать эту диаграмму для описания состояний приложения в целом. Согласно концепции создания приложения в .NET любая программа должна иметь главный объект, следовательно, выбранный подход не противоречит правилам и стилю разработки интернет-приложений при помощи .NET Framework.

Хотя назначение диаграммы *Statechart* и принципы ее построения не изменились в Rational XDE по сравнению с Rational Rose, диаграмма состояний в Rational XDE дополнилась новыми возможностями.

Для создания диаграммы необходимо из контекстного меню модели выбрать пункт *Add=> Statechart*. Toolbox для нее показан на рис. 2.5.

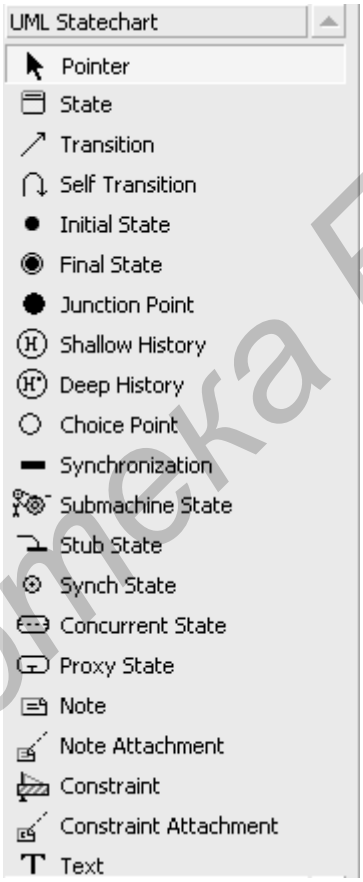

Рис. 2.5. Toolbox для *Statechart*

Поскольку этот тип диаграммы отображает состояние объектов, то на ней должно явно отображаться место, где происходит инициализация или создание объекта и начало его работы. Значок *Initial State* позволяет обозначить событие, которое переводит объект в первое состояние на диаграмме. Это будет обозначением начала работы виртуального магазина. Для Web-приложения обычным началом работы является активизация по запросу пользователя начальной страницы. Таким образом, магазин доступен через интернет в любое время, а система начинает работать только по запросу пользователя, и после того, как последний пользователь покидает виртуальный магазин, работа заканчивается.

Библиотека БГУИР

После запроса браузером пользователя начальной страницы система переходит в первое состояние: ожидание дальнейших команд пользователя. После начала работы система переходит в состояние ожидания выбора пользователя. Поэтому с помощью элемента *State* создается одноименное состояние и связывается стрелкой *Transition* с начальным состоянием. Значок *State* позволяет отразить на диаграмме состояние или ситуацию в течение времени жизни программного объекта, которая отвечает некоторому положению объекта или ожиданию им внешнего события.

На диаграмме создан элемент *Final State* (завершающее состояние) и соединен с элементом состояния. Завершение работы означает, что все внутренние процессы, входящие в состояние, должны быть завершены. Предположим, что переход в финальное состояние происходит по следующему условию: если пользователь в течение десяти минут не предпринимал никаких действий, работа приложения завершается. Чтобы отразить это на диаграмме, необходимо выполнить следующие действия:

1. Из контекстного меню элемента *Ожидание выбора пользователя* выбрать пункт *Collections*.
2. Перейти во вкладку *Outgoing Transition*.
3. Найти строку перехода, которая отмечена значком *Final State*.
4. Ввести в поле *Guard Condition* строку *StateTime* > 10 min.
5. Нажать кнопку *Close* для сохранения изменений.

Рассмотрим посылку сообщений на диаграмме *Statechart*. Для этого добавлено состояние *Регистрация нового пользователя,* которое соединено стрелкой перехода с состоянием *Ожидание выбора пользователя.* Затем из контекстного меню перехода активизирован пункт *Collection*, из него выполнен переход во вкладку *Triggers*, и в поле *Name* введено название сообщения *OnUserRegister*. Затем создано состояние *Просмотр каталога* и соединено с состоянием *Ожидание выбора пользователя*. Для состояния *Просмотр каталога* выбран элемент *SelfTransition* (переход на себя), затем в окне *Collection* для элемента *SelfTransition* задано имя сообщения *OnChangeView*. На рис. 2.6 приведен фрагмент диаграммы *Statechart*.

Перечислим назначение остальных инструментов диаграммы *Statechart*.

Shallow History (неглубокая история) позволяет создавать значок, показывающий, что данное состояние должно отслеживать историю переходов.

Deep History (неглубокая история) похож на предыдущий, однако показывает, что необходимо восстановить последнее состояние любого уровня вложенности, а не последнего, как *Shallow History*.

Submachine State (вложенное состояние) создает элемент, показывающий вложенные состояния.

Stub State (состояние-заглушка) создает элемент, отражающий наличие скрытых вложенных состояний, в которые направлен переход.

Concurrent State (параллельные состояния) создает элемент, отражающий параллельные состояния. По умолчанию в элементе создаются две области, в которых можно задавать состояния.

Библиотека БГУИР

Junction Point (точка объединения) обозначает на диаграмме точку, в которой соединяются несколько переходов из различных состояний.

Choice Point (точка выбора) показывает на диаграмме точку, из которой могут возникнуть несколько переходов в различные состояния.

Synchronizaiton (синхронизация) показывает на диаграмме точку синхронизации, в которой соединяются несколько переходов, и дальнейшего перехода не происходит, пока не завершатся все входящие состояния.

Synch State (состояние синхронизации) показывает на диаграмме точку синхронизации между двумя параллельными процессами.

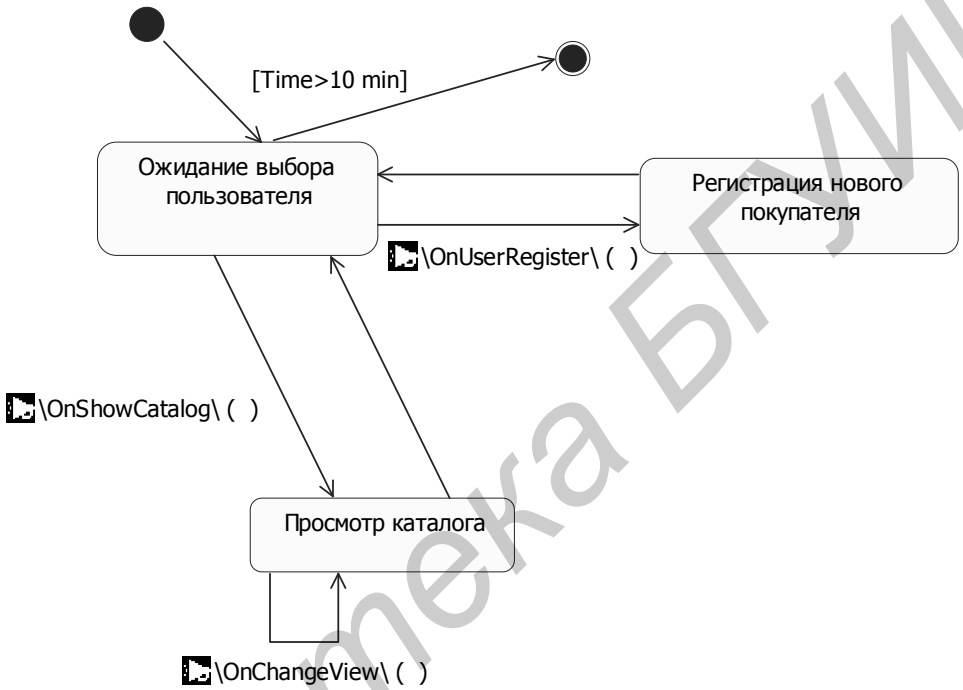

Рис. 2.6. Фрагмент диаграммы Statechart

Описание процессов системы с помощью диаграммы Activity

Диаграмму *Activity* можно считать разновидностью диаграммы состояний, рассмотренной в предыдущей лабораторной работе. Однако в отличие от диаграммы *Statechart*, описывающей состояния объекта и переходы между ними, стандартным применением диаграммы деятельности является моделирование шагов какой-либо процедуры.

Диаграммы деятельности могут создаваться на всех стадиях разработки ПО. Перед началом работ с их помощью моделируются важные рабочие процессы предметной области с целью определения структуры и динамики бизнеса. На этапе обсуждения требований диаграммы деятельности используются для описания процессов и событий выявленных прецедентов. В течение фазы анализа и проектирования *Activity* помогает моделировать процесс операций объектов.

Библиотека БГУИР

Для создания диаграммы из контекстного меню модели необходимо выбрать пункт *Add Diagram => Activity*. В модель проекта добавится *Activity Graph*, так как фактически диаграмма деятельности представляет собой граф. На рис. 2.7 приведен *Toolbox* диаграммы *Activity*.

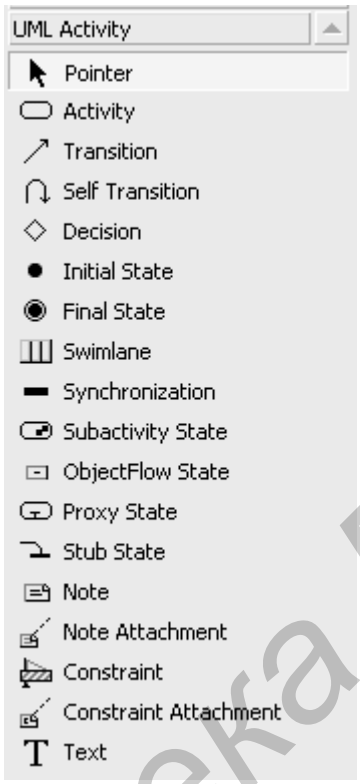

Рис. 2.7. Toolbox диаграммы Activity

Построение диаграммы начинается с элемента *Initial State*, который соединяется с элементом *Activity – Запрос бланка заказа*.

Для наглядности отображения элементов на диаграмме используется инструмент *Swimlane*. Он позволяет моделировать области ответственности исполнителей при описании процессов. На диаграмме выделены две плавательные дорожки: *Покупатель* и *Система*. В каждой из них отражаются объекты системы, выполняющие определенные действия в системе.

Значок *Synchronization* позволяет отображать момент разделения процесса. При этом действия разделяются на несколько, выполняемых независимо, и только по завершении всех действий объект продолжает работу. Для иллюстрации использования этого инструмента в модель добавлено несколько действий. После запроса покупателя система генерирует форму заказа, которая предлагается для заполнения покупателю. Он заполняет форму и передает ее системе, которая требует от покупателя подтверждения заказа по e-mail. В этот момент происходит разветвление процесса, и система ожидает подтверждения заказа, причем ожидание длится не более трех суток. Для отражения этого действию *Ожидание подтверждения* во вкладке *OutGoingTransition* в окне *Collections* установлено условие перехода *GuardCondition*. Получив подтверждение от поль-

Библиотека БГУИР

зователя, система изменяет статус заказа на *Подтверждено*. Здесь разделение потоков деятельности выполняется без дополнительных условий.

Для отражения разделения потоков деятельности по определенным условиям введен инструмент *Decision*. Условия перехода из него указываются в окне *Collection*. Для демонстрации использования значка *Decision* на диаграмму добавлено еще несколько действий. Окончательный вариант диаграммы *Activity* приведен на рис. 2.8.

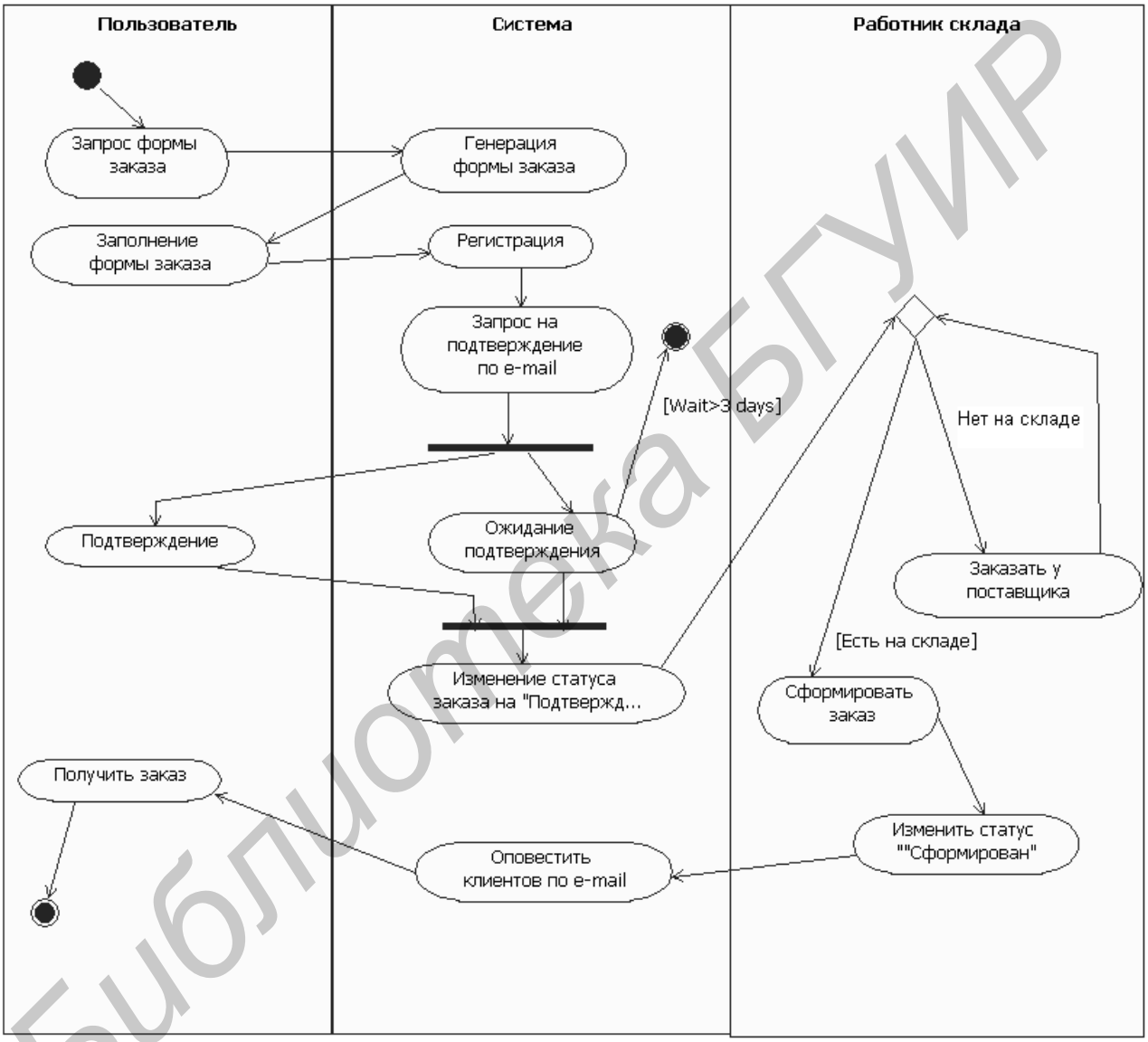

Рис. 2.8. Диаграмма Activity

Библиотека БГУИР

Рассмотрим инструменты *Activity*, не использованные в примере.

Элемент *Object Flow States* позволяет показать состояние объекта на диаграмме деятельности и отражает промежуточное состояние между входящим и исходящим видом деятельности. Элемент помогает моделировать процесс перевода объекта из одного состояния в другое.

Элемент *Sub Activity State* отражает на диаграмме вложенные состояния.

Инструмент *Sub State* позволяет создавать элемент, который показывает, что в данном состоянии есть скрытые вложенные состояния, в которые направлен переход.

Элемент *Proxy State* предназначен для создания элемента, который не подходит под стандарт UML. *Proxy State* необходим для отражения ссылки на элемент другой модели и помогает не копировать элементы в текущую модель в случае необходимости ссылки.

Контрольные вопросы

1. Связь Transition может быть установлена между элементами:

а) Initial State – State; б) Initial State – End State; в) верны все ответы.

2. Для обозначения ситуации, когда два параллельных процесса должны быть завершены, и дальнейшая работа не осуществляется, пока не будут завершены оба, необходимо использовать:

а) Synchronization; б) Junction Point; в) оба ответа правильны.

3. Какая ситуация вызовет ошибку при проверке диаграммы с помощью режима Validate

а) из значка Initial State выходят два перехода Transition;

б) из значка End State выходит один переход Transition;

в) все ответы правильны.

4. Как создать диаграмму деятельности?

а) из контекстного меню модели выбрать *Add Diagram => Activity*;

б) из главного меню выбрать *Modeling => Add Diagram => Activity*;

в) верны все варианты.

5. Какая диаграмма лучше подходит для конкретизации прецедентов?

а) Activity; б) Statechart; в) Deployment; г) верны все варианты.

6. Как отразить на диаграмме то, что несколько процессов должны быть завершены до перехода к следующему элементу деятельности?

а) использовать значок Proxy State.

б) использовать значок Decision.

в) использовать значок Swimlane.

г) ни один из перечисленных.

2.4. Лабораторная работа №4

ОПИСАНИЕ ВЗАИМОДЕЙСТВИЯ ОБЪЕКТОВ СИСТЕМЫ ПРИ ПОМОЩИ ДИАГРАММЫ SEQUENCE

Цель работы:

- научиться строить диаграмму Sequence в среде автоматизированного синтеза Rational XDE;
- разработать диаграмму Sequence для проектируемой системы.

Библиотека БГУИР

Задание:

с помощью диаграммы Sequence получить отражение во времени процесса обмена сообщениями между объектами создаваемой системы.

Порядок выполнения работы

1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.
3. Построить диаграмму Sequence по предложенной тематике.

Описание взаимодействия объектов при помощи диаграммы Sequence

Согласно нотации UML есть два типа диаграмм, которые позволяют моделировать взаимодействие объектов: *Sequence* – диаграмма последовательности и *Collaboration* – диаграмма кооперации. Первая акцентирует внимание на последовательности приема и передачи сообщений, а вторая – на структуре обмена сообщениями между отдельными объектами.

Rational XDE не поддерживает диаграмму *Collaboration* в том виде, в котором она присутствует в среде Rational Rose [8] . *Collaboration* нельзя создать как отдельную диаграмму. Проектирование ведется с использованием только диаграммы *Sequence*.

Для создания диаграммы из контекстного меню модели нужно выбрать *Add Diagram => Sequence*. В меню находятся два пункта для создания диаграммы – это *Sequence: Instance* (реализация) и *Sequence: Role* (роль). Эти пункты отражают различные возможности в использовании диаграммы.

В случае построения диаграммы *Sequence* в окне *Model Explorer* появится новый элемент, представляющий собой иерархию группы значков. И хотя Rational XDE не поддерживает диаграмму *Collaboration,* отображение взаимодействия происходит через элемент *Collaboration*. Он должен представлять реализацию одного из прецедентов системы и может иметь несколько альтернативных последовательностей действий. Главную из них Rational XDE создает сразу, альтернативные можно добавить при необходимости.

Диаграмма *Sequence: Role* отражает роли без ссылки на конкретные классы. Роль – это не класс или объект, а определенная последовательность в обмене сообщениями. Поэтому роли обычно используются для представления образцов. Если в системе задан определенный объект, то каждая роль ссылается на базовый класс, который идентифицируется атрибутами и операциями, нуждающимися во взаимодействии с заданным объектом. В случае использования образцов в системе любой класс, взаимодействующий с ними, должен быть отображен ролью в диаграмме. Замена объектов ролями позволяет использовать один класс для реализации нескольких ролей, которые могут отражаться в нескольких прецедентах. Имя роли отображается с косой чертой; в имени роли через двоеточие может быть представлено имя класса, реализующего эту роль.

На рис. 2.9 приведен *Toolbox* диаграммы *Sequence*.

Библиотека БГУИР

Значок *Lifeline Actor* позволяет создать на диаграмме роль актера с указанием линии жизни. Фактически на диаграмме создается отображение актера, а линия жизни дополняет это отображение.

Однако не всегда необходимо создавать новых актеров в диаграмме. В разрабатываемой модели уже созданы актеры *Пользователь*, *Покупатель* и *Администратор*, поэтому при моделировании реализации прецедента *Регистрация* можно воспользоваться уже созданными элементами.

Инструмент *LifeLine* позволяет создать на диаграмме роль для программного объекта с линией жизни. В отличие от *LifeLine Actor* при создании этого элемента класс не добавляется, а его место в названии остается пустым. Абстрактную роль *Система* будет иллюстрировать в модели взаимодействия незарегистрированного пользователя с виртуальным магазином.

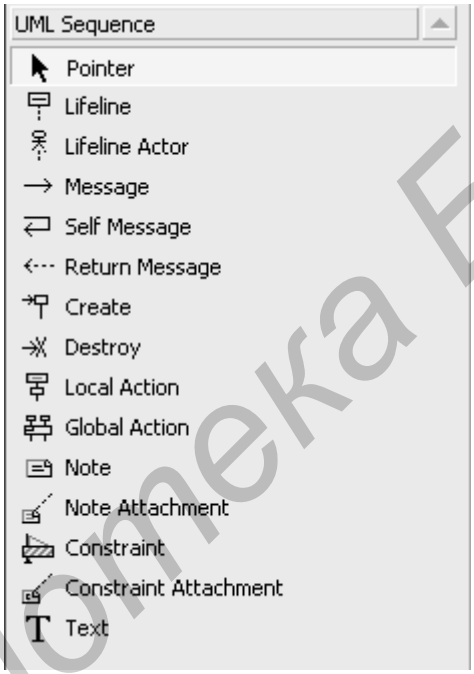

Рис. 2.9. Инструменты диаграммы *Sequence*

В диаграмме *Sequence: Instance* линия жизни может явно указать для объекта момент его создания и уничтожения.

Для отражения обмена сообщениями между незарегистрированным пользователем и окном регистрации создаются роли *Незарегистрированный пользователь* и *Окно регистрации*.

Инструмент *Message* позволяет создать отображение сообщения, передаваемого от одного объекта или роли к другому. Передача сообщения означает передачу управления объекту-получателю.

Значок *Return Message* показывает на диаграмме возврат управления из вызванной подпрограммы на сервере клиенту. На диаграмме такое сообщение отправляется от *Пользователя* к *Окну регистрации*.

Библиотека БГУИР

Значок *Create* предназначен для показа сообщения, при помощи которого один объект создает другой. При этом созданный объект не получает управления, т.е. фокус активности остается у отправителя сообщения.

Значок *Destroy* отражает момент уничтожения программного объекта.

Окончательный вариант диаграммы Sequence показан на рис. 2.10.

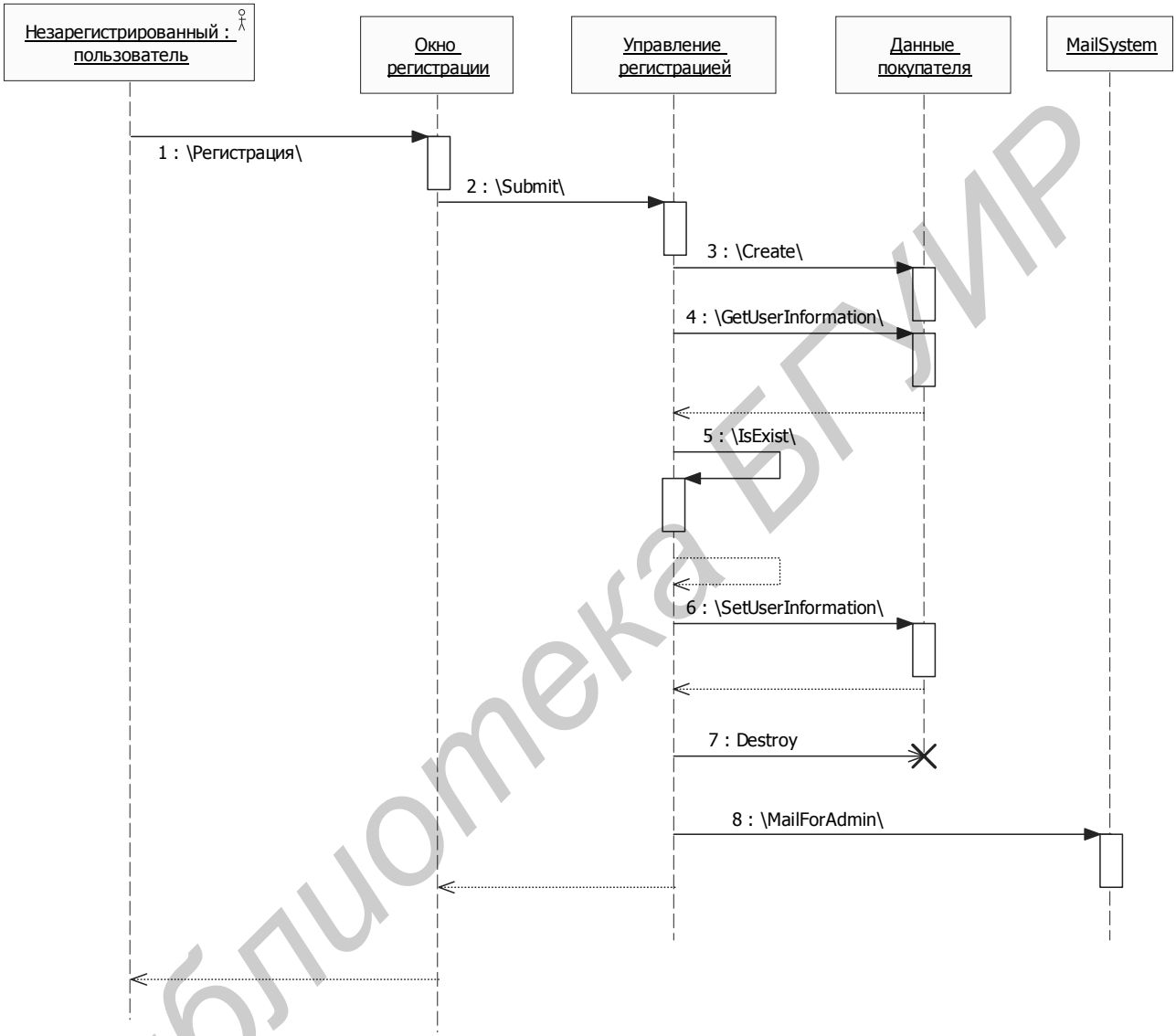

Рис. 2.10. Диаграмма Sequence

Здесь происходят следующие события. *Незарегистрированный пользователь* хочет зарегистрироваться и нажимает кнопку в *Окне регистрации*. Окно регистрации выдает сообщение Submit для объекта управляющего регистрацией, который, в свою очередь, создает объект доступа к данным, выдавая сообщение Create, а затем запрашивает у созданного объекта данные пользователя для проверки, не был ли зарегистрирован такой пользователь ранее. Это происходит внутри обработчика сообщения IsExist, которое объект выдает самому себе. После получения данных и удостоверения того, что записи об этом поль-

Библиотека БГУИР

зователе еще нет в системе, отправляется сообщение о сохранении данных о пользователе SetUserInformation, после чего объект уничтожается.

Сообщение MailFormAdmin дает команду на отправку оповещения администратору о том, что в системе зарегистрировался новый пользователь, причем обработка этого сообщения происходит асинхронно, т.е. без ожидания завершения почтовой системы, поскольку отправка почты может занимать довольно длительное время.

Контрольные вопросы

1. Для отображения обмена сообщениями между объектами используются следующие виды диаграммы Sequence:

а) Sequence: Instance; б) Sequence: Role; в) Sequence: Object.

2. Сообщение Message от одного объекта к другому может обозначать:

а) вызов операций класса;

б) передачу управления вызываемому объекту;

в) передачу управления самому себе;

г) все ответы правильные.

3. Для отображения времени жизни объекта необходимо использовать следующие элементы:

а) Lifeline Actor; б) Lifeline; в) оба варианта правильные.

2.5. Лабораторная работа №5

ПОСТРОЕНИЕ ДИАГРАММ COMPONENT И CLASS

Цель работы:

- научиться строить диаграммы Component и Class в среде автоматизированного синтеза Rational XDE;
- разработать диаграммы Component и Class для проектируемой прикладной системы.

Задание:

1. Средствами диаграммы Component показать организацию и связи между программными компонентами системы.
2. Средствами диаграммы Class разработать внутреннюю структуру системы, описать наследование и взаимное положение классов, используя соответствующие типы связей между ними.

Порядок выполнения работы

1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.

Библиотека БГУИР

3. Построить диаграмму Component по предложенной тематике.
4. Построить диаграмму Class по предложенной тематике.

Создание модели реализации средствами диаграммы Component

Количество инструментов диаграммы компонентов в Rational XDE по сравнению с Rational Rose сокращено. Здесь используется только два основных значка для обозначения компонентов: Component и Component Realization. Поскольку .NET – полностью объектно-ориентированная среда разработки, при создании программ C# не используется ни разделение файлов, ни отдельное обозначение подпрограмм, как это было в случае построения диаграмм компонентов в Rational Rose [8].

Диаграмма компонентов предназначена для отображения зависимостей между компонентами системы и находящимися в них классами. Для построения диаграммы нужно выбрать из контекстного меню модели *Add Diagram=>Component.* На рис. 2.11 приведен Toolbox диаграммы Component.

Рассмотрим основные элементы диаграммы компонентов.

Значок *Component* позволяет создать на диаграмме отображение компонентов. Компонент – это элемент реализации с четко определенным интерфейсом. Компонентами могут быть любые библиотечные или программные файлы, содержащие реализации классов системы.

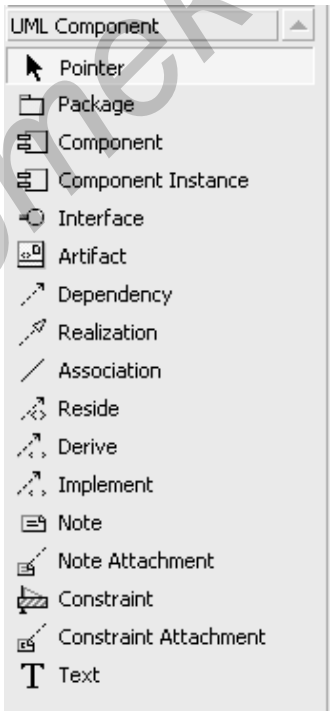

Рис. 2.11. Toolbox диаграммы Component

Значок *Dependency* позволяет создать на диаграмме отображение использования одного компонента другим или их зависимость друг от друга.

Библиотека БГУИР

Инструмент *Interface* отображает на диаграмме интерфейс. Это список операций, посредством которых компоненты взаимодействуют между собой.

Элемент *Realization* показывает на диаграмме реализацию интерфейса компонентом.

Значок *Association* отражает на диаграмме ассоциации элементов.

Инструмент *Reside* создает на диаграмме отображение принадлежности класса к компоненту.

Значок *Component Instance* предназначен для создания на диаграмме отображения экземпляра компонента.

Проектирование классов приложения с помощь диаграммы Class

Изучив все диаграммы, предназначенные для построения модели системы, можно переходить к разработке диаграммы классов, которая считается основной в создании приложения ASP.NET. Диаграмма Class выполняет целый ряд функций: используется для создания иерархии классов; на ее основе строятся модели данных и проектируется структура Web-приложений; на этапе анализа и проектирования используется для создания диаграмм реализации прецедентов; с ее помощью создается модель предметной области, которая используется на этапе анализа. Кроме того, широко применяются стереотипы классов, позволяющие адаптировать стандартную UML-диаграмму для конкретных целей, расширяя ее возможности. Рассмотрим основные этапы разработки системы с помощью диаграммы классов.

Для создания модели используются три стереотипа классов, которые определяют их назначение: граничный класс (boundary), сущность (entity), управление (control).

Стереотип *граничный класс* показывает, что класс предназначен для взаимодействия с внешними актерами и стоит на границе системы, поэтому и называется граничным. Такой класс, получая сообщение от внешнего актера, транслирует их внутрь системы, генерируя и передавая соответствующие сообщения другим классам.

Классы *сущности* используются для моделирования классов, которые отвечают за хранение определенной информации. Эти классы реализуют возможности по получению, изменению и сохранению информации в базе данных. Классы *сущности* обычно не отражаются ни на одной диаграмме прецедентов, но требуются для выполнения внутреннего хранения данных.

Классы *управления* используются для координации работы других классов приложения. Поведение этих классов обычно реализует один или несколько прецедентов, показанных на диаграммах моделирования. Классы *управления* реализуют поведение системы при помощи потоков управления. Они являются промежуточными звеньями между *граничными* классами и классами *сущностями*.

На этапе проектирования диаграмма классов используется для проектирования подсистем и иерархии классов. Одна или несколько диаграмм классов

Библиотека БГУИР

описывают классы верхнего уровня. При включении на диаграмму пакетов в модель добавляются диаграммы классов, описывающие содержимое пакетов. Пакеты обычно используют для группирования классов по подсистемам. Кроме классов, в подсистемы могут включаться реализации вариантов использования, интерфейсы и другие подсистемы. Разделение на подсистемы значительно упрощает параллельную разработку, конфигурирование и инсталляцию конечного продукта. Создание подсистем позволяет проще устанавливать различные категории доступа к информации для пользователей, а также отделить алгоритмы для организации связи с внешними продуктами.

Для разрабатываемого виртуального книжного магазина можно выделить следующие подсистемы: *Управление регистрацией, Управление каталогами, Управление заказами, Управление сервисными функциями.*

При разработке приложения .NET интерфейс обозначает полноценные объекты, которые проектируются наравне с классами системы. В .NET интерфейс – это элемент, включаемый в классы, которые должны его реализовывать. Такой подход позволяет разрабатывать интерфейсы отдельно, а затем включать их в нужные классы. На структуру интерфейса накладываются ограничения. В его состав входят только абстрактные члены, в нем могут быть определены события, методы, свойства, но он не может содержать конструкторов, деструкторов и констант.

Следующий этап работы с диаграммой *Class* – это описание логического представления системы. Обычно диаграмма классов создается для всех классов системы в отличие от других диаграмм, которые строятся для отдельных объектов со сложным поведением и взаимодействием. Классы на диаграмме могут представлять любые C#-классы: простые, параметризованные или метаклассы.

Рассмотрим работу с диаграммой Class при проектировании подсистем виртуального магазина. На рис. 2.12 приведен Toolbox диаграммы классов, где доступны все необходимые элементы.

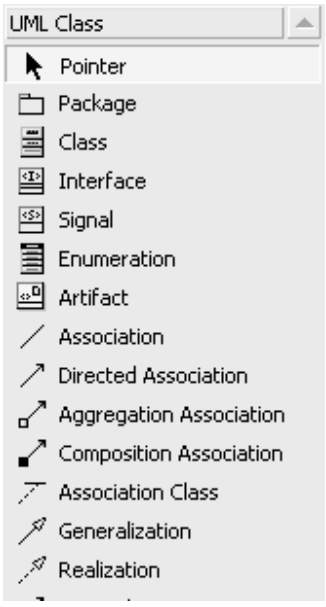

Рис. 2.12. Toolbox диаграммы Class

Библиотека БГУИР

Рассмотрим назначение и возможности инструментов диаграммы *Class*.

Для создания классов используется значок *Class*. Важным достоинством работы в Rational XDE является возможность одновременного просмотра диаграммы в графическом виде и получаемого по этой диаграмме исходного кода (рис. 2.13). Для этого необходимо воспользоваться пунктом меню *Window=>New Horizontal Tab.*

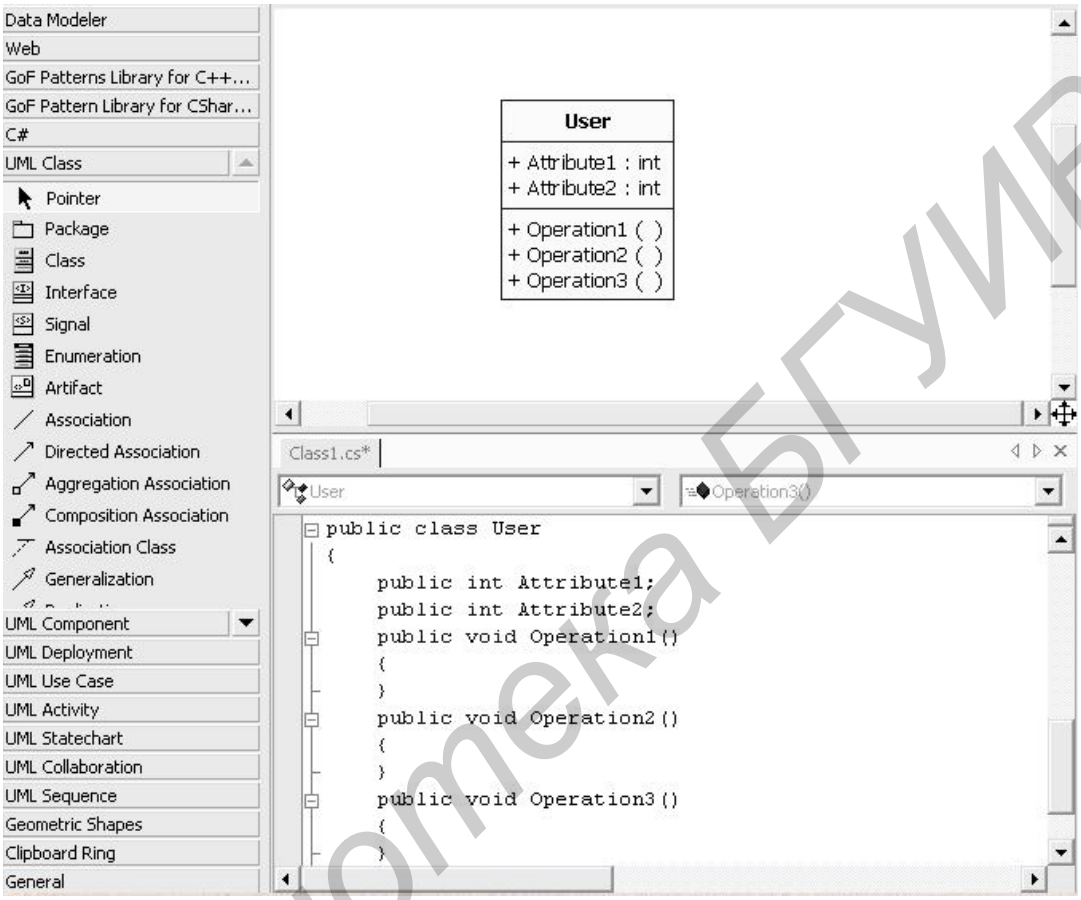

Рис. 2.13. Работа с классом в среде Rational XDE

Добавлять атрибуты и операции в класс можно двумя способами: во-первых, посредством контекстного меню *Add UML => Attribute (Operation)*, а во-вторых, с помощью пункта контекстного меню *Add C-Sharp*. Вторым способом можно добавлять все элементы, которые необходимы для создания приложения, а не только свойства и операции. Добавим классу *Account* свойство *Name*, для чего выберем *Add C-Sharp=>Property*. Соответствующее этому окно показано на рис. 2.14. Видно, что в окне можно ввести ряд данных, относящихся к свойству класса. Автоматически Rational XDE добавляет операции доступа к свойству, создавая необходимый для этого исходный код. Поле *Associated Field* позволяет создать скрытый атрибут класса, к которому осуществляется доступ посредством операций *Set* и *Get*. Также в этом окне можно настроить параметры для создания свойства.

В C# каждый класс может содержать в себе определения других элементов, а именно: классов, интерфейсов, структур и перечислений. При помощи

Библиотека БГУИР

пункта контекстного меню *Add C-Sharp* возможно добавление этих элементов и делегатов, не выходя из диаграммы. Делегаты в C# представляют собой контейнеры, где хранятся описания метода класса. Они позволяют вызывать статические методы класса под именем делегата, скрывая от клиента информацию о том, что он пользуется методом класса.

Окно *Collection*, которое активизируется при вызове пункта контекстного меню *Collection Editor*, предоставляет доступ к настройке элементов класса, таких, как атрибуты, операции, связи и другие элементы, которые невозможно отредактировать через окно свойств, поскольку они имеют списочную структуру. Просматривать свойства списком удобно, если на диаграмме элемент не показан или его свойства скрыты.

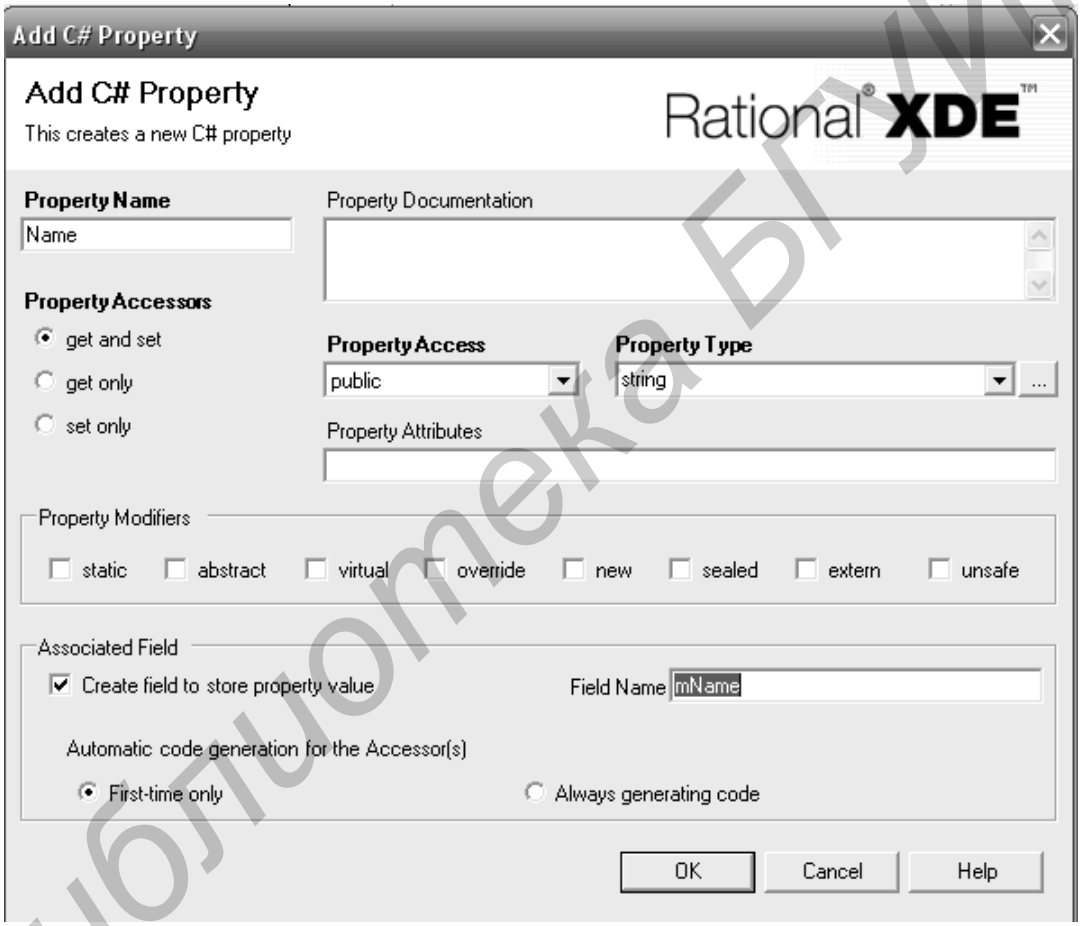

Рис. 2.14. Окно Rational XDE C# Property

Перейдем к рассмотрению остальных инструментов диаграммы классов. Значок *Interface* позволяет создать элемент интерфейса. Он представляет собой абстрактный контейнер абстрактных операций, которые должны быть реализованы в классе. Значок *Signal* позволяет создать элемент сигнала, который отражает асинхронное сообщение от одного класса к другому, т.е. без ожидания ответа. Значок *Enumeration* создает элемент перечисление. Это тип данных, состоящий из набора констант базового типа. Перечисления используются для создания набора именованных страниц.

Библиотека БГУИР

Классы не могут существовать обособленно. Основная их задача – взаимодействовать друг с другом при помощи обмена сообщениями. При построении модели классы связываются друг с другом связями различных видов. Согласно спецификации UML таких связей довольно много, однако в отличие от C++ в C# многие виды связей дают одинаковые результаты при генерации исходного кода.

Значок *Association* показывает ассоциации классов. Поскольку направление не указано, Rational XDE не может определить, какой класс первичен в установленной ассоциации, поэтому считает оба класса равноправными. Этот вид связи не используется для проектирования классов, по которым планируется получить исходный код, т.к. он никак на него не влияет.

Значок *Directed Association* позволяет создать направленную ассоциацию между классами. Этот тип связи показывает, что объект одного класса включается в другой класс для получения доступа к его методами. Rational XDE определяет название переменной для объекта класса, и при генерации исходного кода эта переменная будет включена в него перед определением методов и атрибутов класса.

Значок *Aggregation association* позволяет отразить на диаграмме агрегацию элементов. Этот тип связей показывает, что один элемент входит в другой как часть. Агрегация используется при моделировании как дополнительное средство, показывающее, что элемент состоит из отдельных частей.

Значок *Composition* позволяет отразить состав объекта и элементы, включенные в композицию.

Значок *Association Class* используется для отображения класса, ассоциированного с двумя другими. Фактически данный тип связи отражает то, что некоторый класс со своими атрибутами включается как элемент в два других, при этом никак не отражаясь на создаваемом исходном коде.

Значок *Realization* отражает на диаграмме отношение между классом и интерфейсом, который этим классом реализуется.

Значок *Dependency* показывает на диаграмме такое отношение зависимости, когда один класс использует объекты другого. Для классов C# этот тип связи не имеет широкого применения и на создаваемый код не влияет.

Значок *Generalization* указывает, что один класс является родительским по отношению к связанному, при этом будет создан код наследования класса.

Значок *Bind* создает особый тип зависимости между классами и используется для работы с шаблонами классов. Эта связь показывает замещение параметров, определенных в шаблоне.

Значок *Usage* отражает тип зависимости, показывающей, что один из элементов модели требует наличия другого, связанного с ним такой связью.

Значок *Friend Permission* строит тип зависимости, показывающий, что один класс предоставляет доступ к своему содержимому другому классу.

Значок *Abstraction* определяет, что элементы модели представляют одно понятие, но на разных уровнях абстракции.

Библиотека БГУИР

Значок *Instantiate* позволяет показать, что элемент модели отражает особый случай другого элемента.

Контрольные вопросы

1. В чем заключается принципиальное различие между диаграммами Deployment и Component?

2. На каком этапе разработки модели системы рекомендуется строить диаграмму Component?

3. Область видимости операции класса в окне диаграммы может отображаться:

а) графическим значком слева от операции;

б) текстовым значком справа от операции;

в) оба ответа правильные.

4. Какой тип связи необходимо использовать, чтобы включить в модель отражение того, что класс Class1 включает в себя Class2:

а) использовать связь Aggregation;

б) использовать связь Composition;

в) оба ответа правильны.

5. Для отражения отношения между классом и интерфейсом необходимо использовать следующей тип связи:

а) Realization; б) Generalization; в) Association.

2.6. Лабораторная работа №6

КОДОГЕНЕРАЦИЯ В СРЕДЕ RATIONAL XDE

Цель работы:

- изучить приемы и возможности кодогенерации в Rational XDE;
- выполнить кодогенерацию на основе диаграммы классов.

Задание:

1. Создать исходный код приложения на основе диаграммы классов.

2. Добавить функциональность в построенные классы и получить работающее программное приложение.

Порядок выполнения работы

1. Изучить теоретическую часть лабораторной работы.

2. Ответить на контрольные вопросы.

3. Выполнить кодогенерацию, а затем добавить функции и получить работающее приложение.

Библиотека БГУИР

Генерация исходного кода в среде Rational XDE

После того как рассмотрены все диаграммы, предназначенные для проектирования системы, можно переходить к генерации исходного кода на основе диаграммы классов. Генерация по готовым диаграммам позволяет исключить рутинный труд по ручному кодированию и созданию шаблонов, что сокращает количество ошибок на стадии преобразования готовой модели в программный код. Кроме того, по любой программе, написанной при помощи среды Microsoft .NET, можно провести обратное проектирование и разобраться в архитектуре, представленной графически в виде иерархии классов.

Rational XDE предоставляет большое количество настроек для автоматической и ручной генерации кода приложения. Все установки по-разному влияют на генерацию кода и призваны повышать продуктивность работы.

Для работы с исходным кодом в контекстном меню класса доступна следующая группа пунктов меню:

– *Generate Code* – генерация кода по созданной модели;

– *Synchronize* – синхронизация кода и модели. При этом все изменения, внесенные в код, отражаются в модели, а изменения модели переносятся в исходный код;

– *Browse Code* – позволяет переключаться в режим просмотра созданного кода.

Для работы с одним-двумя классами этого вполне достаточно, но при работе с большими моделями удобнее использовать синхронизацию в автоматическом режиме.

При создании архитектуры классов приложения используются различные типы связей. Одни не требуют настройки, другие обладают значительным количеством свойств, влияющих на исходный код.

При создании исходного кода ассоциативные связи создают одинаковый код независимо от того, направленная это ассоциация или нет, отражает ли связь агрегацию или композицию классов (рис. 2.15). При генерации кода будет создана переменная класса, заданная в параметрах связи.

При этом направление связи не играет роли, а важно задание имени переменной. При создании связи *Directed Association* Rational XDE автоматически создает переменную класса того типа, который находится в окончании стрелки связи. Эта переменная включается в класс, из которого стрелка исходит. Установки, созданные при построении связи, несложно изменить с помощью окна *Properties*.

Для настройки параметров автоматической синхронизации модели необходимо выбрать из главного меню пункт *Tools => Options => Rational XDE => => Round Trip Engineering => Synchronization Setting* (рис. 2.16).

После отметки в приведенном окне пункта *Automatic Synchronization* становятся доступными следующие варианты синхронизации:

– *When Saving Model Files* – синхронизация в момент сохранения модели;

– *When Model gets Focus* – синхронизация в момент активизации модели;

Библиотека БГУИР

– *When saving Code Files* – синхронизация происходит в момент сохранения кода после его изменения;

– *When Code gets Focus* – синхронизация происходит в момент активизации окна кода.

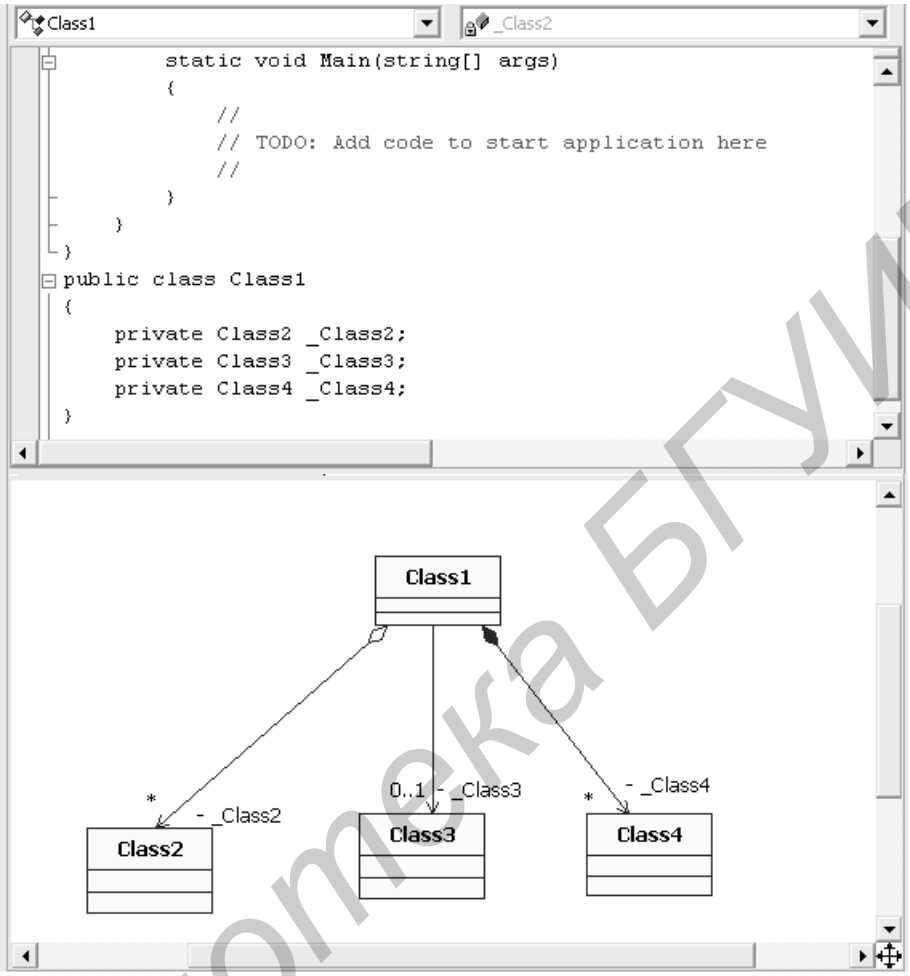

Рис. 2.15. Исходный код для различных типов связей

Каждый вариант синхронизации удобен для своего случая. На этапе анализа и проектирования, когда основную работу по моделированию выполняет аналитик, синхронизация в момент сохранения модели позволит программистам, работающим в проекте, пользоваться самой последней версией структуры приложения. На этапе реализации, когда основная работа ложится на программистов, установка синхронизации после сохранения внесения изменений в файлы кода позволит поддерживать модель системы в актуальном состоянии. Когда же разработка закончена или создается новая версия уже работающей системы, чтобы не нарушить работу программы, синхронизация может быть вообще отключена.

Библиотека БГУИР

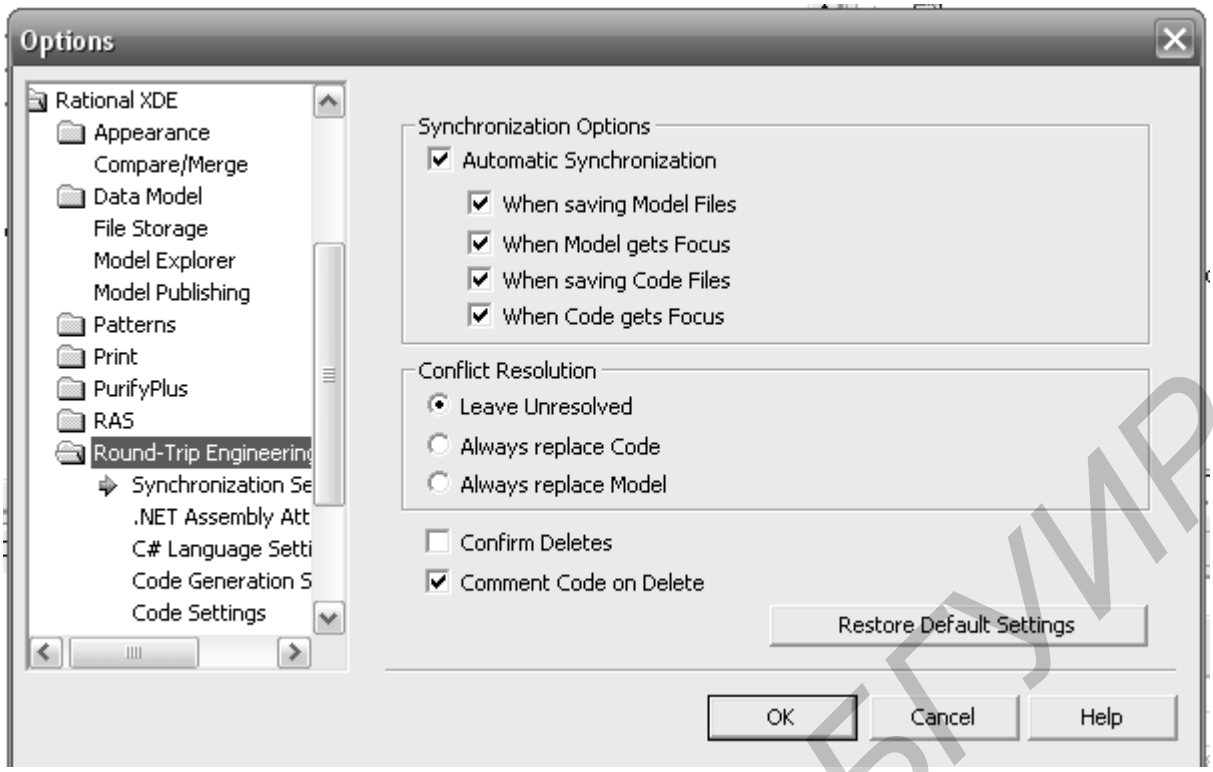

Рис. 2.16. Настройка синхронизации кода

Контрольные вопросы

1. Создание кода класса выполняется при помощи:

а) пункта контекстного меню класса Build;

б) пункта контекстного меню класса Generate Code;

в) нет правильных ответов.

2. Для задания двусторонней синхронизации необходимо установить следующее значение свойства Synchronization Police:

а) Both Direction; б) Synchronize; в) Auto Update; г)нет правильных ответов.

3. Для изменения модификатора языка для кода класса необходимо воспользоваться:

а) окном Properties класса; б) окном Code Properties; в) все ответы правильные.

2.7. Лабораторная работа №7

МОДЕЛИРОВАНИЕ ДАННЫХ В RATIONAL XDE

Цель работы:

- изучить возможности моделирования данных в Rational XDE;
- построить физическую модель данных для разрабатываемого программного приложения.

Библиотека БГУИР

Задание:

создать файл модели, в которой разместить базу данных и наполнить ее связанными между собой элементами модели данных.

Порядок выполнения работы

1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.
3. Разработать модель данных.

Моделирование данных при помощи Date Modeler

При разработке программных систем процесс создания модели данных является одним из важнейших этапов. Для его реализации в Rational XDE включен специальный модуль для моделирования данных – *Data Modeler*. Он позволяет моделировать физическую структуру данных. В Rational XDE включены дополнительные стереотипы классов, не имеющие отражения в стандартном языке UML, не позволяющие на основании стандартных элементов создавать модели структур данных.

Data Modeler предназначен для работы со всеми необходимыми объектами базы данных: таблицами, триггерами, хранимыми процедурами и представлениями данных. Модуль может моделировать данные в стандарте ANSI SQL, а также поддерживает форматы для основных систем управления базами данных: Oracle, Microsoft SQL Server, Sybase.

Для того чтобы воспользоваться возможностями моделирования данных и выполнением обратного проектирования, необходимо включить в модели поддержку профиля *Data Modeler*. В случае если модель уже создана, в ее свойства необходимо добавить поддержку работы с базами данных. Для этого необходимо выполнить следующие действия:

1. Перейти в окно *Model Explorer*.
2. Перейти в созданную модель, например виртуальный магазин.
3. Из контекстного меню модели выбрать пункт *Properties Window*.
4. В окне *Properties* установить значение свойства Applied Profiles = Date Modeler.

После этого в контекстном меню модели появятся новые пункты меню, относящиеся к проектированию данных и обратному проектированию (рис. 2. 17).

Для запуска режима Reverse Engineering необходимо выбрать соответствующий пункт в контекстном меню модели, после чего активизируется мастер обратного проектирования. Он позволяет по шагам пройти весь процесс обратного проектирования, предоставляя инструкции по дальнейшим действиям. Поскольку для различных систем управления БД синтаксис процедур и таблиц данных различен, то мастер позволяет указать конкретную БД для проектирования, начав с указания источника, в котором будет выполняться поиск БД. Та-

Библиотека БГУИР

ких источников может быть два: непосредственно БД и описание на языке определения данных DDL. В последнем случае возможно указание и SQL-скрипта, по которому будет создана структура.

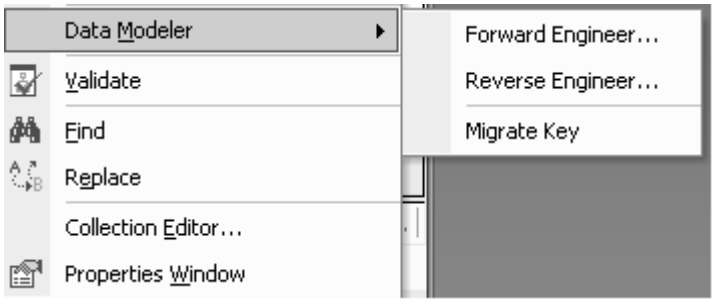

Рис. 2.17. Пункт Data Modeler в контекстном меню модели

Следующий шаг – это определение типа БД и параметров соединения. Затем следует указать необходимую схему БД. Позже схема преобразуется в пакет, который содержит все полученные из БД элементы. Далее необходимо указать, какие элементы нужно получить из БД. После окончания работы будет создан пакет *dbo*, в котором присутствуют все полученные из БД элементы. Однако пока модель еще трудна для обозрения, так как все элементы представлены в окне Model Explorer и связи между ними не очевидны. Для того чтобы получить наглядную диаграмму, необходимо перенести все элементы в область рисования диаграммы и расставить связи между элементами.

Таким образом, для работы с моделями данных необходимо выполнить несколько общих шагов. Сначала создать файл модели, затем в ней создать базу данных, которую наполнить различными элементами: таблицами, триггерами, хранимыми процедурами и представлениями.

На рис. 2.18 показано окно инструментов, используемых для создания модели данных.

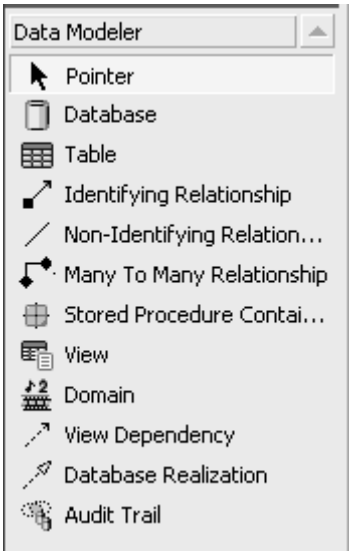

Рис. 2.18. Окно инструментов Data Modeler

Библиотека БГУИР

Значок *Database* позволяет создать на диаграмме отображение базы данных. Он представляется аналогией компонента для класса, поскольку БД является контейнером, объединяющим все другие элементы. С помощью окна спецификаций можно настроить значительное количество дополнительных параметров.

Элемент *Table Space* не имеет значка в окне инструментов, однако он может быть создан посредством пункта *Add Data Modeler => Table Space* контекстного меню *Database. Table Space* – логическое хранилище, в которое помещаются таблицы данных. Возможно создание нескольких таких элементов. В момент создания *Table Space* выполняется автоматическое связывание его с элементом *Database* при помощи связи *Dependency,* а связи с таблицами данных определяются при помощи *Realization*, которую необходимо с каждой таблицей создавать вручную.

Пункт *Table* позволяет создать на диаграмме отображение таблицы данных. Хотя в окне инструментов для таблицы используется свой значок, в самой диаграмме она изображается как обычный класс со стереотипом «Table». В дальнейшем таблицу также можно преобразовать в класс.

Значок *View* позволяет отразить в модели представление данных. Элемент является виртуальной таблицей, которая физически не присутствует в БД, и создается в момент запроса. Доступ к *View* такой же, как и к обычным таблицам. Поскольку создание *View* оптимизируется на уровне СУБД, то использование представлений позволяет увеличить производительность приложений. Представления также могут использоваться для предоставления пользователям доступа к данным без возможности внесения изменений.

Создание модели реляционной БД невозможно без установления связей между таблицами. Rational XDE поддерживает все необходимые типы связей, которые используются при создании структуры данных.

Значок *Identifying Relatioship* позволяет создать на диаграмме отображение такой связи между таблицами. Она отображает связь между родительской и дочерней таблицами, где дочерняя таблица не может существовать без родительской. При создании такого типа связи создается элемент внешнего ключа (*foreign key*), для которого автоматически создается ограничение (*constraint*) (рис. 2.19).

Значок *Non-Identifying Relationship* позволяет создать на диаграмме отображение отношений между таблицами, при котором нет необходимости в создании внешнего ключа, т.е. записи дочерней таблицы могут существовать и без ссылки на родительскую таблицу. Обычно это показывают мощностью связи (cardinality), например 0...1 (рис. 2.20).

Значок *Many To Many Relationship* не создает нового типа связей «многие ко многим», поскольку такая связь не может быть создана без промежуточной таблицы. При выборе этого типа связи система сама создает еще одну таблицу и связывает с ней родительские таблицы идентифицирующими связями.

Значок *Dependency* позволяет указать зависимость одного элемента диаграммы данных от другого, причем делается это системой автоматически.

Библиотека БГУИР

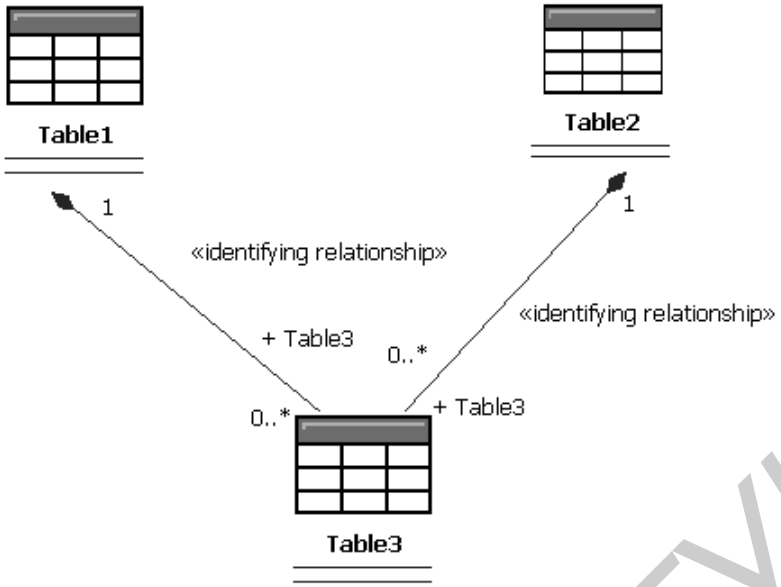

Рис. 2.19. Таблицы с идентифицирующими связями

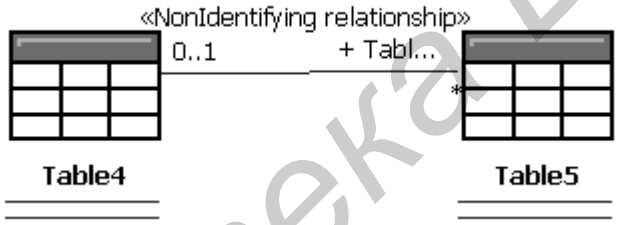

Рис. 2.20. Таблицы с неидентифицирующими связями

Элемент *Realization* позволяет показать реализацию элемента.

Пункт *Stored Procedure* позволяет создать на диаграмме контейнер для хранимой не сервере процедуры. Она представляет собой скрипт, который может быть в любой момент запущен любым клиентом.

Пункт *Domain* позволяет создать шаблон для типов данных пользователя, а также для определения правил обработки колонок в таблицах.

Контрольные вопросы

1. Какие действия необходимо выполнить для создания пустой модели данных:

а) из меню File выбрать New => File => Data Model;

б) из контекстного меню решения выбрать Data Model => Create;

в) все ответы подходят.

2. Для создания скрипта по модели данных необходимо использовать следующий пункт контекстного меню:

а) Reverse Engineering;

б) Generate Script;

Библиотека БГУИР

в) Forward Engineering.

3. Для создания поля Foreign Key необходимо использовать следующий тип связи:

а) Dependency; б) Realization; в) Identifying Relationship.

2.8. Лабораторная работа №8

СОЗДАНИЕ WEB-ПРИЛОЖЕНИЙ В RATIONAL XDE

Цель работы:

- изучить возможности создания Web-моделей в Rational XDE;
- построить Web-приложение в Rational XDE.

Задание:

1. Создать проект на заданную тему, синхронизировав его с Web-моделью.
2. Разработать структуру модели и генерировать код входящих в нее элементов.

Порядок выполнения работы

1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.
3. Разработать Web-модель приложения на заданную тему.

Особенности создания Web-приложений в Rational XDE

В Rational XDE для создания структуры системы, ориентированной на работу в Web, используется диаграмма классов, в которой при создании архитектуры приложения учитываются ограничения реализации Web-приложения. В спецификации UML не предусмотрен отдельный тип диаграмм для выполнения этой задачи. Вполне достаточно диаграммы классов с дополнительными стереотипами для работы с Web-элементами. На рис. 2.21 приведены инструменты для разработки структуры Web-приложения. Набор инструментов существенно расширился по сравнению с программой Rational Rose [4]. Это объясняется тем, что Rational Rose поддерживает разработку только ASP-страниц, а Rational XDE предназначена для ASP.NET. Кроме стандартных Web-инструментов, на панель вынесены значки, которые создают целые группы элементов.

Для построения Web-модели можно использовать новое или существующее приложение, создать в нем новую модель, а затем выполнить синхронизацию проекта с моделью. Для этого в контекстном меню проекта необходимо выбрать пункт *Synchronize*. В случае если модель еще не создана, Rational XDE создает новую модель с таким же названием, как у проекта, и проставляет все необходимые ссылки. После синхронизации Rational XDE открывает новую диаграмму, готовую к работе.

Библиотека БГУИР

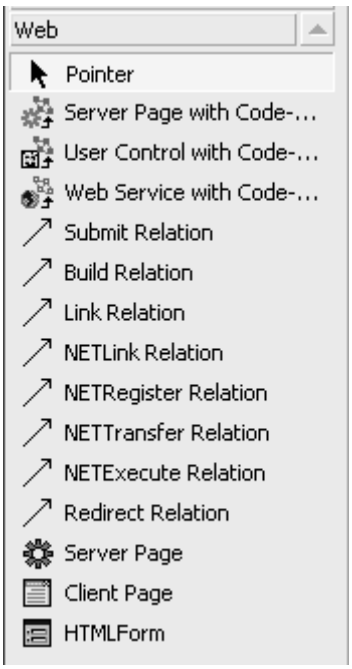

Рис. 2.21. Панель Web-окна Toolbox

Рассмотрим возможности Rational XDE по созданию Web-модели.

Значок *Client Page* позволяет создать на диаграмме отображение простых страниц HTML, не имеющих собственного поведения. Обычно такие страницы предоставляют пользователям определенную, заранее заданную информацию. Страницы *Client Page* так же, как и классы, могут содержать атрибуты и операции, которые добавляются посредством контекстного меню элемента, после чего код обновляется автоматически или вручную.

Элемент *Link Relation* позволяет отобразить связи между страницами в том случае, когда на одной странице есть ссылка на другую. Rational XDE не знает, куда добавлять созданную ссылку, и вставляет ее в конец файла.

Значок *HTML Form* позволяет отобразить формы ввода, которые присутствуют на страницах HTML. Форма не может существовать сама по себе, она включается на страницу при помощи агрегирования. Поэтому для ее разработки начинают с создания страницы, на которой будет находиться форма. Сначала форма соединяется со страницей связью *Link*, а затем посредством пункта *Properties Window* из контекстного меню связи значение свойства *UML=>Kind* изменяется на *Agregation*. Для того чтобы код *Page1* изменился, необходимо в окне Model Explorer отбуксировать элемент *Form1* в элемент *Page1*. После чего в результате генерации кода в код страницы добавятся строки обработки формы. Для добавления полей в форму можно воспользоваться пунктом ее контекстного меню *Add Web*.

На рис. 2.22 показаны клиентские страницы *Page1* и *Page2*, а также *Form1*. Все элементы соединены вышеописанными связями. Под диаграммой приведен автоматически сгенерированный код.

Библиотека БГУИР

В случае необходимости отразить обработку данных, передаваемых из формы клиентской или серверной странице, используется значок связи *Submit Relation* (отношение предоставления).

Создание клиентских страниц вручную происходит достаточно редко: только в случае разработки статичного приложения. Поскольку основная логика приложения должна работать на сервере сети, Web-приложение создает клиентские страницы динамически по запросам пользователей. Для этого используются *Server Page*, которые и реализуют генерацию страниц для пользователя, что отображается при помощи связи *Build Relation*. Таким образом, *Server Page* являются связующим звеном между классами приложения и их визуальным отображением.

На рис. 2.23 приведена серверная страница и сгенерированный по ней исходный код.

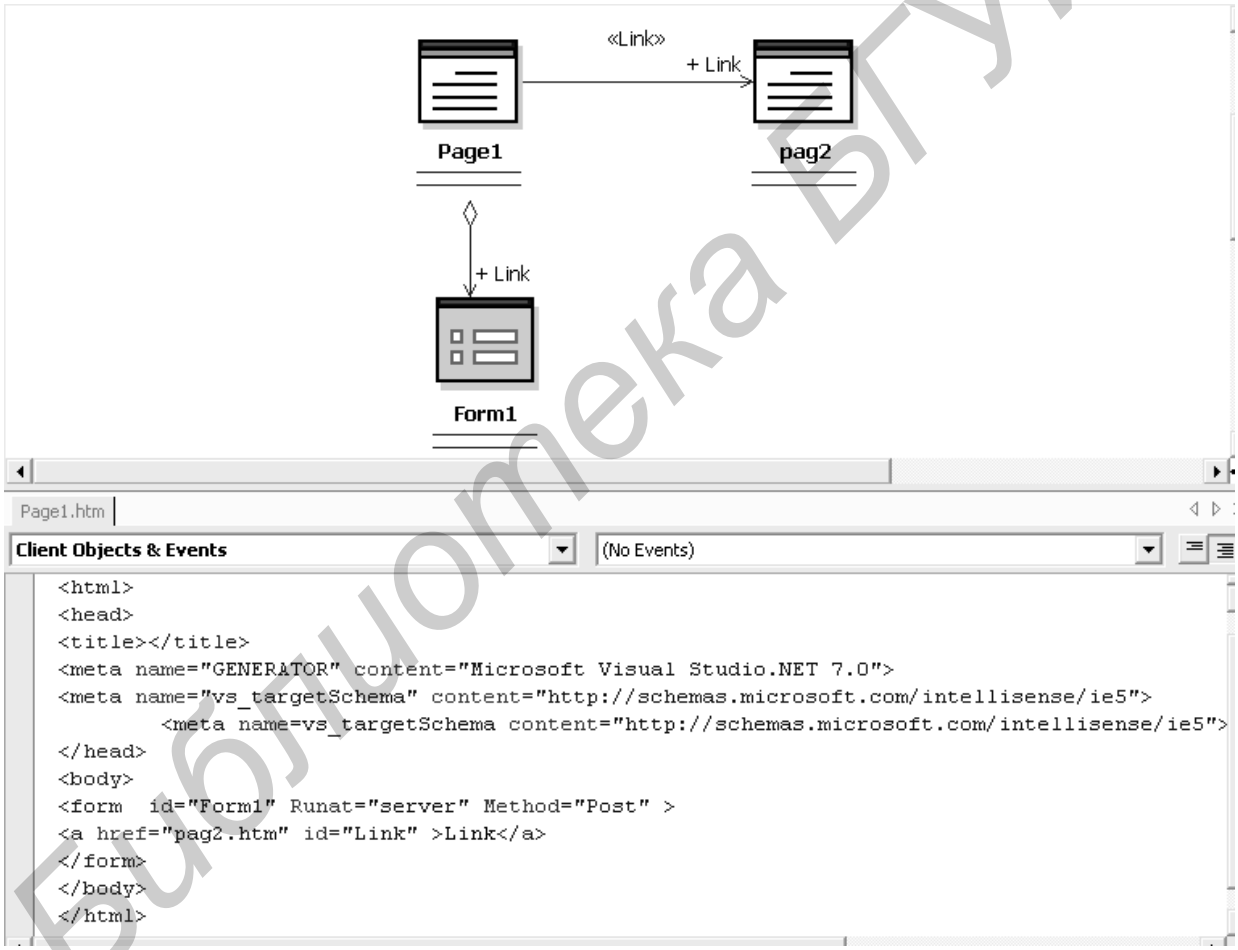

Рис. 2.22. Web-модель и сгенерированный по ней код

При помощи значка *Server Page with Code-Behind* создается набор элементов (рис. 2.24), связанных между собой и содержащих необходимые элементы для создания ASP.NET приложения.

Значок *User Control with Code-Behind* позволяет отражать создание пользовательских элементов управления вместе с классом их обработки. На диаграм-

Библиотека БГУИР

ме создаются два элемента: страница пользовательского элемента управления и класс, от которого выполняется наследование.

Рис. 2.23. Серверная страница и сгенерированный по ней код

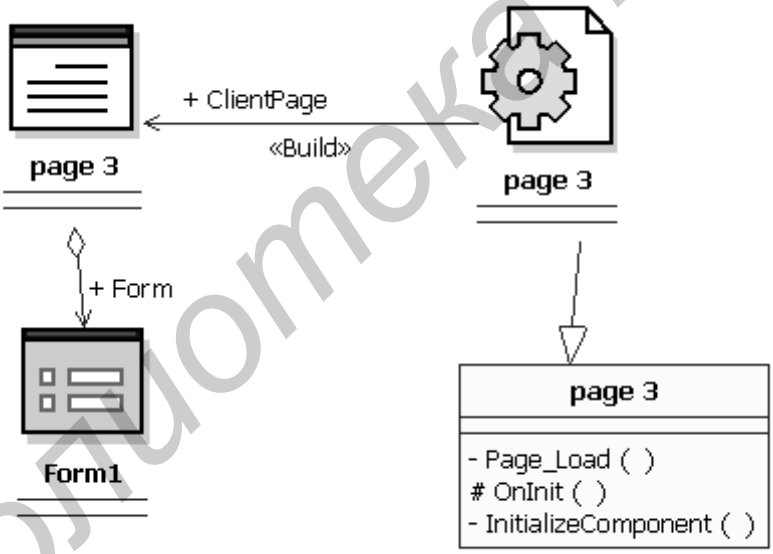

Рис. 2.24. Пример Server Page with Code-Behind

Значок *Web Service with Code-Behind* позволяет отражать создание сервисов, которые предоставляют информацию приложениям вместе с классом их обработки.

Для отражения связей между страницами ASP используется значок *NET-Link Relation*. Если форма, расположенная на ASP странице, использует элементы управления, созданные пользователем, то значок *NETRegister Relation* позволяет отразить связи между страницей ASP и элементом управления пользователя.

Библиотека БГУИР

Для отражения передачи управления другой странице используется значок *NETTransfer Relation*. При генерации кода создается директива *Transfer*, которая позволяет передавать управление другой странице с сохранением доступа к внутренним объектам исходной страницы.

Значок *NETExecute* позволяет отразить передачу управления другой странице, но при генерации кода создается директива *Execute*, позволяющая не только передать управление с сохранением доступа к внутренним объектам, но и по завершении вернуть управление вызывающей странице.

Для отражения простой переадресации с одной страницы на другую используется связь при помощи значка *Redirect Relation*. При этом не сохраняется доступ к внутренним объектам, как это происходит при использовании связей *NETTransfer* и *NETExecute*. Такая переадресация используется при необходимости активизации страницы, чье изображение зависит от установленного языка или возможностей браузера.

Контрольные вопросы

1. В рабочем поле диаграммы изображения Web-элементов по умолчанию представляют собой:

а) значки, определенные стереотипами;

б) значки, аналогичные изображению класса;

в) значки, зависящие от настройки Options для модели.

2. Для указания страницы, в которую передаются данные из формы, необходимо использовать следующие виды связей:

а) Link Relation;

б) Submit Relation;

в) Redirect Relation.

3. При создании страницы Server Page with Code-Behind создается следующее количество элементов на диаграмме:

а) один; б) два; в) четыре.

ЗАКЛЮЧЕНИЕ

В пособии рассмотрены основные условия и составляющие успешного создания программных систем. Эффективность этой работы зависит от правильного сочетания персонала, процесса разработки и инструментальных средств.

В данном случае процесс разработки ПО представлен технологией RUP. В ее основу положены объектно-ориентированные методы анализа и проектирования ПС, CASE-подход и унифицированный язык моделирования UML.

Предложенный лабораторный практикум предназначен для всестороннего изучения объектно-ориентированного CASE-средства Rational XDE. Это расширенная среда разработки, полностью интегрируемая с Microsoft Visual Studio

Библиотека БГУИР

.NET и позволяющая проектировать ПС при помощи UML-моделей. Возможности Rational XDE таковы, что в среде можно строить диаграммы, генерировать по созданным моделям исходный код приложения, строить диаграммы по разработанному ранее исходному коду, создавать вложенные диаграммы, задавать ссылки на внешние документы системы, публиковать разработанные модели в сети Internet и получать по ним подробные отчеты.

При использовании Rational XDE диаграммы создают единую, связанную между собой модель, которая может быть подключена к программе контроля версий для отслеживания вносимых разными разработчиками изменений. При построении моделей Rational XDE контролирует соблюдение UML-нотации и не позволяет ее нарушать.

Однако Rational XDE не создает за программиста весь исходный код. На сегодняшний день это невозможно, поскольку процесс программирования – это, с одной стороны, формальная работа, а с другой, – творческий процесс, пока недоступный вычислительной машине.

ЛИТЕРАТУРА

1. Крачтен, Ф. Введение в Rational Unified Process / Ф. Крачтен. – М. : Изд. Дом «Вильямс», 2002.
2. Поллис, Г. Разработка программных проектов на основе Rational Unified Process (RUP) / Г. Поллис, Л. Огастин. – М. : Бином, 2005.
3. Фаулер, М. UML. Основы. Краткое руководство по стандартному языку объектного моделирования / М. Фаулер. – М. : Символ-Плюс, 2011.
4. Трофимов, С. А. CASE-технологии: практическая работа в Rational Rose / С. А. Трофимов. – М. : Бином-Пресс, 2002.
5. Кватрани, Т. Визуальное моделирование с помощью Rational Rose 2002 и UML / Т. Кватрани. – М. : Изд. Дом «Вильямс», 2003.
6. Боггс, У. Rational XDE / У. Боггс, М. Боггс. – СПб. : Лори, 2007.
7. Трофимов, С. А. Rational XDE для Visual Studio .NET / С. А. Трофимов. – М. : Бином-Пресс, 2004.
8. Бочкарёва, Л. В. Системы автоматизации проектирования программного обеспечения. Работа в среде Rational Rose : учеб.-метод. пособие / Л. В. Бочкарева, М. В. Кирейцев. – Минск : БГУИР, 2006. – 38 с.

Библиотека БГУИР

Св.план 2012, резерв

Учебное издание

Серебряная Лия Валентиновна

ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. СОЗДАНИЕ ПРИЛОЖЕНИЯ В СРЕДЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО CASE-СРЕДСТВА

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Библиотека БГУИР

Редактор *Т. П. Андрейченко*
Корректор *Е. Н. Батурчик*
Компьютерная верстка *Ю. Ч. Клочкевич*

Подписано в печать 29.10.2012. Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 3,14. Уч.-изд. л. 3,0. Тираж 100 экз. Заказ 235.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.
220013, Минск, П. Бровки, 6