Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования

Владимирский государственный университет

Д.В. Александров, И.В. Грачев, Д.Н. Фадин

CASE-ТЕХНОЛОГИИ

Учебное пособие

«В печать»

Авторы:	Д. В. Александров И. В. Грачев Д. Н. Фадин
Зав. кафедрой	А. В. Костров
Редактор	Е. В. Афанасьева
Начальник РО	Е.П. Викулова
Ответственный секретарь Издательства	Е. А. Амирсейидова
Директор Издательства	Ю.К. Жулев

Владимир 2006

Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования

Владимирский государственный университет

Д. В. Александров, И. В. Грачев, Д. Н. Фадин

CASE-ТЕХНОЛОГИИ

Учебное пособие

Владимир 2006

Д.В. Александров, И.В. Грачев, Д.Н. Фадин

CASE-ТЕХНОЛОГИИ

Учебное пособие

Владимир 2006

УДК 004.41 (075.8)
ББК 32.988-5 я73
А46

Рецензенты:
Доктор технических наук, профессор
Владимирского государственного педагогического университета
Н.Г. Наянзин

Доктор технических наук, профессор,
зав. кафедрой информационных систем и информационного менеджмента
Владимирского государственного университета
А.В. Костров

Печатается по решению редакционно-издательского совета
Владимирского государственного университета

А46 **Александров, Д. В.** CASE-технологии : учеб. пособие / Д. В. Александров, И. В. Грачев, Д. Н. Фадин ; Владим. гос. ун-т. – Владимир : Изд-во Владим. гос. ун-та, 2006. – 64 с. – ISBN 5-89368-688-8

Рассматриваются особенности моделирования бизнес-процессов и программного обеспечения информационных систем на основе технологии *RUP* (*Rational Unified Process*) с использованием CASE-средств *Rational Rose* (*Rational Software Corporation*), *Enterprise Architect* (*Sparx Systems*) и *ArcStyler Tool Suite* (*iO-Software*). В частности, уделяется внимание изучению особенностей стандарта UML 2 при построении комплексных моделей информационных систем и бизнес-процессов.

Предназначено для студентов специальности 230201 – информационные системы и технологии при выполнении курсовой работы и лабораторных занятий по курсу «CASE-технологии», магистрантов направления 230200 – информационные системы при выполнении курсовой работы по дисциплине «Распределенные информационные системы» и практических занятий по дисциплинам «Администрирование в распределенных автоматизированных системах» и «Информационный менеджмент», а также для аспирантов специальности 051301 – системный анализ, управление и обработка информации (промышленность) при выполнении практических и лабораторных занятий по дисциплинам «Информационные технологии в науке и образовании» и «Распределенные информационные системы». Рекомендуется преподавателям при проведении занятий со студентами, магистрантами и аспирантами.

Табл. 3. Ил. 24. Библиогр.: 11 назв.

УДК 004.41 (075.8)
ББК 32.988-5 я73

ISBN 5-89368-688-8

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

CASE-технология (*Computer Aided Software Engineering*) представляет собой методологию проектирования информационных систем (ИС), а также набор инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать эту модель на всех этапах разработки и сопровождения ИС и разрабатывать приложения в соответствии с информационными потребностями пользователей.

Под термином CASE-средства понимаются программные средства, поддерживающие процессы создания и сопровождения ИС, включая анализ и формулировку требований, проектирование прикладного программного обеспечения и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы. CASE-средства вместе с системным программным обеспечением и техническими средствами образуют полную среду разработки ИС [1].

Большинство существующих CASE-средств основано на методологиях структурного или объектно-ориентированного анализа и проектирования, использующих спецификации в виде диаграмм или текстов для описания внешних требований,

связей между моделями системы, динамики поведения системы и архитектуры программных средств.

В настоящем учебном пособии рассматриваются технология объектно-ориентированной разработки программного обеспечения *Rational Unified Process* и унифицированный язык моделирования UML 2. Отдельная глава посвящена обзору концепции модельно-ориентированной архитектуры – современного подхода к разработке и поддержке информационных систем, функционирующих в гетерогенных средах.

Приложение содержит методические материалы для проведения практических занятий по соответствующим дисциплинам. Предлагаемые занятия полностью охватывают теоретическое содержание данного пособия и позволяют учащимся получить навыки работы с современными CASE-средствами.

Глава 1. ОСНОВЫ ЯЗЫКА UML

1.1. Назначение и структура языка UML

Язык UML представляет собой общецелевой язык визуального моделирования, разработанный для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем. Этот язык одновременно является простым и мощным средством моделирования и может быть эффективно использован для построения концептуальных, логических и графических моделей сложных систем самого различного целевого назначения.

Язык UML основан на некотором числе базовых понятий, которые могут быть изучены и применены большинством программистов и разработчиков, знакомых с методами объектно-ориентированного анализа и проектирования. При этом базовые понятия могут комбинироваться и расширяться таким образом, что специалисты объектного моделирования получают возможность самостоятельно разрабатывать модели больших и сложных систем в самых различных областях приложений.

Конструктивное использование языка UML основывается на понимании общих принципов моделирования сложных систем и особенностей процесса объектно-ориентированного анализа и проектирования в частности. Выбор выразительных средств для построения моделей сложных систем предопределяет те задачи, которые могут быть решены с использованием данных моделей. При этом один из основных принципов построения моделей сложных систем – принцип абстрагирования, предписывающий включать в модель только те аспекты проектируемой системы, которые имеют непосредственное отношение к выполнению системой своих функций или своего целевого предназначения. Все второстепенные детали опускаются, чтобы чрезмерно не усложнять процесс анализа и исследования полученной модели.

Другим принципом построения моделей сложных систем является принцип многомодельности. Он представляет собой утверждение о том, что никакая единственная модель не может с достаточной степенью адекватности описывать различные аспекты сложной системы. Применительно к методологии объектно-ориентированного анализа и проектирования (ООАП) это означает, что достаточно полная модель сложной системы допускает некоторое число взаимосвязанных представлений (views), каждое из которых адекватно отражает некоторый аспект поведения или структуры системы. Наиболее общими представлениями сложной системы принято считать статическое и динамическое представления, которые в свою очередь могут подразделяться на другие более частные представления. Феномен сложной системы как раз и состоит в том, что никакое ее единственное представление не является достаточным для адекватного выражения всех особенностей моделируемой системы.

Еще один принцип прикладного системного анализа – принцип иерархического построения моделей сложных систем. Он предписывает рассматривать процесс построения модели на разных уровнях абстрагирования или детализации в рамках фиксированных представлений. При этом исходная модель сложной системы имеет наиболее общее представление (метапредставление), строится на начальном этапе проектирования и может не содержать многих деталей и аспектов моделируемой системы. Тогда процесс ООАП можно представить как поуровневый спуск от наиболее общих моделей и представлений концептуального уровня к более частным и детальным представлениям логического и физического уровней. При этом на каждом из этапов ООАП данные модели последовательно дополняются все большим количеством деталей, что позволяет им более адекватно отражать различные аспекты конкретной реализации сложной системы.

С самой общей точки зрения описание языка UML состоит из двух взаимодействующих частей, таких как:

- *Семантика языка UML*. Представляет собой некоторую метамодель, которая определяет абстрактный синтаксис и семантику понятий объектного моделирования на языке UML.
- *Нотация языка UML*. Представляет собой графическую нотацию для визуального представления семантики языка UML.

Абстрактный синтаксис и семантика языка UML описываются с использованием некоторого подмножества нотации UML. В дополнение к этому нотация UML описывает соответствие или отображение графической нотации в базовые понятия семантики. Таким образом, с функциональной точки зрения эти две части дополняют друг друга.

Семантика определяется для двух видов объектных моделей: структурных и поведения. *Структурные модели*, известные также как статические, 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.

Таким образом, язык UML предназначен, прежде всего, для решения следующих основных задач:

- Предоставить в распоряжение пользователей легко воспринимаемый и выразительный язык визуального моделирования, специально предназначенный для разработки и документирования моделей сложных систем самого различного целевого назначения.
- Снабдить исходные понятия языка UML возможностью расширения и специализации для более точного представления моделей систем в конкретной предметной области.
- Поддерживать такую спецификацию моделей, которая не зависит от конкретных языков программирования и инструментальных средств проектирования программных систем.
- Интегрировать в себя новейшие и наилучшие достижения практики ООАП.

Для решения столь широкого диапазона задач разработана достаточно полная семантика для всех компонентов графической нотации, которая включает в себя описание отдельных семантических элементов, применяемых при построении специальных графических конструкций – диаграмм.

В терминах языка UML 2 все представления о модели сложной системы фиксируются на тринадцати видах диаграмм (рис. 1).

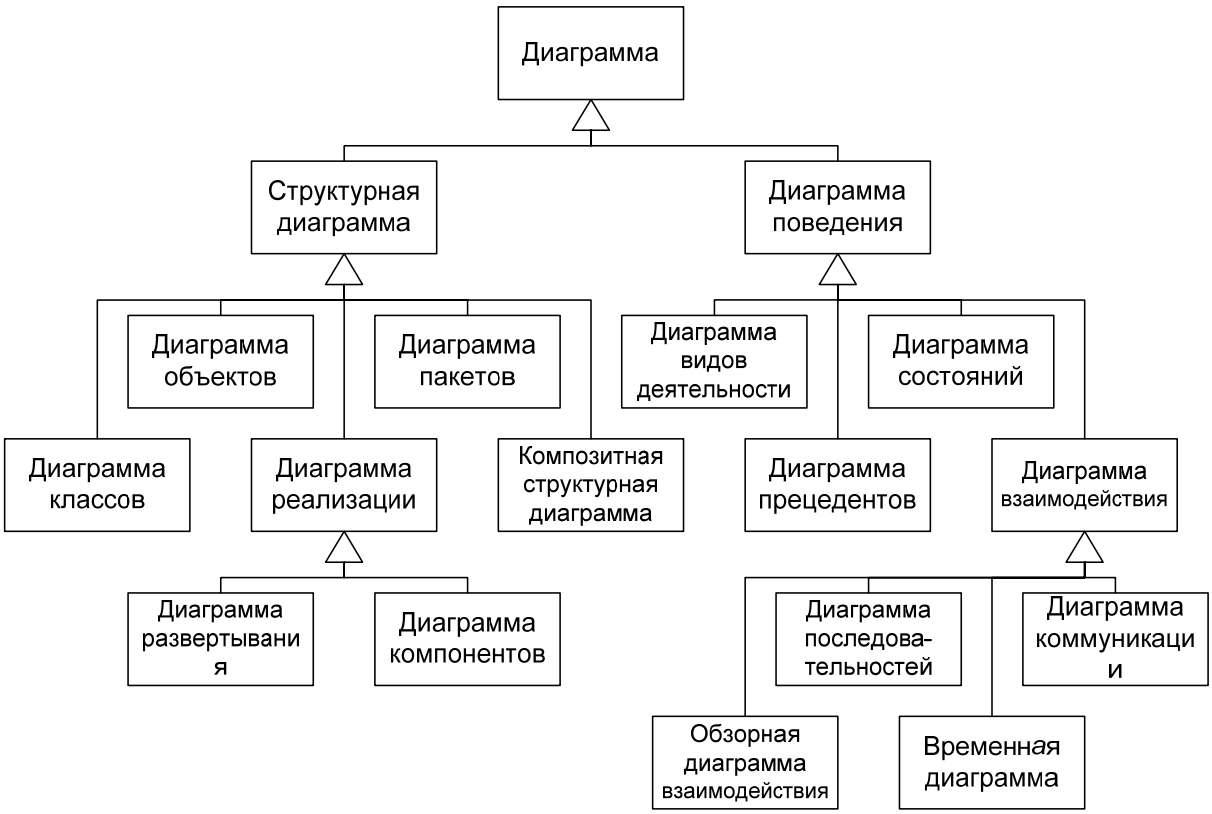

Рис. 1. Иерархия диаграмм языка UML

Перечень этих диаграмм и их названия являются каноническими в том смысле, что представляют собой неотъемлемую часть графической нотации языка UML. Более того, процесс ООАП неразрывно связан с процессом их построения. При этом совокупность построенных таким образом диаграмм самодостаточна в том смысле, что в них содержится вся информация, необходимая для реализации проекта сложной системы.

Каждая из диаграмм детализирует и конкретизирует различные представления о модели сложной системы в терминах языка UML. Диаграмма вариантов использования – наиболее общая концептуальная модель сложной системы, исходная для построения всех остальных диаграмм. Диаграмма классов, по своей сути – логическая модель, отражающая статические аспекты структурного построения сложной системы.

1.2. Особенности изображения диаграмм языка UML

Большинство перечисленных выше диаграмм являются в своей основе графами специального вида, состоящими из вершин в форме геометрических фигур, связанных между собой ребрами или дугами. Поскольку информация, которую содержит в себе граф, имеет в основном топологический характер, ни геометрические размеры, ни расположение элементов диаграмм (за некоторыми исключениями, такими как диаграмма последовательностей с осью времени) не имеют принципиального значения.

В языке UML используется четыре основных вида графических конструкций:

1. *Значки, или пиктограммы*. Значок представляет собой графическую фигуру фиксированных размера и формы. Она не может увеличивать свои размеры, чтобы разместить внутри себя дополнительные символы. Значки могут размещаться как внутри других графических конструкций, так и вне их. Примерами значков могут служить окончания связей элементов диаграмм или некоторые другие дополнительные обозначения (украшения).

2. *Графические символы на плоскости*. Такие двумерные символы изображаются с помощью некоторых геометрических фигур и могут иметь различные высоту и ширину с целью размещения внутри этих фигур других конструкций языка UML. Наиболее часто внутри таких символов помещаются строки текста, которые уточняют семантику или фиксируют отдельные свойства соответствующих элементов языка UML. Информация, содержащаяся внутри фигур, имеет важное значение для конкретной модели проектируемой системы, поскольку регламентирует реализацию соответствующих элементов в программном коде.

3. *Пути*, которые представляют собой последовательности из отрезков линий, соединяющих отдельные графические символы. При этом концевые точки отрезков линий должны обязательно соприкасаться с геометрическими фигурами, служащими для обозначения вершин диаграмм.

С одной стороны, с концептуальной точки зрения путям в языке UML придается особое значение, поскольку они являются простыми топологическими сущностями. С другой стороны, отдельные части пути или сегменты могут не существовать сами по себе вне содержащего их пути. Пути всегда соприкасаются с другими графическими символами на обеих

границах соответствующих отрезков линий. Другими словами, пути не могут обрываться на диаграмме линией, которая не соприкасается ни с одним графическим символом. Как отмечалось выше, пути могут иметь в качестве окончания, или терминатора, специальную графическую фигуру – значок, который изображается на одном из концов линий, являющихся сегментами этого пути.

4. *Строки текста*. Служат для представления различных видов информации в некоторой грамматической форме. Предполагается, что каждое использование строки текста должно соответствовать синтаксису в нотации языка UML, посредством которого может быть реализован грамматический разбор этой строки. Последний необходим для получения полной информации о модели. На использование строк накладывается важное условие – семантика всех допустимых символов должна быть заранее определена в языке UML или служить предметом его расширения в конкретной модели.

При графическом изображении диаграмм следует придерживаться следующих основных рекомендаций:

- Каждая диаграмма должна служить законченным представлением соответствующего фрагмента моделируемой предметной области. Отсутствие тех или иных элементов на диаграмме – признак неполноты модели, что может потребовать ее последующей доработки.
- Все сущности на диаграмме модели должны быть одного концептуального уровня. В случае достаточно сложных моделей систем желательно придерживаться стратегии последовательного уточнения или детализации отдельных диаграмм.
- Вся информация о сущностях должна быть явно представлена на диаграммах.
- Диаграммы не должны содержать противоречивой информации. Противоречивость модели может стать причиной серьезнейших проблем при ее реализации и последующем использовании на практике. Наличие элементов с одинаковыми именами и различными атрибутами свойств в одном пространстве имен также приводит к неоднозначной интерпретации и может служить источником проблем.
- Диаграммы не следует перегружать текстовой информацией. Принято считать, что визуализация модели является наиболее эффективной, если она содержит минимум пояснительного текста. Как правило, на-

личие больших фрагментов развернутого текста – признак недостаточной проработанности модели или ее неоднородности, когда в рамках одной модели представляется различная по характеру информация.

- Каждая диаграмма должна быть самодостаточной для правильной интерпретации всех ее элементов и понимания семантики всех используемых графических символов. Любые пояснительные тексты, которые не являются собственными элементами диаграммы (например, комментариями), не должны приниматься во внимание разработчиками.
- Количество типов диаграмм для конкретной модели приложения не является строго фиксированным. Для простых приложений нет необходимости строить все без исключения типы диаграмм. Некоторые из них могут просто отсутствовать в проекте системы, и этот факт не будет считаться ошибкой разработчика. Важно понимать, что перечень диаграмм зависит от специфики конкретного проекта системы.

Любая из моделей системы должна содержать только те элементы, которые определены в нотации языка UML. Имеется в виду требование начинать разработку проекта, используя только те конструкции, которые уже определены в метамодели UML. Как показывает практика, этих конструкций вполне достаточно для представления большинства типовых проектов программных систем. И только в случае отсутствия необходимых базовых элементов языка UML следует использовать механизмы их расширения для адекватного представления конкретной модели системы. При этом не допускается какое бы то ни было переопределение семантики тех элементов, которые отнесены к базовой нотации метамодели языка UML.

Процесс построения отдельных типов диаграмм имеет свои особенности, которые тесно связаны с семантикой элементов этих диаграмм.

1.3. Диаграмма прецедентов

Визуальное моделирование в UML можно представить как некоторый процесс поуровневого спуска от наиболее общей и абстрактной концептуальной модели исходной системы к логической, а затем и к физической модели соответствующей программной системы. Для достижения этих целей вначале строится модель в форме так называемой *диаграммы прецедентов* (use case diagram), которая описывает функциональное назна-

чение системы или, другими словами, то, что система будет делать в процессе своего функционирования (рис. 2). Диаграмма прецедентов является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.

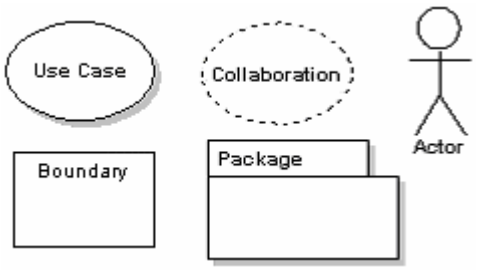

Рис. 2. Элементы диаграммы прецедентов

Разработка диаграммы прецедентов преследует следующие цели:

1. Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы.

2. Сформулировать общие требования к функциональному поведению проектируемой системы.

3. Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.

4. Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть данной диаграммы состоит в том, что проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых прецедентов.

Актером, или действующим лицом, называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик (см. рис. 2).

Прецедент служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый прецедент определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой (см. рис. 2).

В самом общем случае диаграмма прецедентов представляет собой граф специального вида, который является графической нотацией для представления конкретных прецедентов, актеров, возможно некоторых интерфейсов, и отношений между этими элементами. При этом отдельные компоненты диаграммы могут быть заключены в прямоугольник, обозначающий проектируемую систему в целом. Следует отметить, что отноше-

ниями данного графа могут быть только некоторые фиксированные типы взаимосвязей между актерами и прецедентами, которые в совокупности описывают сервисы или функциональные требования к моделируемой системе. На рис. 3 представлен пример диаграммы прецедентов.

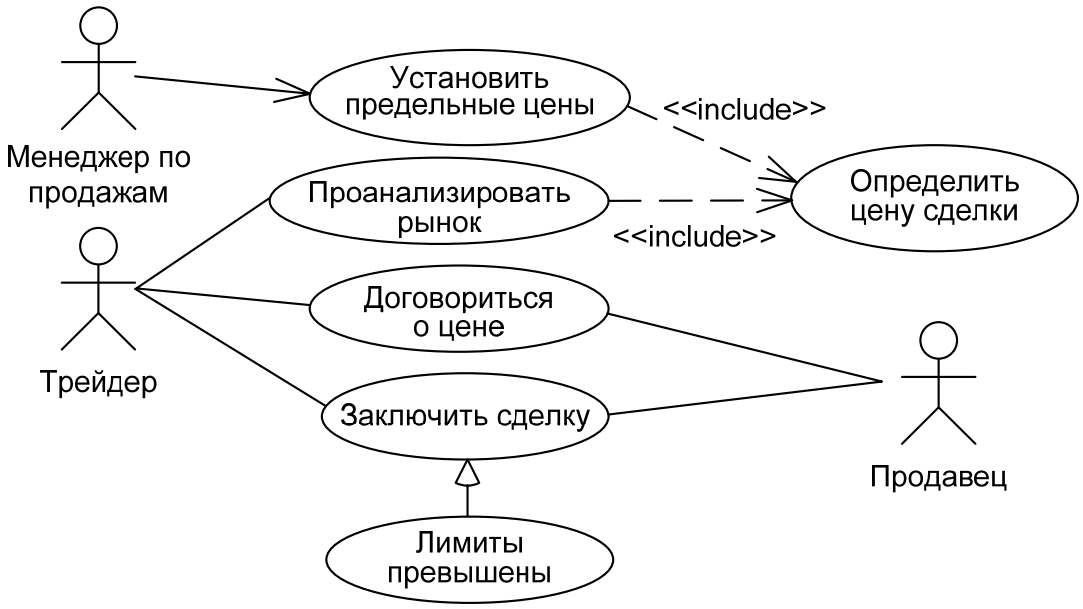

Рис. 3. Пример диаграммы прецедентов

1.3.1. Особенности построения диаграмм прецедентов

При построении диаграмм прецедентов актеров нужно связывать с прецедентами, один актер может выполнять несколько прецедентов, и наоборот, у одного прецедента может быть несколько актеров, которые его выполняют. При построении диаграмм прецедентов нельзя использовать связи между двумя актерами. При построении диаграмм прецедентов актер всегда инициализирует тот или иной прецедент, иными словами, стрелка всегда должна идти от актера к прецеденту.

Помимо связей между актерами и прецедентами, на диаграммах могут быть также представлены и отношения между прецедентами. Отношение *включения* (include) используется, когда в системе имеется фрагмент, повторяющийся многократно в нескольких прецедентах, и вы не хотели бы, чтобы его описание копировалось в каждом из этих прецедентов (см. рис. 3). Отношение *расширения* (extend) используется тогда, когда имеется

прецедент, который подобен другому прецеденту, но намного шире его, кроме того, при построении модели расширяющий прецедент может дополнять поведение базового прецедента, но для этого в базовом прецеденте должны быть определены так называемые точки расширения.

В самом языке UML пакет «Прецеденты» является подпакетом пакета «Элементы поведения». Последний специфицирует понятия, при помощи которых определяют функциональность моделируемых систем. Элементы пакета «Прецеденты» первичны по отношению к тем, с помощью которых могут быть описаны сущности, такие как системы и подсистемы. Однако внутренняя структура этих сущностей никак не описывается. Базовые элементы этого пакета – прецедент и актер.

1.3.2. Рекомендации по разработке диаграмм прецедентов

Главное назначение диаграммы прецедентов заключается в формализации функциональных требований к системе с помощью понятий соответствующего пакета и возможности согласования полученной модели с заказчиком на ранней стадии проектирования. Любой из прецедентов может быть подвергнут дальнейшей декомпозиции на множество прецедентов для отдельных элементов, которые образуют исходную сущность. Рекомендуемое общее количество актеров в модели – не более 20, а прецедентов – не более 50. В противном случае модель теряет свою наглядность и, возможно, заменяет собой одну из некоторых других диаграмм.

Семантика построения диаграммы прецедентов должна определяться следующими особенностями рассмотренных выше элементов модели. Отдельный экземпляр прецедента по своему содержанию является выполнением последовательности действий, которая инициализируется посредством экземпляра сообщения от экземпляра актера. В качестве отклика или ответной реакции на сообщение актера экземпляр прецедента выполняет установленную для него последовательность действий. Экземпляры актеров могут генерировать новые экземпляры сообщений для экземпляров прецедентов. Подобное взаимодействие будет продолжаться до тех пор, пока не закончится выполнение требуемой последовательности действий экземпляром прецедента. Окончание взаимодействия означает отсутствие

инициализации экземпляров сообщений от экземпляров актеров для соответствующих экземпляров прецедентов.

Прецеденты могут быть специфицированы в виде описания определенного вида, а в последующем – с помощью операций и методов вместе с атрибутами, в виде диаграммы видов деятельности, диаграммы состояний или любого другого механизма описания поведения, включающего предусловия и постусловия. Взаимодействие между прецедентами и актерами может уточняться на диаграмме коммуникации, когда описываются взаимосвязи между сущностью, содержащей эти прецеденты, и окружением или внешней средой этой сущности.

В случае, когда для представления иерархической структуры проектируемой системы используются подсистемы, система может быть определена в виде прецедентов на всех уровнях. Отдельные подсистемы или классы могут выступать в роли таких прецедентов. При этом прецедент, соответствующий некоторому из этих элементов, в последующем может уточняться множеством более мелких прецедентов, каждый из которых будет определять сервис элемента модели, содержащийся в сервисе исходной системы.

Функциональность, определенная для более общего прецедента, полностью наследуется всеми прецедентами нижних уровней. Однако следует заметить, что структура элемента-контейнера не может состоять из прецедентов, поскольку они могут представлять только функциональность отдельных элементов модели. Подчиненные прецеденты взаимодействуют для совместного выполнения прецедента верхнего уровня. Это взаимодействие также может быть представлено на диаграмме коммуникации в виде совместных действий отдельных элементов модели.

Отдельные прецеденты нижнего уровня могут играть определенную роль при выполнении сразу нескольких прецедентов верхнего уровня. Для отдельных таких взаимодействий могут быть определены соответствующие роли актеров, выполняющих конкретные прецеденты нижнего уровня. Эти роли будут играть актеры нижнего уровня модели системы. Хотя некоторые из таких актеров могут быть актерами верхнего уровня, это не противоречит принятым в языке UML семантическим правилам построения диаграмм прецедентов. Более того, интерфейсы прецедентов верхнего уровня могут полностью совпадать по своей структуре с соответствующими интерфейсами прецедентов нижнего уровня.

Прецеденты классов соответствуют операциям этого класса, поскольку сервис класса является, по существу, выполнением операций данного класса. Некоторые прецеденты могут соответствовать применению только одной операции, в то время как другие – конечного множества операций, определенных в виде последовательности операций. В то же время одна операция может быть необходима для выполнения нескольких сервисов класса и поэтому будет появляться в нескольких прецедентах этого класса.

Реализация прецедента зависит от типа элемента модели, в котором он определен. Например, поскольку прецеденты класса определяются посредством операций этого класса, они реализуются соответствующими методами. С другой стороны, прецеденты подсистемы реализуются элементами, из которых состоит данная подсистема. Поскольку подсистема не имеет своего собственного поведения, все реализуемые подсистемой сервисы должны представлять собой композицию сервисов, предлагаемых отдельными элементами этой подсистемы, т. е., в конечном итоге, классами. Эти элементы могут взаимодействовать друг с другом для совместного обеспечения требуемого поведения отдельного прецедента.

Если в качестве моделируемой сущности выступает система или подсистема самого верхнего уровня, то отдельные пользователи прецедентов этой системы моделируются актерами. Такие актеры, являясь внутренними по отношению к моделируемым подсистемам нижних уровней, часто в явном виде не указываются, хотя и присутствуют в модели подсистемы. Вместо этого прецеденты непосредственно обращаются к тем модельным элементам, которые содержат в себе подобных неявных актеров, т. е. экземпляры которых играют роли таких актеров при взаимодействии с прецедентами. Эти модельные элементы могут содержаться в других пакетах или подсистемах. В последнем случае роли определяются в том пакете, к которому относится соответствующая подсистема.

Важно понимать, что все сервисы системы должны быть явно определены на диаграмме прецедентов, и никаких других сервисов, которые отсутствуют на данной диаграмме, проектируемая система не может выполнять по определению.

1.4. Диаграмма классов

Диаграмма классов (class diagram) служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные отношения между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений.

Имеется два вида основных статических отношений:

- *ассоциации* (человек может сделать покупку в магазине);
- *подтипы* (корпоративный клиент является разновидностью клиента).

На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между объектами (рис. 4). На рис. 5 представлен пример диаграммы классов.

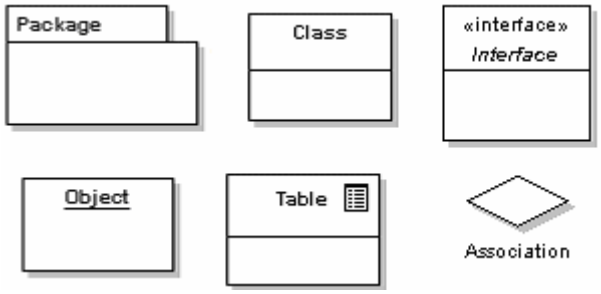

Рис. 4. Элементы диаграммы классов

Отношения ассоциации представляют собой отношения между экземплярами классов. Каждая ассоциация имеет два конца ассоциации**,** которыми она присоединяется к классам на диаграмме, а конец ассоциации, в свою очередь, обладает кратностью, которая показывает, сколько объектов может участвовать в данном отношении.

В общем случае кратность указывает верхнюю и нижнюю границы количества объектов, которые могут участвовать в отношении. Часто используемые варианты кратности:

1 – означает, что в ассоциации участвует один и только один экземпляр класса, с которым связана ассоциация;

* – в ассоциации может участвовать неограниченное число экземпляров класса;

0..1 – в ассоциации участвует либо один, либо ни одного экземпляра класса;

0..N – в ассоциации участвует от 0 до N экземпляров класса.

Стрелками в ассоциации обозначается направление навигации**,** таким образом, если в ассоциации присутствует стрелка, то она из симметричной преобразуется в одностороннюю (см. рис. 5).

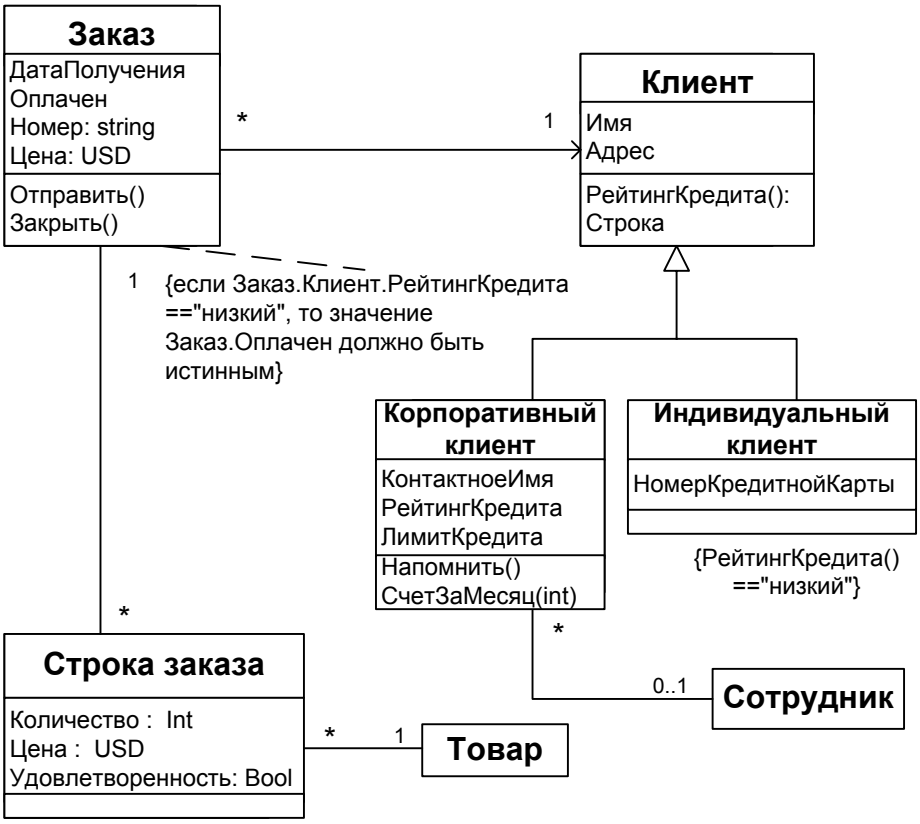

Рис. 5. Пример диаграммы классов

Если навигация указана только в одном направлении, то такая ассоциация называется однонаправленной, а если навигация указана с обеих сторон, то ассоциация считается двунаправленной. Если ассоциация на диаграмме не имеет стрелок навигации, то она является двунаправленной.

Связь, заданная при помощи ассоциации, существует в течение всего жизненного цикла объектов, даже если соединяемые ею экземпляры классов могут изменяться во времени.

Атрибуты являются элементами класса, определяющими его сущность. В синтаксисе UML описание атрибута выглядит следующим образом: <видимость><имя>:<тип>=<значение по умолчанию>. В примере на рис. 5 атрибутами являются: «Имя», «Адрес», «ЛимитКредита» и др.

Процессы, реализуемые классами, представляют собой *операции*. Синтаксис операции в UML выглядит следующим образом:

<видимость><имя>(<список параметров>):<выражение, возвращающее значение типа>(<строка свойств>), где:

- *видимость* – принимает одно из трех значений: «+» – общедоступная (public), «#» – защищенная (protected) либо «-» – закрытая (private);
- *имя* – строка символов;
- *список параметров* – содержит перечисленные через запятую параметры, которые описываются так же, как и атрибуты;
- *выражение, возвращающее значение типа* – содержит перечисленные через запятую значения типов;
- *строка свойств* – указывает свойства, которые имеются у данной операции.

При определении классов, атрибутов и операций и задании их имен и типов перед отечественными разработчиками встает вопрос: какой из языков использовать в качестве естественного, русский или английский? Наиболее целесообразно придерживаться следующих рекомендаций. При построении диаграммы прецедентов, являющейся наиболее общей концептуальной моделью проектируемой системы, применение русскоязычных терминов является не только оправданным с точки зрения описания структуры предметной области, но и эффективным с точки зрения взаимодействия с заказчиком и пользователями. При построении остальных типов диаграмм следует придерживаться разумного компромисса.

После разработки диаграммы классов процесс проектирования может быть продолжен в двух направлениях. Если поведение системы тривиально, то можно приступить к разработке диаграмм коммуникации и компонентов, однако для сложных динамических систем поведение – важнейший аспект их функционирования. Детализация поведения осуществляется последовательно при разработке диаграмм видов деятельности, последовательностей и состояний.

1.5. Диаграмма видов деятельности

Диаграмма видов деятельности (activity diagram) отражает динамику системы и особенно полезна при описании поведения, включающего в себя

большое количество параллельных процессов, а также для моделирования поведения системы в самом общем виде на этапе анализа (рис. 6).

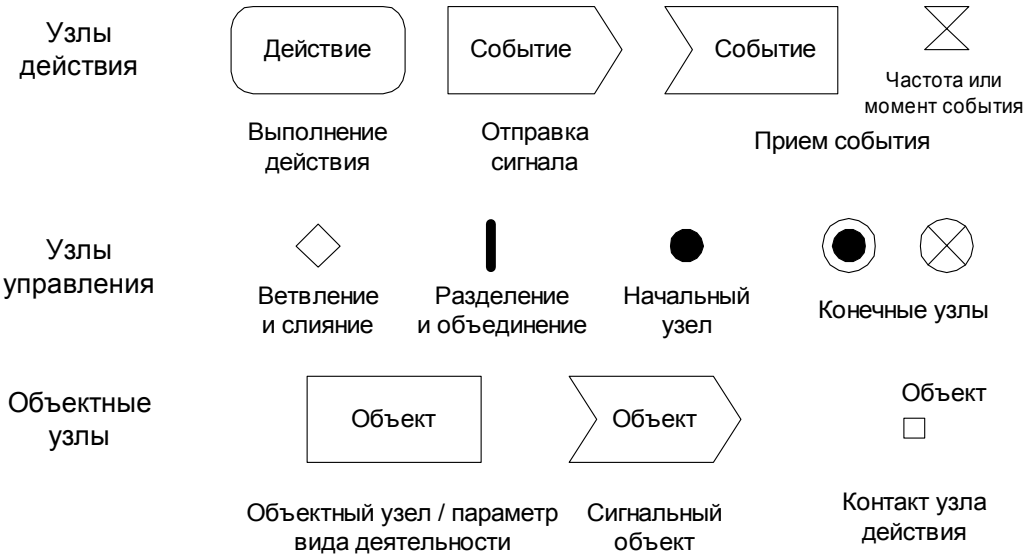

Рис. 6. Узлы диаграммы видов деятельности

В данном представлении поведения системы упор делается на ее функциональные составляющие, или *действия* (Action). Действие представляет собой базовую единицу функциональности системы, не декомпозируемую в рамках содержащего их вида деятельности. Все действия являются предопределенными. Например, в языке UML определены действия для создания объектов, установки значений атрибутов, связывания объектов и т.д. Кроме того, имеются действия для выполнения работ, определяемых пользователем (в том числе и видов деятельности). Действия могут иметь входы и выходы, называемые *контактами* (Pin), которые связываются ребрами потоков объектов. Контакты – это разновидность узлов объектов, поэтому они временно сохраняют значения данных в потоке. В простейшем случае для начала выполнения действия требуется наличие всех его входных данных.

Вид деятельности (Activity) – это спецификация поведения системы в виде координируемой последовательности действий.

Подобно всем разновидностям работ в UML, вид деятельности можно инициировать с помощью *вызывающего действия* (CallAction). Вид деятельности содержит *параметры* (ActivityParameterNode) для получения входных данных от вызывающего действия и предоставления ему выход-

ных данных. Параметры не являются контактами, поскольку последние используются для соединения действий в потоке, а вид деятельности – это работа, вызываемая действием. Для доступа к значениям параметров из действий внутри вида деятельности параметры моделируются как особый вид узлов потоков объектов. Параметры размещаются на границе диаграммы, и ребра потока объектов соединяют их с контактами.

Раздел (ActivityPartition) представляет собой средство группировки действий в соответствии с некоторым признаком. Действие может входить одновременно в несколько разделов. Раздел может содержать несколько подразделов. Особым видом разделов являются так называемые внешние разделы, используемые для представления сущностей, внешних по отношению к системе. Внешние разделы помечаются стереотипом <<external>>. Наиболее часто разделы применяются на этапе определения требований для моделирования подразделений организационной структуры или отдельных исполнителей в бизнес-моделях. Так, в примере на рис. 7 разделы «Отдел заказов», «Бухгалтерия» и «Покупатель» отражают исполнителей бизнес-процесса обработки заказа.

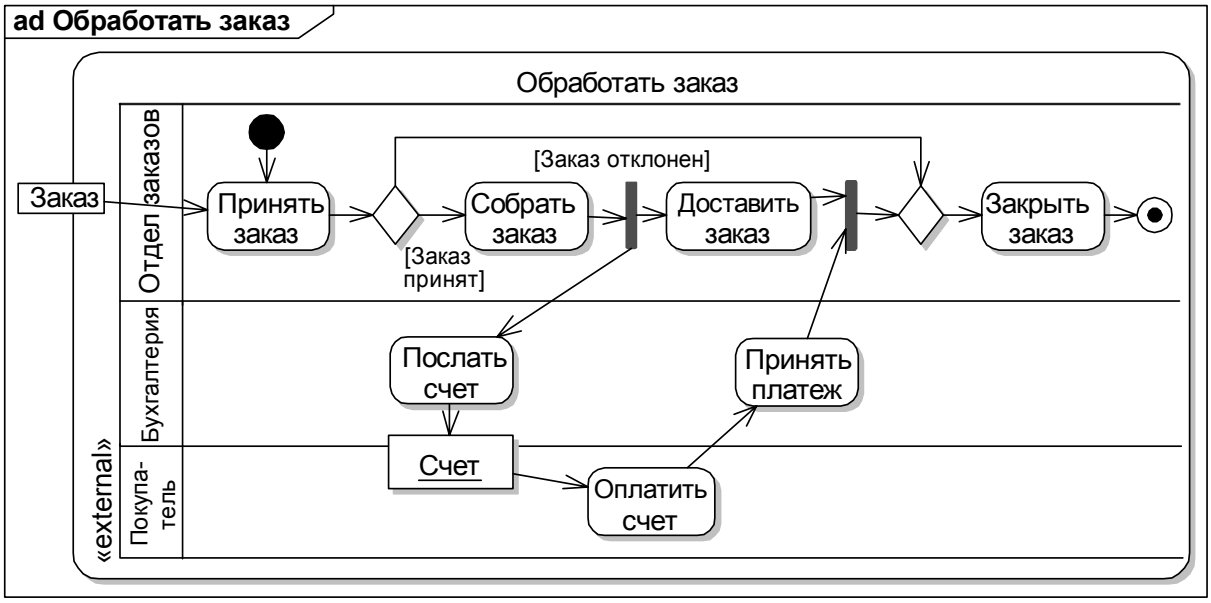

Рис. 7. Пример диаграммы видов деятельности

Диаграмма видов деятельности представляет собой граф, узлы которого соединены ребрами, представляющими потоки управления и данных. Маркеры управления и данных двигаются вдоль ребер и обрабатываются узлами, направляются на другие узлы или временно сохраняются. Каждые

узел и ребро определяют, когда через них могут проходить управляющие воздействия и значения данных. Эти правила движения маркеров можно комбинировать с целью прогнозирования поведения всего графа.

В диаграммах видов деятельности имеется три вида узлов (условные обозначения приведены на рис. 6).

Узлы действия оперируют получаемыми управляющими воздействиями и значениями данных и предоставляют управление и данные другим действиям.

Узлы управления маршрутизируют перемещение маркеров управления и данных по графу. Эти узлы содержат конструкции для выбора между альтернативными потоками, для параллельного движения по нескольким потокам и т.д.

Объектные узлы временно удерживают маркеры данных, которые ожидают продолжения движения по графу.

Узлы связываются направленными ребрами двух типов:

- *ребра потоков управления* связывают действия. Они обозначают, что действие на конце ребра со стрелкой не может начаться до того, как закончится исходное действие. По ребрам потоков управления могут проходить только маркеры управления;
- *ребра потоков объектов* соединяют узлы объектов (в том числе, контакты действий и параметры вида деятельности). По ребрам потоков объектов могут проходить только маркеры данных.

Действие активизируется только при наличии маркеров управления на всех входных ребрах потоков управления и маркеров данных на всех входных контактах. Перед выполнением действия соответствующие маркеры изымаются, а по его окончании на выходных ребрах потоков управления и выходных контактах размещаются маркеры управления и данных, после чего возможна активизация следующего действия.

Ветвление (DecisionNode) дает возможность показать разделение по условиям управляющих потоков. Данный узел имеет единственный вход и несколько выходов со сторожевыми условиями. Так как может выполниться только один из выходных переходов, то сторожевые условия должны взаимно исключать друг друга. Сторожевые условия изображаются в квадратных скобках около линии перехода. Если в качестве сторожевого условия используется [иначе], то это означает, что переход с данной меткой срабатывает в том случае, когда все другие переходы для данного ветвле-

ния являются ложными. *Слияние* (MergeNode) имеет несколько входов и единственный выход. Данный блок означает окончание условного поведения, которое было начато соответствующим ветвлением.

Параллельное поведение изображается при помощи *разделений* (ForkNode) и *объединений* (JoinNode). Последовательность выполнения параллельных действий является произвольной.

Конечный узел (ActivityFinalNode) завершает выполнение всех действий в виде деятельности (в том числе других видов деятельности, вызванных синхронно из текущего), удаляет все маркеры управления и данных, за исключением маркеров данных в выходных параметрах вида деятельности, и возвращает управление в вызвавшее его действие.

Конечный узел потока управления (FlowFinalNode) позволяет завершить отдельный поток управления, не затрагивая выполнение вида деятельности целиком. Это особенно полезно для моделирования видов деятельности, обрабатывающих потоки данных.

1.6. Диаграммы взаимодействия

Диаграммы взаимодействия (interaction diagrams) представляют собой модели, которые необходимы для описания поведения взаимодействующих групп объектов.

В языке UML существует четыре вида диаграмм взаимодействия: диаграмма последовательностей, диаграмма коммуникации, обзорная диаграмма взаимодействия и временная диаграмма (в пособии не рассматривается). Эти типы диаграмм дополняют друг друга, представляя взаимодействие элементов системы в динамике, но под различными углами зрения.

1.6.1. Диаграмма последовательностей

Диаграмма последовательностей (sequence diagram) отражает поток событий, происходящих при реализации некоторого прецедента, на этой диаграмме изображаются актеры, объекты, а также принимаемые и посылаемые ими сообщения (рис. 8).

При построении диаграммы последовательностей в первую очередь отражается временная последовательность происходящих событий. На рис. 9 представлен пример диаграммы последовательностей, которая имеет два измерения: вертикальное направление представляет время, горизонтальное – различные объекты.

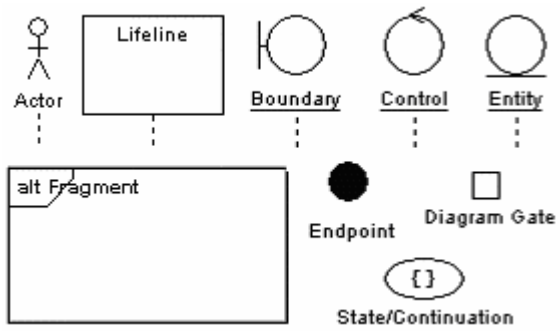

Рис. 8. Элементы диаграммы последовательностей

Объект (Object) – это экземпляр класса, конкретная сущность или образец, он может инициировать некоторые события, которые могут влиять на систему. На диаграмме последовательностей все объекты расположены последовательно в верхней ее части, за исключением объектов, создаваемых в результате тех или иных сообщений (примерами последних являются объекты Order и Invoice на рис. 9).

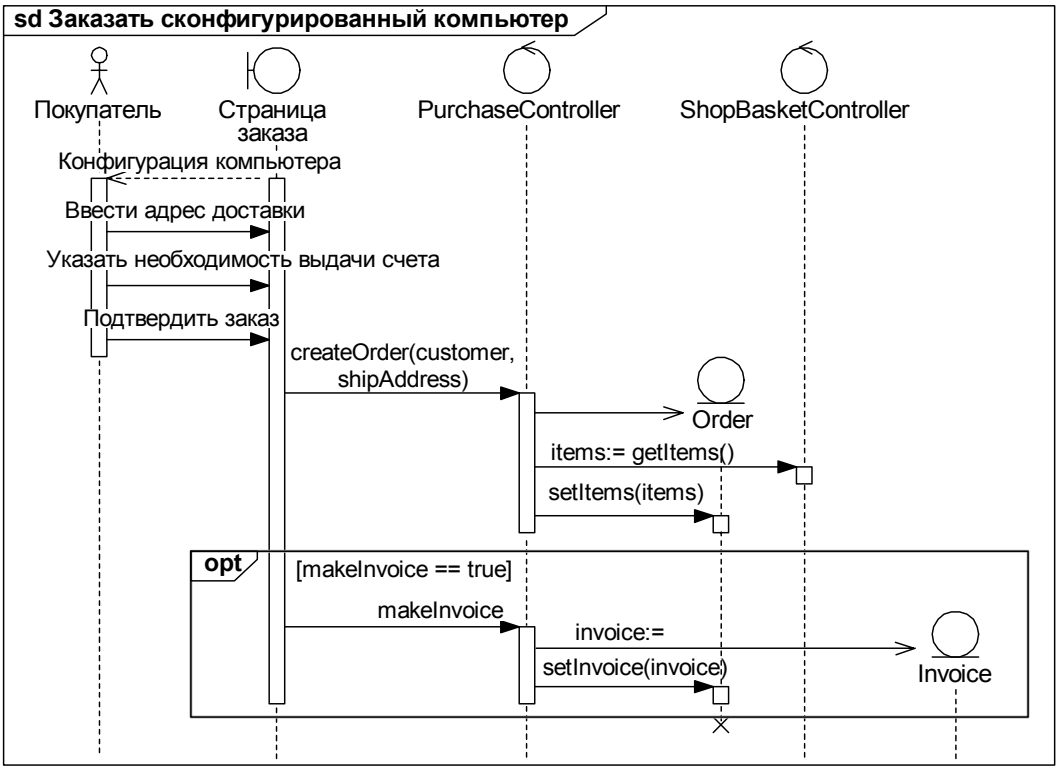

Рис. 9. Пример диаграммы последовательностей

От каждого объекта вниз отходит штриховая линия, называемая *линией жизни* (Lifeline) объекта. На ней показывают все, что происходит с объектом с момента его создания и до разрушения.

Частным случаем объекта может быть актер, он используется в том случае, когда необходимо показать связь участников и прецедентов.

Сообщения, передающиеся от одного объекта к другому, представляются стрелками между линиями жизни этих объектов. Порядок следования сообщений устанавливается сверху вниз. Каждое сообщение имеет как минимум *имя сообщения*, а также может иметь аргументы и некоторую управляющую информацию.

Для того чтобы показать момент времени, в который объект является активным, используются *прямоугольники активности,* их изображают поверх линии жизни.

Управляющая информация может быть представлена в виде условия, которое указывает, когда сообщение может быть передано, либо при помощи *маркера итерации*, показывающего, что сообщение посылается множество раз. Такая итерация дается в следующем виде: *[условие итерации].

В зависимости от типа сообщение изображается при помощи различных линий:

——▶ – вызов процедуры или другого потока управления;

——> – поток управления, показывает направление потока управления и последовательности шагов;

——) – асинхронное стимулирующее воздействие;

- - - -> – возврат из процедуры, используется для того, чтобы показать, что каждая процедура возвращает управление после своего завершения. Обычно стрелка возврата указывается только в том случае, если это вносит в диаграмму дополнительную ясность.

В качестве элементов диаграммы последовательностей могут применяться так называемые *фрагменты взаимодействия* (InteractionFragment), которые по своим свойствам не отличаются от полного взаимодействия объектов и позволяют наглядным образом группировать последовательности сообщений.

Включение (InteractionUse) представляет собой ссылку на имеющееся в модели взаимодействие вместо копирования последнего на диаграмму, что поддерживает декомпозицию и повторное использование отдельных взаимодействий. Включение обозначается как фрагмент взаимодействия с оператором «ref», содержимое которого состоит из названия соответствующего взаимодействия.

Комбинированный фрагмент (CombinedFragment) представляет собой выражение над фрагментами взаимодействия, состоящее из оператора и операндов – фрагментов взаимодействия. С каждым операндом может быть связано сторожевое условие. В языке UML над фрагментами взаимодействий определены следующие основные операторы:

- alt – выполняется не более одного операнда, для которого сторожевое условие истинно;
- opt – необходимость выполнения единственного операнда определяется его сторожевым условием;
- break – единственный операнд выполняется вместо всех остальных сообщений в объемлющем фрагменте взаимодействия в случае, если его сторожевое условие истинно;
- loop – циклическое выполнение единственного операнда, минимальное и максимальное количество итераций указывается в операнде;
- par – параллельное выполнение операндов при сохранении последовательности сообщений в каждом из них;
- strict – строго последовательное выполнение операндов;
- seq – условно последовательное выполнение операндов, при котором упорядочиваются только сообщения, относящиеся к одной и той же линии жизни; в последнем случае сообщение из первого операнда выполняется прежде сообщения из второго операнда и т.д.;
- critical – критический участок, операнды выполняются без перекрытия во времени с любыми другими сообщениями, относящимися к тем же линиям жизни, которые задействованы в операндах.

На рис. 9 приведен пример условного комбинированного фрагмента opt, включающего единственный операнд, для которого определено необходимое сторожевое условие.

Инвариант состояния (StateInvariant) представляет собой ограничение на состояние объекта в процессе выполнения взаимодействия. Если данное ограничение не выполняется, то вся предыдущая последовательность сообщений считается неверной. Пример инварианта состояния приведен на рис. 12, где с его помощью показано, что возможна отмена только тех заказов, которые еще не доставлены.

Диаграммы последовательностей являются наглядным и легко читаемым средством описания функционирования системы. Они помогают быстрее разобраться в процессах поведения системы.

1.6.2. Диаграмма коммуникации

Диаграмма коммуникации (communication diagram) отображает ту же информацию, что и диаграмма последовательностей, но на диаграмме коммуникации зависимость от времени указывается посредством нумерации сообщений. На диаграмме коммуникации отражается распределение процессов между объектами и их зависимости друг от друга, что очень полезно при разработке различных проектов. Основной целью построения данной диаграммы является понимание структурной организации занятых в системе объектов, принимающих и передающих сообщения. На рис. 10 показан пример диаграммы коммуникации.

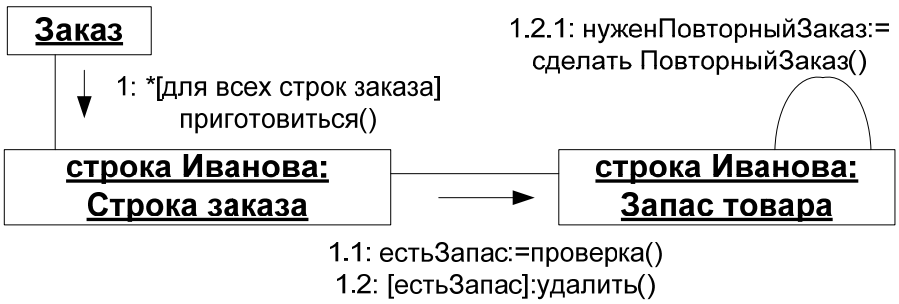

Рис. 10. Пример диаграммы коммуникации

В UML применяется десятичная схема нумерации, её использование позволяет понять, какая из операций вызывает другую операцию. Независимо от нумерации на диаграмме коммуникации, так же как и на диаграмме последовательностей, можно разместить управляющую информацию.

1.6.3. Обзорная диаграмма взаимодействия

Обзорная диаграмма взаимодействия (interaction overview diagram) отражает потоки управления, возникающие при реализации некоторого прецедента. Она является частным случаем диаграммы видов деятельности и может содержать те же элементы, с тем исключением, что вместо узлов

действий и объектов используются диаграммы взаимодействия или ссылки на них (условные обозначения приведены на рис. 11, пример – на рис. 12).

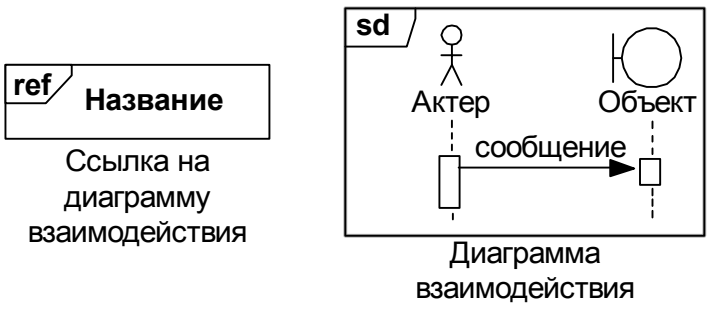

Рис. 11. Элементы обзорной диаграммы взаимодействия

Обзорная диаграмма взаимодействия позволяет альтернативным способом представить следующие виды комбинированных фрагментов взаимодействия: фрагменты alt и opt заменяются узлом разветвления и парным ему узлом слияния; фрагмент par заменяется узлом разделения и парным ему узлом объединения; фрагмент loop представляется в виде простого цикла.

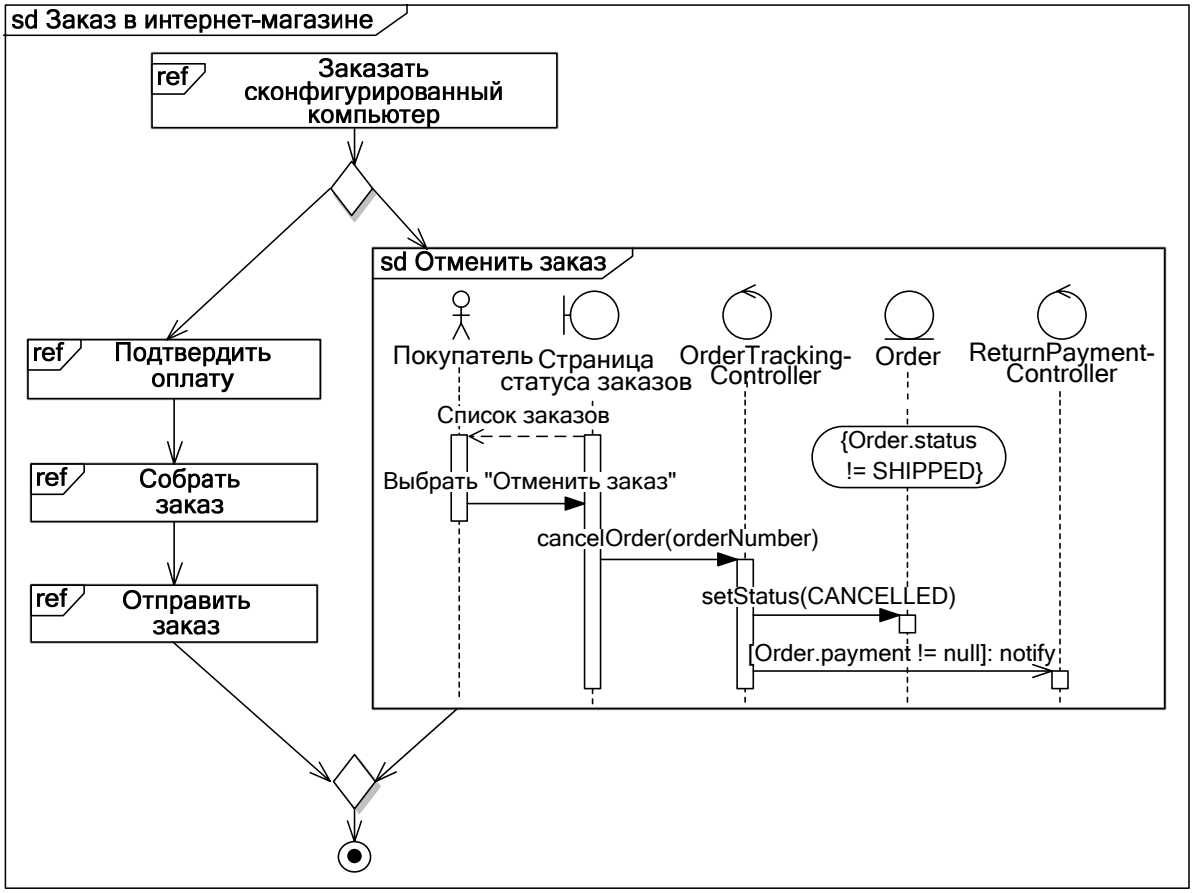

Рис. 12. Пример обзорной диаграммы взаимодействия

В силу этого в рассматриваемых диаграммах условные и параллельные потоки управления должны быть строго вложенными.

Данный вид диаграмм соединяет особенности диаграмм видов деятельности и диаграмм взаимодействия, акцентируя внимание разработчика не только на прохождении потоков управления, но и на реализации отдельных действий в виде взаимодействия объектов.

1.7. Диаграмма состояний

Диаграмма состояний (state machine diagram) отражает внутренние состояния объекта в течение его жизненного цикла от момента создания до разрушения, позволяя описать поведение объекта в различных прецедентах. Обычно диаграммы состояний строятся для единственного класса, чтобы показать динамику поведения единственного объекта. На рис. 13 представлен пример диаграммы состояний.

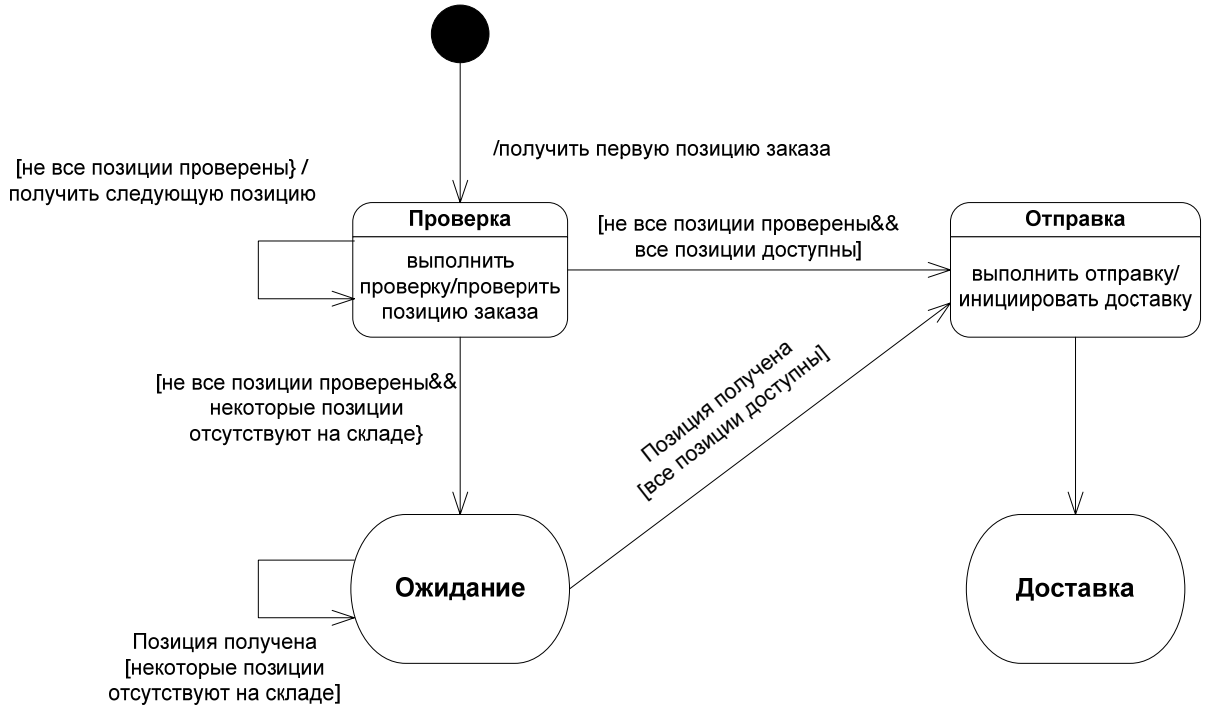

Рис. 13. Пример диаграммы состояний

Диаграмма состояний – это конечный автомат, реализованный средствами UML. Существует несколько разновидностей диаграмм состояний, в UML принята нотация Дэвида Харела.

Рассмотрим основные элементы диаграммы состояний (рис. 14).

Состояние (State) отображает одно из возможных состояний, в котором может находиться объект.

Кроме имени в элементе State может содержаться также краткое описание состояния и деятельности, осуществляемой в этом состоянии.

В общем случае состояние определяют следующие характеристики:

- *входное воздействие* – поведение, которое наступает при переходе объекта в данное состояние. Входное действие не прерывается и всегда выполняется до конца;
- *деятельность* – поведение, которое реализует объект, находящийся в данном состоянии;
- *выходное действие* – действие, которое выполняется при выходе объекта из текущего состояния.

Входные и выходные действия отображаются внутри графического элемента State под горизонтальной чертой, а друг от друга отделяются наклонной чертой «/» или двоеточием.

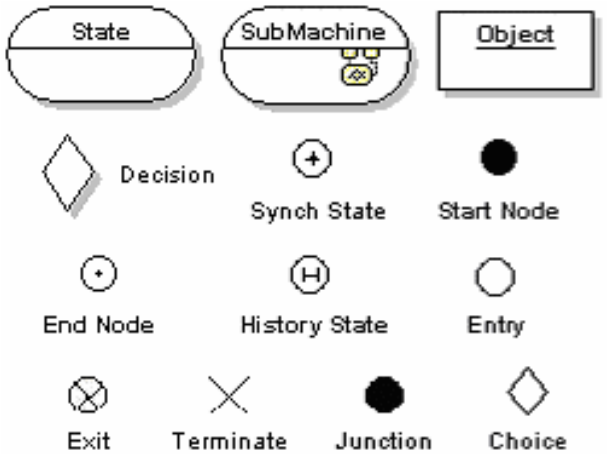

Рис. 14. Элементы диаграммы состояний

Переход (Transition) объекта из одного состояния в другое отображается направленной стрелкой. Синтаксис метки перехода состоит из трех частей, каждая из которых является необязательной: Событие:[Сторожевое условие]/Действие.

Событие (Event) – это некоторый факт, который инициирует переход из одного состояния в другое. События отображаются в виде поясняющей надписи около стрелки перехода.

Начальное состояние (Start) – это состояние, в котором объект находится непосредственно после его создания. Начальное состояние – это обязательный элемент диаграммы, причем на диаграмме может быть только один такой элемент (изображается черным кружком), стрелка перехода соединяет его с первоначальным состоянием объекта.

Конечное состояние (Stop) – состояние, в котором объект пребывает непосредственно перед его уничтожением. Это необязательный элемент, причем количество таких элементов на диаграмме не ограничено.

Глава 2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ ПО ТЕХНОЛОГИИ RUP*

В ходе работы над языком UML А. Джекобсоном были проанализированы распространенные в 1990-х гг. технологии разработки сложных программных систем, результатом чего явилась концепция унифицированного процесса разработки программного обеспечения (ПО) Rational Unified Process** (RUP), поддерживающего объектно-ориентированную технологию.

Технология RUP обладает следующими основными особенностями:

1. Итеративный характер процесса разработки ПО. В современных динамичных условиях практически невозможно создавать сложные программные системы последовательно, т. е. сначала выявлять все проблемы, затем принимать проектные решения, потом формировать программное обеспечение и, наконец, проверять полученное изделие. Итеративный подход позволяет улучшать понимание проблем на основе последовательных усовершенствований и конкретизировать их в эффективных решениях.

2. Управления проектной деятельностью на базе прецедентов, определяющих функционал системы. Прецеденты способствуют эффективному управлению технологическим маршрутом от бизнес-моделирования и определения требований вплоть до испытаний. Они обеспечивают связанные и доступные для анализа направления разработки и развертывания системы.

---

* При подготовке данного раздела были использованы данные, собранные В. В. Власенко.

** Унифицированный процесс компании Rational, или рациональный унифицированный процесс (*англ.*). Таким же названием (RUP) обладает продукт компании IBM, представляющий собой базу знаний по процессу разработки ПО и инструмент конфигурации процесса под нужды пользователя.

3. Ориентация на создание и физическое воплощение визуальных моделей. Основное внимание в RUP сосредоточивается на разработке и дальнейшем развитии надежной и гибкой архитектуры, которая облегчает параллельную разработку и минимизирует необходимость изменений.

Для практического введения в технологию RUP рассмотрим пример разработки системы онлайновой торговли.

Производитель компьютеров предлагает возможность приобретения своей продукции через Internet. Клиент может выбрать компьютер на Web-странице производителя. Компьютеры подразделяются на серверы, настольные и портативные. Заказчик может выбрать любую конфигурацию. Компоненты конфигурации представляются как список. Для каждой конфигурации предоставляется цена.

Чтобы оформить заказ, клиент должен заполнить информацию по доставке и оплате. В качестве платежных средств допускается использование кредитных карточек. После ввода заказа система отправляет клиенту по электронной почте сообщение с подтверждением получения заказа. Пока клиент ожидает прибытия компьютера, он может проверить состояние заказа в любое время. Серверная часть обработки заказа состоит из заданий, необходимых для проверки кредитоспособности и способа расчета клиента за покупку, из требования заказанной конфигурации, печати счета и подачи заявки на склад о доставке компьютера клиенту.

В процессе реализации системы подробно остановимся лишь на этапе проектирования системы.

В рамках RUP выделяются девять основных технологических процессов разработки программных систем, которые представляют собой логическое разбиение всех исполнителей и видов деятельности на соответствующие группы:

1) процесс управления проектом;
2) процесс моделирования производства;
3) процесс управления требованиями;
4) процесс анализа и проектирования;
5) процесс реализации;
6) процесс тестирования;
7) процесс управления конфигурацией и изменениями;
8) процесс управления средой;
9) процесс распространения.

Для решения поставленной задачи нам понадобятся лишь два из них: технологический процесс управления требованиями и технологический процесс анализа и проектирования. Посредством данных процессов мы продемонстрируем основные принципы проектирования программных систем в технологии RUP, а также покажем, как этапы проектирования взаимодействуют между собой.

2.1. Технологический процесс управления требованиями

Проектирование системы онлайновой торговли начинается с процесса управления требованиями. Рассмотрим основные элементы данного технологического процесса применительно к нашей системе.

Выявление требований

Требование – это условие или характеристика, которой должна соответствовать система. Требования представляются как подробное текстовое описание функций системы. Для эффективного управления всеми требованиями необходимо иметь полное понимание нужд пользователей и других заинтересованных лиц, которым должна удовлетворять разрабатываемая система.

Выявленные требования представлены в табл. 1.

Таблица 1

Распределение требований по актерам и прецедентам

Требование	Актер	Прецедент
1. Для знакомства со стандартной конфигурацией выбираемого компьютера клиент использует Web-страницу Internet-магазина. При этом также приводится цена конфигурации	Customer (Клиент)	Display Standard Configuration (Отображение стандартной конфигурации компьютера)
2. Клиент выбирает детали конфигурации, с которыми он хочет познакомиться, возможно, с намерением купить готовую или составить более подходящую конфигурацию. Цена для каждой конфигурации может быть подсчитана по требованию пользователя	Customer	Build Computer Configuration (Составление конфигурации компьютера)

Окончание табл. 1

Требование	Актер	Прецедент
3. Клиент может выбрать вариант заказа компьютера по Internet либо попросить, чтобы продавец связался с ним для объяснения деталей заказа, договорился о цене и т.п. прежде, чем заказ будет фактически размещен	Customer, Salesperson (Продавец)	Order Configured Computer (Заказ сконфигурированного компьютера), Request Salesperson Contact (Обращение с просьбой к продавцу)
4. Для размещения заказа клиент должен заполнить электронную форму с адресами для доставки товара и отправки счета-фактуры, а также деталями, касающимися оплаты (кредитная карточка или чек)	Customer	Order Configured Computer, Verify and Accept Customer Payment (Проверка и прием платежа от клиента)
5. После ввода заказа клиента в систему продавец отправляет на склад электронное требование, содержащее детали, касающиеся заказанной конфигурации	Salesperson, Warehouse (Склад)	Inform Warehouse About Order (Информирование склада о заказе)
6. Детали сделки, включая номер заказа, номер счета клиента, отправляются по электронной почте клиенту, так что заказчик может проверить состояние заказа через Internet	Salesperson, Customer	Order Configured Computer, Update Order Status (Обновление состояния заказа)
7. Склад получает счет-фактуру от продавца и отправляет компьютер клиенту	Salesperson, Warehouse	Print Invoice (Печать счета-фактуры)

Выявление актеров и прецедентов

Актеры и прецеденты определяются в результате анализа требований. После того как определены требования, выявим актеров и прецеденты и построим таблицу, которая распределяет требования по актерам и прецедентам (см. табл. 1).

Построение диаграммы прецедентов

Диаграмма прецедентов приписывает прецеденты к актерам; она также позволяет установить отношения между прецедентами, если таковые существуют. Диаграмма прецедентов представлена на рис. 15.

Составление документа описания прецедентов

Структура документа, описывающего прецеденты, может варьироваться, однако типичное описание должно содержать следующие разделы:

1) краткое описание;

2) участвующие актеры;

3) предусловия, необходимые для инициирования прецедента;

4) детализированное описание потока событий, которое включает:

– основной поток событий;

– альтернативные потоки для определения исключительных ситуаций;

5) Постусловия, определяющие состояние системы по завершении прецедента.

Документ описания прецедента развивается по ходу разработки. На ранней стадии определения требований составляется лишь краткое описание, остальные части документа создаются постепенно и итеративно.

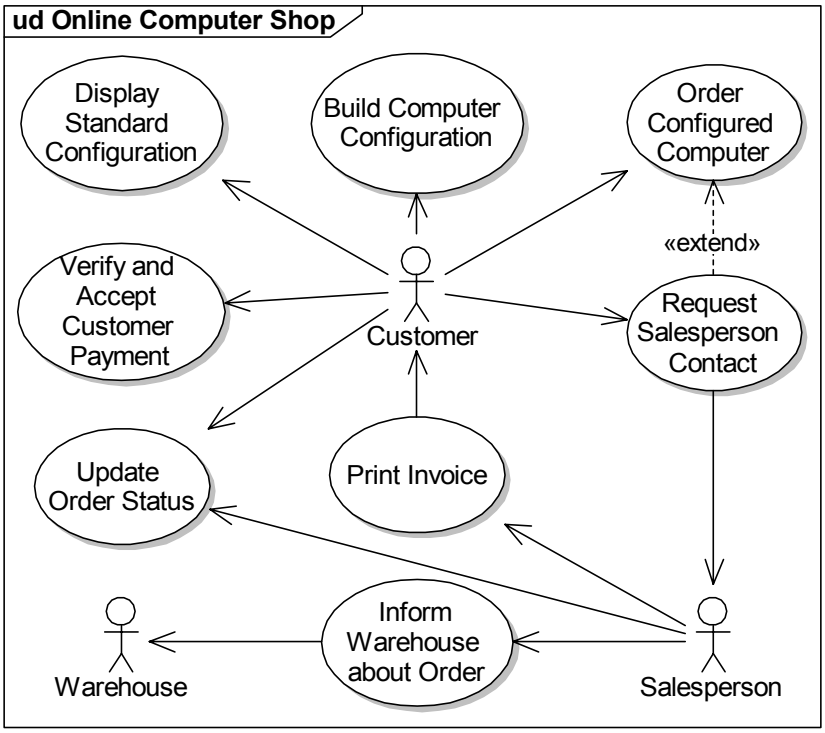

Рис. 15. Диаграмма прецедентов

Проектирование пользовательского интерфейса

Проектирование пользовательского интерфейса – это визуальное формирование пользовательского интерфейса, удовлетворяющего различным требованиям практичности.

Наибольшее значение при проектировании пользовательского интерфейса имеют модель прецедентов и дополнительные спецификации, разрабатываемые совместно с пользователями и другими заинтересованными сторонами.

На основе полученных результатов создается прототип пользовательского интерфейса. Далее полученные прототипы прикрепляются к соответствующим прецедентам (например, с помощью Rational RequisitePro).

2.2. Технологический процесс анализа и проектирования

Целью технологического процесса анализа и проектирования является перевод системных требований в технические инструкции, указывающие, как реализовать систему. Для этого следует понять требования и преобразовать их в проект системы, выбрав при этом лучшую стратегию реализации.

Процесс анализа и проектирования разбивается на отдельные этапы, на каждом из которых осуществляется разработка соответствующих типов канонических диаграмм модели системы. При этом на начальных этапах процесса разработки строятся логические представления статической модели структуры системы, затем – логические представления модели поведения, и лишь после этого – физические представления модели системы.

2.2.1. Определение потенциальной архитектуры

На данном этапе возникает необходимость выявить наиболее важные прецеденты, которые будут реализованы в текущей итерации. В качестве такого прецедента выберем прецедент Order Configured Computer (Заказ сконфигурированного компьютера) и именно для него будем проводить все последующие действия.

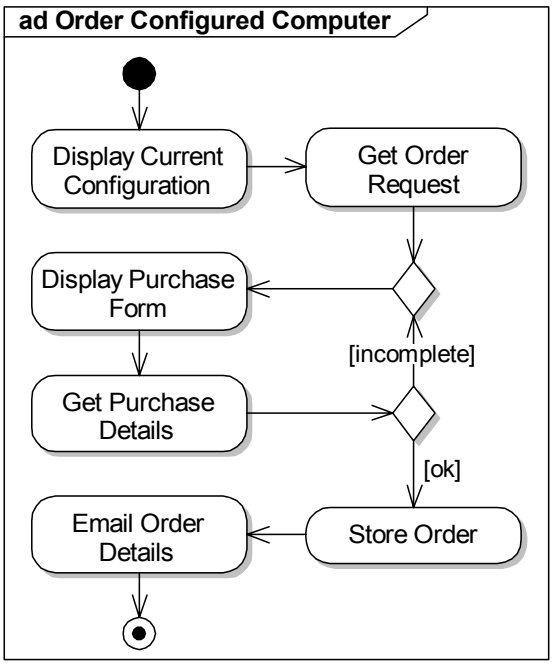

Рис. 16. Диаграмма видов деятельности

Построение диаграммы видов деятельности для выбранного прецедента (рис. 16) следует начать с определения возможных действий. В табл. 2 показаны события, относящиеся к основному и альтернативным потокам, заимствованные из документа описания прецедента, и приведены соответствующие действия.

В рассматриваемом примере используются только действия, вызывающие некоторое поведение системы, поэтому в таблице указаны только названия соответствующих поведений (это могут быть виды деятельности, конечные автоматы или взаимодействия).

Таблица 2

Определение возможных действий

Формулировка прецедента	Действие
1. Начало прецедента совпадает с решением клиента заказать сконфигурированный компьютер с помощью функции Continue (Продолжить) при отображении на экране детализированной информации о заказе	Display Current Configuration (Отображение текущей конфигурации); Get Order Request (Получение запроса на заказ)
2. Система просит клиента ввести детализированную информацию о покупке: детали касающиеся доставки, информация по оплате, способ оплаты и произвольные комментарии.	Display Purchase Form (Отображение закупочной формы)
3. Клиент выбирает функцию Purchase (Заказать) для отправки заказа	Get Purchase Details (Детализировать информацию о покупке)
4. Система присваивает уникальный номер заказа и клиентский учетный номер заказу на покупку и запоминает информацию о заказе в базе данных	Store Order (Запомнить заказ)
5. Система отправляет клиенту по электронной почте номер заказа и клиентский номер клиенту вместе со всеми деталями в качестве подтверждения принятия заказа	Email Order Details (Отправить детальную информацию по заказу)
6. Клиент инициирует функцию Purchase (Заказать) до того, как введет всю обязательную информацию. Система отображает на экране сообщение об ошибке и просит ввести пропущенную информацию	Get Purchase Details; Display Purchase Form
7. Клиент выбирает функцию Reset (Отмена), чтобы вернуться к исходной форме. Система дает возможность клиенту вновь ввести информацию	Display Purchase Form

Определение внутреннего состояния системы дается в модели классов. Аналогично табл. 1 можно построить таблицу, которая поможет выявить классы в результате анализа требований (табл. 3).

Таблица 3

Соответствие требований и классов-сущностей

Требование	Класс-сущность
1. Для знакомства со стандартной конфигурацией выбираемого компьютера клиент использует Web-страницу Internet-магазина.	Customer (Клиент); Computer (Компьютер); StandardConfiguration (Стандартная конфигурация); Product (Товар)
2. Клиент выбирает детали конфигурации, с которыми он хочет познакомиться, возможно, с намерением купить готовую или составить более подходящую конфигурацию. Цена для каждой конфигурации может быть подсчитана по требованию пользователя	Customer; ConfiguredComputer (Сконфигурированный компьютер); ConfigurationItem (Элемент конфигурации)
3. Клиент может выбрать вариант заказа компьютера по Internet либо попросить, чтобы продавец связался с ним для объяснения деталей заказа, договорился о цене и т.п. прежде, чем заказ будет фактически размещен	Customer; ConfiguredComputer; Order (Заказ); Salesperson (Продавец)
4. Для размещения заказа клиент должен заполнить электронную форму с адресами для доставки товара и отправки счета-фактуры, а также деталями, касающимися оплаты (кредитная карточка или чек)	Customer; Order; Shipment (Поставка); Invoice (Счет-фактура); Payment (Платеж)
5. После ввода заказ клиента в систему продавец отправляет на склад электронное требование, содержащее детали, касающиеся заказанной конфигурации	Customer; Order; Salesperson; ConfiguredComputer; ConfigurationItem
6. Детали сделки, включая номер заказа, номер счета клиента, отправляются по электронной почте клиенту, так что заказчик может проверить состояние заказа через Internet	Customer; Order; OrderStatus (Состояние заказа)
7. Склад получает счет-фактуру от продавца и отправляет компьютер клиенту	Shipment; Invoice

Диаграмма классов дает обобщенное визуальное представление обо всех элементах модели классов. Диаграмма классов для разрабатываемой системы онлайновой торговли приведена на рис. 17.

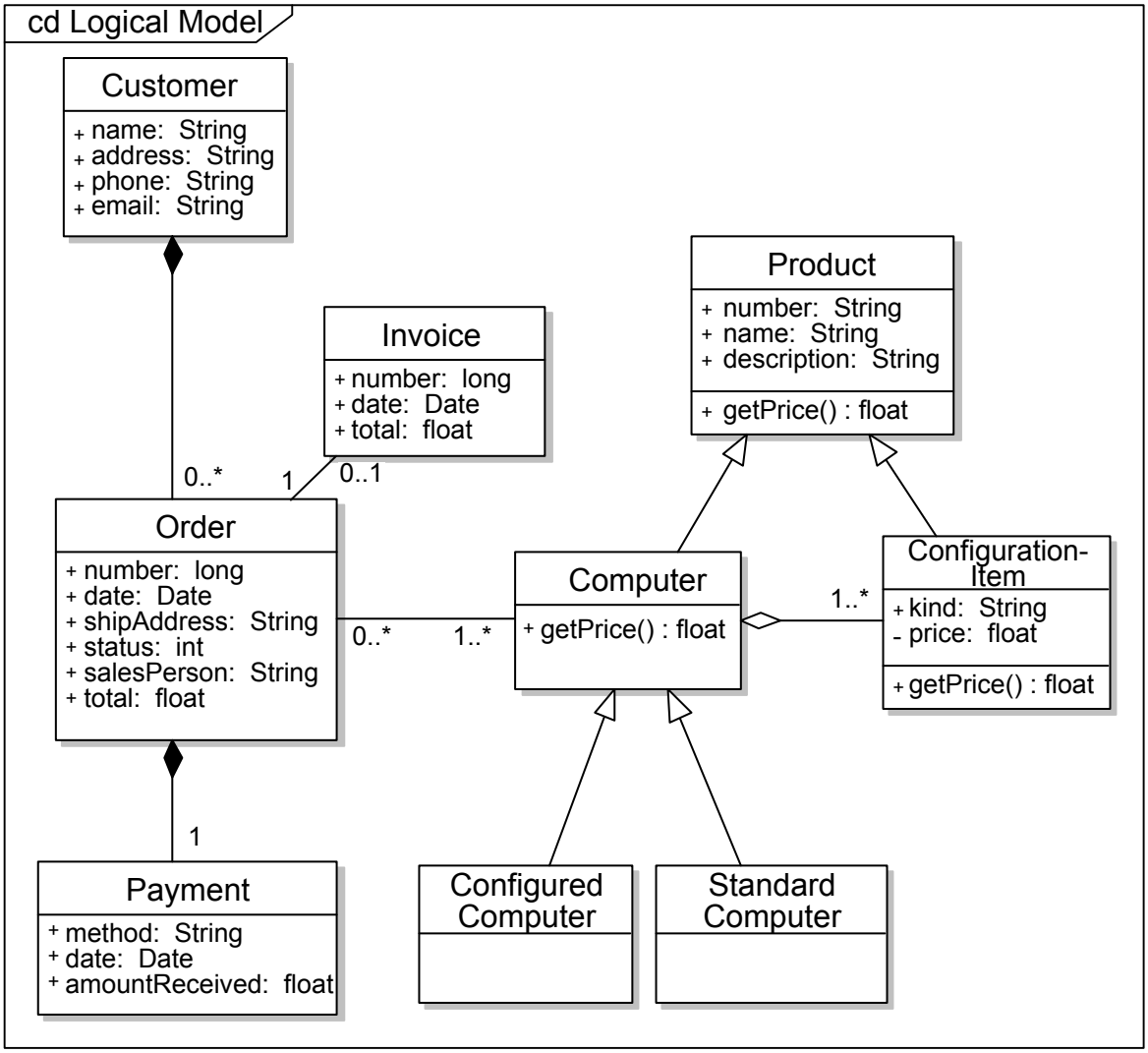

Рис. 17. Диаграмма классов

2.2.2. Уточнение архитектуры

Плавный переход от анализа к проектированию напрямую связан с процессом моделирования взаимодействий, в рамках которого определяется последовательность событий (сообщений) в их связи с действующими в кооперации объектами.

Диаграмма коммуникаций, как правило, используется на этапе системного проектирования для сложных и больших проектов; в нашем примере она рассматриваться не будет.

Сначала строится обзорная диаграмма взаимодействия, которая включает в себя ссылки на диаграммы последовательностей, каждая из которых раскрывает логику соответствующего ей прецедента (см. рис. 12). Для каждого прецедента строится отдельная диаграмма последовательностей (см. рис. 9).

2.2.3. Анализ поведения

Целью данного элемента процесса является «преобразование» последовательности событий в последовательность поведений. Таким «преобразованием» является модель состояний.

Модель состояний служит детализированным описанием класса или, более точно, динамических изменений состояний класса. Модель состояний определяет возможные состояния, в которых может находиться класс, и эффективно фиксирует «жизненный путь» класса. На протяжении своего жизненного цикла объект остается одним и тем же – его идентичность никогда не изменяется, изменяется лишь его состояние.

Диаграмма состояний обычно присоединяется к классу, определяя способ реагирования объектов класса на события. Другими словами, диаграмма определяет (для каждого состояния объекта) действие, выполняемое объектом при получении им сигнала о событии. Один и тот же объект может выполнять различные действия в ответ на одно и то же событие в зависимости от состояния объекта. Диаграмма состояний для класса Order представлена на рис. 18.

2.2.4. Проектирование компонентов

Данный элемент технологического процесса состоит из следующих видов деятельности:

1. Проектирование диаграммы пакетов для прецедентов;
2. Проектирование диаграммы пакетов для классов;
3. Проектирование диаграммы компонентов;
4. Проектирование диаграммы развертывания;
5. Составление проектного документа по прецедентам.

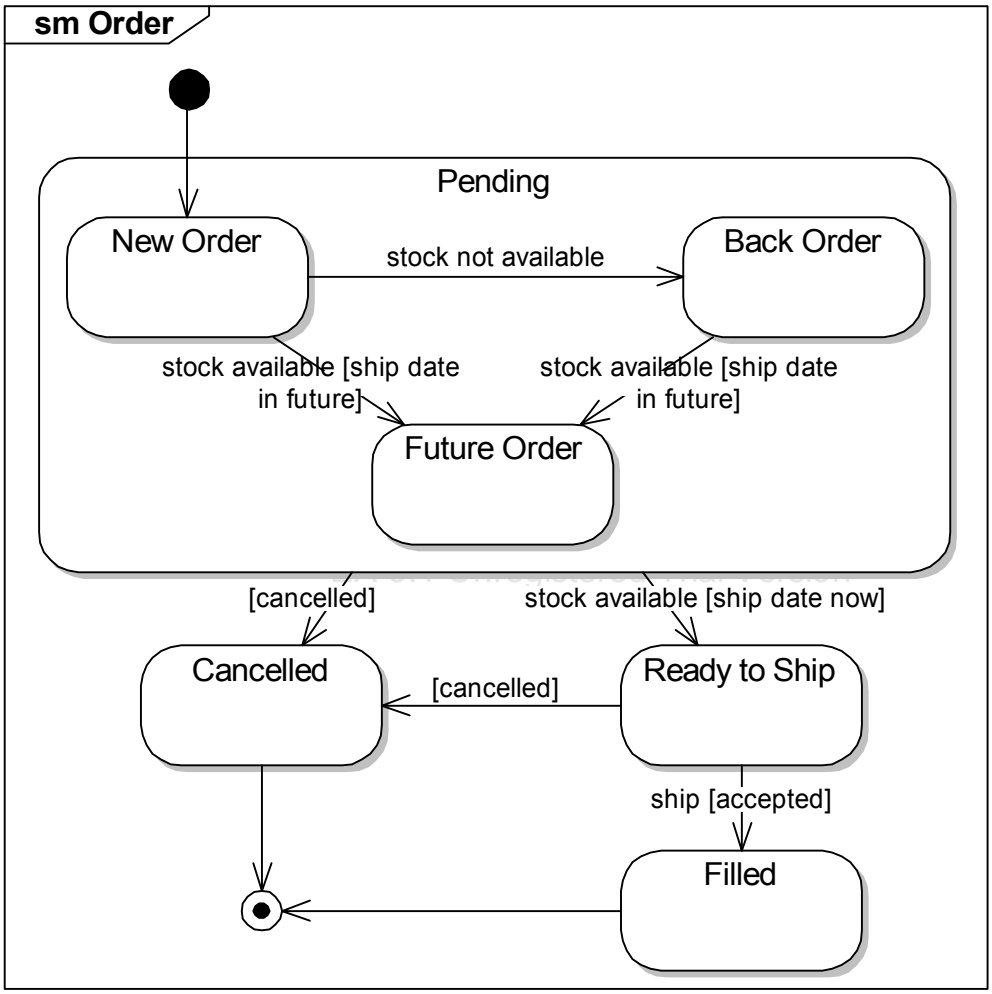

Рис. 18. Диаграмма состояний для класса Order

1. Проектирование диаграммы пакетов для прецедентов

Пакеты могут объединять в группы классы или, чаще всего, – прецеденты. Использование моделей пакетов оправдано только при структуризации больших систем. Небольшие системы можно понять и управлять ими, не прибегая к помощи пакетов.

Модель пакетов прецедентов практически не используется за пределами этапов проектирования, поскольку модель пакетов классов эффективно заменяет ее. На рис. 19 представлена модель пакетов, которые вы-

строились в соответствии с последовательностью выполнения бизнес-процессов для системы онлайновой торговли. Они отражают порядок, в котором клиенты, приобретающие компьютер, используют Web-страницы и формы.

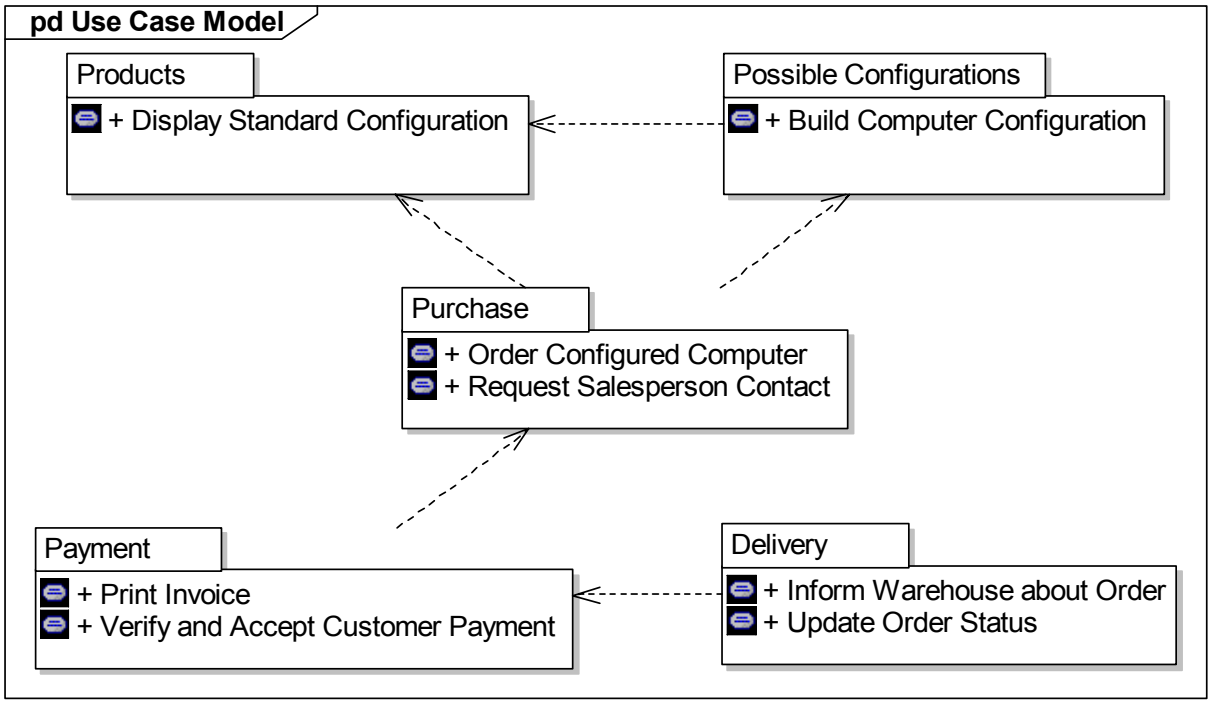

Рис. 19. Пакеты прецедентов

2. Проектирование диаграммы пакетов для классов

Модель пакетов классов играет ведущую роль при проектировании. Лучший способ «расправиться» с примером – «сымитировать» систему и представить, что необходимо сделать, чтобы принять заказ клиента на сконфигурированный компьютер. В соответствии с этим построим диаграмму пакетов для классов (рис. 20).

3. Проектирование диаграммы компонентов

Компоненты – это физические части системы. Следовательно, проектирование компонентов нельзя отделить от платформы реализации. Проектируемая система онлайновой торговли – это Web-приложение с сервером баз данных. Web-приложение – это Web-система, позволяющая ее пользователям работать в соответствии с заложенной в ней бизнес-логикой с помощью Web-браузера. Программа, поддерживающая бизнес-логику, может

находить на сервере и/или на клиенте. Следовательно, Web-приложение – не что иное, как разновидность системы клиент/сервер с Web-узлом.

Относительно нашего примера рассмотрим типичную последовательность доступа к Web-страницам. Первая Web-страница, которую может посетить пользователь, – это Web-страница поставщика, на которой перечислены группы изделий (серверные, настольные системы, портативные компьютеры) и приводятся ссылки на Web-страницы, на которых представлены перечни изделий и дано краткое описание каждого изделия. Это единый функциональный блок, который может образовать компонент ProductList (Перечень изделий) и т.д.

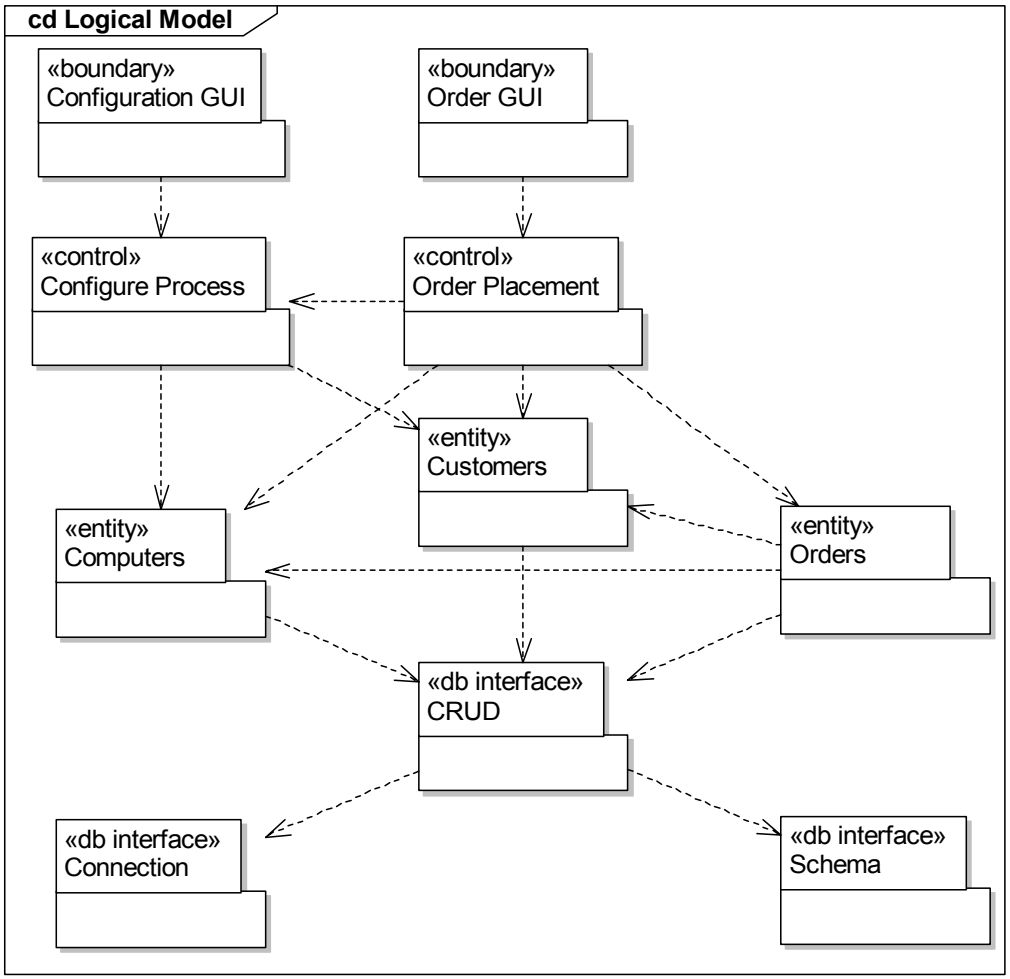

Рис. 20. Пакеты классов

В соответствии с этим построим диаграмму компонентов (рис. 21). Важно отметить сходство этой диаграммы с диаграммой пакетов преце-

дентов, что неудивительно, поскольку пакеты прецедентов и компоненты представляют собой функциональные модули с четким границами.

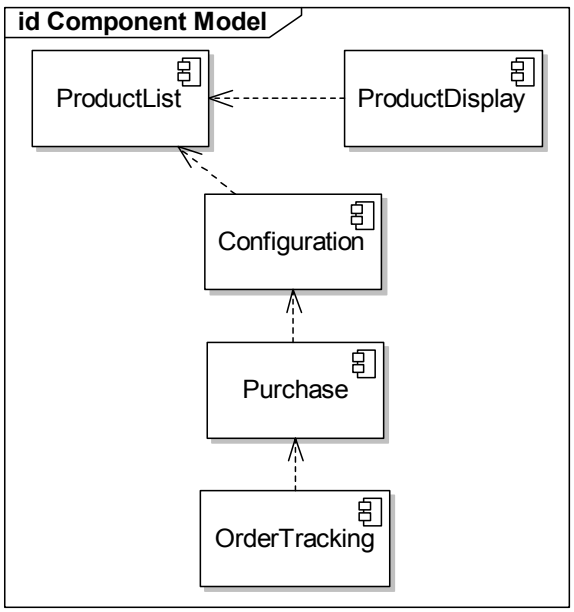

Рис. 21. Диаграмма компонентов

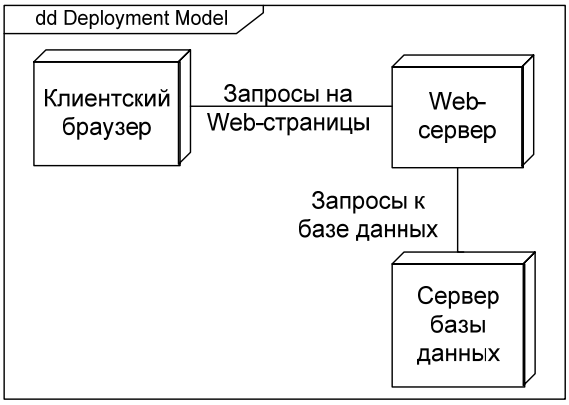

Рис. 22. Диаграмма развертывания

4. Проектирование диаграммы развертывания

Характер Internet-систем без установления прямого соединения делает развертывание Web-приложений значительно более сложной задачей, чем развертывание приложений баз данных в архитектуре клиент/сервер. Чтобы приступить к развертыванию, требуется установить Web-сервер в качестве пункта маршрутизации между всеми браузерами клиентов и базой данных. На рис. 22 показана диаграмма развертывания для нашего примера. Система онлайновой торговли развернута без отдельного сервера приложений.

5. Составление проектного документа по прецедентам

Цель данного этапа заключается в уточнении документа описания прецедентов таким образом, чтобы он представлял собой спецификацию прецедентов проектного уровня.

Далее соответствующий документ может быть использован в процессе тестирования для проектирования тестовых прецедентов.

Глава 3. РАЗРАБОТКА СИСТЕМ НА ОСНОВЕ МОДЕЛЬНО-ОРИЕНТИРОВАННОЙ АРХИТЕКТУРЫ

Модельно-ориентированная архитектура (Model Driven Architecture, далее – MOA) представляет собой стратегическую инициативу консорциума Object Management Group (OMG), пришедшую в конце 2000 г. на смену архитектуре управления объектами (Object Management Architecture), которая служила фундаментом платформы распределенных объектных вычислений CORBA.

Основным отличием модельно-ориентированного подхода от прочих (структурного, объектно-ориентированного) является выбор в качестве конечной цели разработки не конкретной реализации системы, а некоторой модели системы или совокупности таких моделей (например, модели организации согласно архитектуре Захмана или иной аналогичной методологии). При этом происходит перераспределение ресурсов времени и персонала: уменьшение значимости стадии кодирования и прирост стадии анализа и проектирования.

Средства MOA позволяют реализовать одну и ту же функциональную модель системы на множестве платформ. Благодаря данному подходу можно интегрировать существующие и разрабатываемые приложения путем связывания их моделей, что обеспечивает их интероперабильность и поддерживает эволюцию ИС по мере смены платформ и технологий.

Основные понятия MOA – модель, платформа и отображение.

Модель в MOA является представлением части функциональности, структуры или поведения системы в виде формальной спецификации. Спецификация считается формальной, если она выражена средствами языка с четко определенным синтаксисом и семантикой (подобный язык также принято называть метамоделью). Таким образом, любой модели в MOA должно быть сопоставлено определение языка моделирования. Отметим, что исходный код приложения также является моделью, так как базируется

на некотором языке программирования, обязательно обладающим синтаксисом и семантикой. Обычно под «платформой» понимается инфраструктура для программного обеспечения, реализуемая на определенной аппаратной базе с использованием определенной технологии. Так, в наиболее общем виде различают Unix-, Windows-платформы, а при рассмотрении программных средств промежуточного слоя (то есть инфраструктуры распределенного ПО) выделяют платформы CORBA, J2EE и Microsoft.NET. В рамках концепции MOA «платформа» обозначает особенности используемой технологии, не влияющие на базовую функциональность разрабатываемого ПО.

Наиболее абстрактными являются бизнес-модели, так как описываемые ими объекты (процессы, ресурсы) существуют независимо от вычислительной поддержки (так называемые computation-independent, или CIM-модели). Модель называется платформо-независимой (platform-independent model, или PIM-модель), если она абстрагирована от особенностей конкретной технологии. Соответственно, все модели, отражающие аспекты применения тех или иных технологий, являются платформо-зависимыми (platform-specific models, или PSM-модели). На рис. 23 приведен пример PIM-модели, а на рис. 24 – пример PSM-модели, уточняющей первую модель подробностями реализации на платформе J2EE.

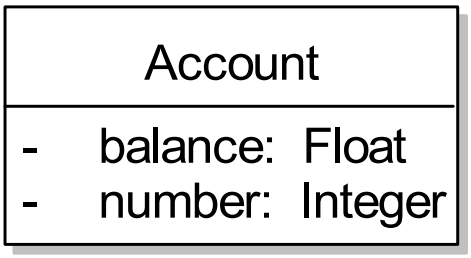

Рис. 23. Платформо-независимая модель

Подобное разделение обладает следующими положительными качествами:

1) проверка корректности PIM-модели значительно проще по сравнению с PSM-моделью, так как нет необходимости проверять особенности компонентной архитектуры, способы передачи параметров и реализации ассоциаций и т.п.;

2) возникает возможность реализации проекта на различных платформах с сохранением структуры и функциональности разрабатываемой системы; проектируемая ИС может быть описана, в том числе, на уровне бизнес-модели (то есть независимо от возможностей применения средств вычислительной техники), что открывает перед разработчиками еще более широкие возможности;

3) взаимодействие между отдельными приложениями может быть описано на высоком уровне абстракции, а затем по PIM-модели проектируются механизмы обеспечения взаимодействия для конкретной платформы (так называемая glue logic).

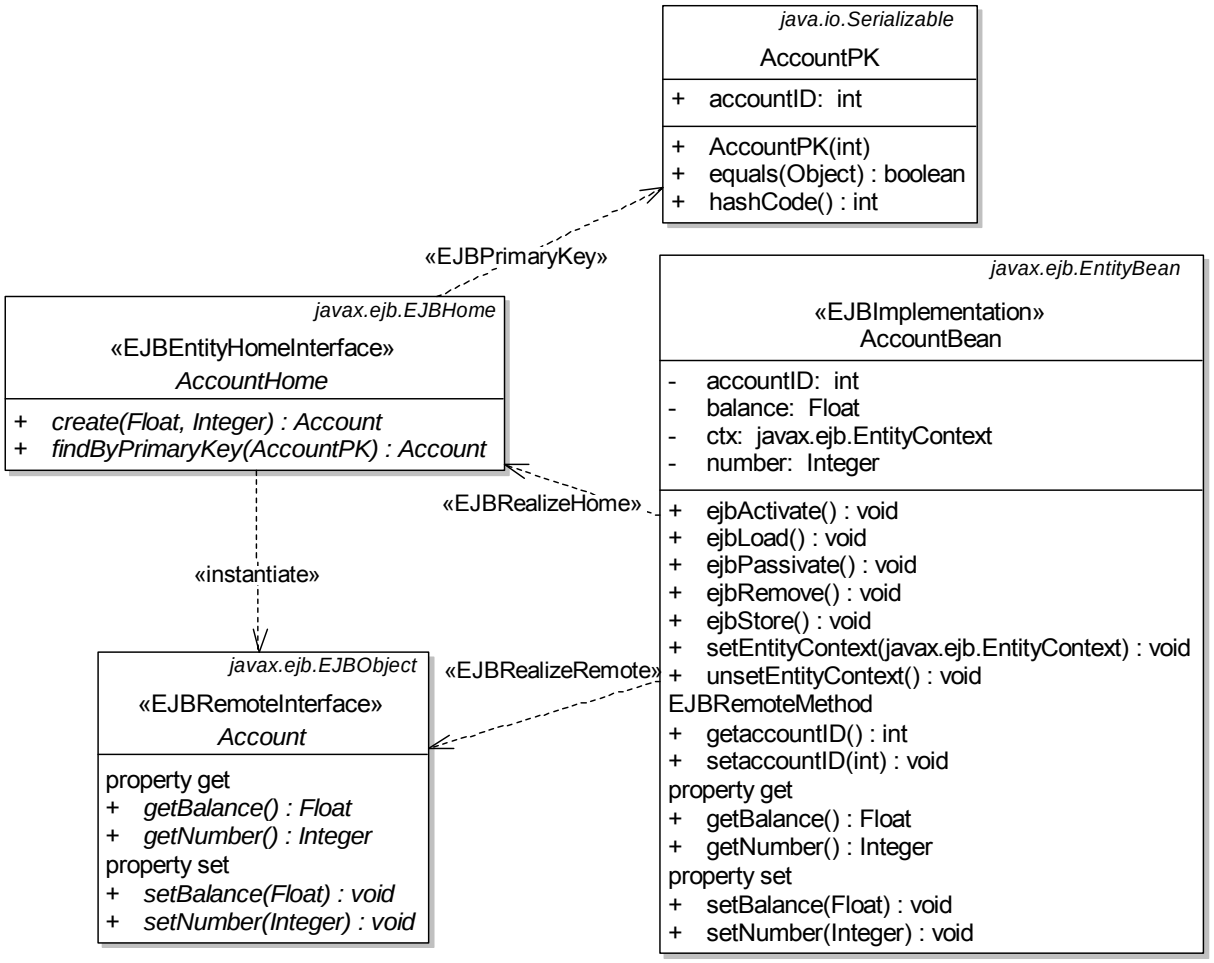

Рис. 24. Платформо-зависимая модель

Предпочтительным языком моделирования в рамках MOA является UML. Высокая степень абстракции языка UML не всегда удобна при моделировании концепций, связанных с конкретной платформой или предметной областью. Данная проблема решается путем создания расширений UML (UML extensions).

Для создания расширения UML необходимо: 1) точно определить набор концепций предметной области и их свойств, входящих в расширение; 2) установить соотношение между выделенными концепциями и элементами метамодели UML.

Расширения UML можно описать одним из следующих способов: 1) метамодель MOF; 2) профиль UML.

Организация OMG уже приняла в качестве дополнения к стандарту UML ряд профилей, предназначенных, в том числе, для спецификации вычислений в распределенных объектных средах (UML Profile for EDOC), интеграции приложений уровня предприятия (UML Profile for EAI) и т.п. Профили, как и UML-модели, можно разделить на платформо-зависимые (например, профиль для CORBA-приложений) и платформо-независимые (EDOC, EAI).

Критерий платформо-зависимости является в рамках MOA ключевым для построения иерархии моделей системы. В пределах каждого уровня (бизнес-модели, платформо-независимые, платформо-зависимые модели) может существовать множество моделей, отличающихся по степени абстракции, по точке зрения на систему и т.п. Между моделями соседних уровней иерархии MOA устанавливаются отношения уточнения (refinement), в которых более платформо-зависимые модели уточняют менее платформо-зависимые.

Подобная иерархия моделей обладает следующими достоинствами:

1) для каждого заинтересованного лица строится модель, соответствующая его уровню понимания и точке зрения на систему (для члена совета директоров – бизнес-модель, для начальника отдела АСУ – платформо-независимая модель, для рядового программиста – платформо-зависимая, возможно даже в виде каркаса исходного кода);

2) обеспечивается естественная поддержка множества распространенных методологий разработки ПО.

Возможность взаимообратного преобразования PIM- и PSM-моделей является одной из ключевых особенностей MOA. Преобразование моделей в рамках MOA используется для следующих целей:

1) преобразование и уточнение UML-моделей (трансформация PIM- в PSM-модели);

2) исследование и оценка UML-моделей при помощи других методик моделирования (например, построение по UML-модели динамической модели для исследования поведения проектируемой системы);

3) генерация исходного кода.

Преобразование моделей базируется на концепции отображения, под которым понимается набор правил и методов модификации одной модели

для получения другой модели. При отображении результирующая модель может быть выражена на языке моделирования, отличном от языка исходной модели (например, получение по UML-модели описания интерфейса в виде IDL-файла). Отображения являются основными инструментами разработки с использованием подхода MOA.

В описании отображения определяются элементы языка моделирования исходной модели и соответствующие им элементы языка моделирования результирующей модели, а также условия, которым должна отвечать исходная модель для выполнения преобразования с использованием описываемого отображения.

Формализация преобразования моделей, фиксируемая в отображении, позволяет осуществлять его как вручную, так и при помощи специальных средств преобразования моделей. Автоматизация рутинных операций преобразования моделей позволяет повысить эффективность разработки ПО, что составляет одно из преимуществ модельно-ориентированного процесса разработки.

Модельно-ориентированный процесс разработки ПО, неформально определяемый MOA, представляет собой итерационное создание и уточнение системы моделей, описывающих некоторую ИС с различных точек зрения и организованных в виде иерархии по степени платформо-зависимости. В рамках каждой итерации уточнение моделей выполняется, как правило, в направлении от наиболее абстрактных (бизнес-моделей) к наиболее платформо-зависимым, частным случаем которых является исходный код на некотором языке программирования.

Следование принципам MOA обеспечивает повышение уровня абстракции при разработке ИС, что приводит к увеличению сохранности инвестиций в ИТ-проекты. Наиболее важные элементы «ноу-хау» ИС организации, составляющие ее основную ценность (как то маршруты прохождения информации, алгоритмы обработки информации, методы поддержки принятия решений и т.п.), фиксируются в бизнес-моделях и платформо-независимых моделях. Это позволяет в дальнейшем перейти на другую техническую платформу с сохранением предыдущих наработок за счет преобразования соответствующих моделей в платформо-зависимые модели, отражающие особенности наиболее передовой технологии.

ПРИЛОЖЕНИЕ

Практическое занятие 1: УПРАВЛЕНИЕ ТРЕБОВАНИЯМИ И ПОСТРОЕНИЕ ДИАГРАММЫ ПРЕЦЕДЕНТОВ

1. Цель

Изучить процесс управления требованиями к разрабатываемой системе, представление требований в виде прецедентов, освоить построение диаграмм прецедентов и написание описаний прецедентов.

Задание: в соответствии с выбранной предметной областью выделить и описать требования к проектируемой системе в виде модели прецедентов.

2. Порядок выполнения

1. Согласовать с преподавателем индивидуальную предметную область для проектирования информационной системы.

2. Изучить теоретические сведения по технологическому процессу управления требованиями в технологии RUP и разработке диаграмм прецедентов в языке UML.

3. Определить пользователей проектируемой системы и других заинтересованных лиц, составить список их требований к функциональности проектируемой системы, выделить из описания требований потенциальных актеров и прецеденты.

4. Построить диаграмму прецедентов. Рекомендуется использовать отношения включения и расширения для определения структуры прецедентов разрабатываемой системы.

5. Составить краткие описания прецедентов. Согласовать с преподавателем два прецедента, для которых должен быть составлен документ описания прецедентов.

6. Для выбранных прецедентов определить предусловия, основной и альтернативные потоки событий и постусловия. Должно быть описано не менее одного альтернативного потока событий, связанного с некоторой исключительной ситуацией.

3. Содержание отчета

1. Цель работы.
2. Краткое описание разрабатываемой системы (в соответствии с выданным индивидуальным заданием).
3. Список выделенных функциональных требований.
4. Таблица распределения требований по актерам и прецедентам.
5. Диаграмма прецедентов.
6. Описание двух прецедентов, согласованных с преподавателем.

4. Контрольные вопросы

1. Перечислите основные элементы процесса управления требованиями в технологии RUP.
2. Назовите и приведите условные обозначения основных элементов диаграмм прецедентов.
3. Перечислите виды отношений между элементами диаграммы прецедентов и приведите примеры их использования.
4. Объясните назначение разделов документа описания прецедентов.
5. Поясните взаимосвязь описания прецедентов и проектирования интерфейса пользователя.

Практическое занятие 2: ПРОЕКТИРОВАНИЕ СТРУКТУРЫ СИСТЕМЫ В ВИДЕ ДИАГРАММЫ КЛАССОВ

1. Цель

Изучить процесс анализа и проектирования в части определения потенциальной архитектуры системы, освоить построение диаграммы классов.

Задание: спроектировать структуру информационной системы в соответствии с выбранной предметной областью в виде диаграммы классов.

2. Порядок выполнения

1. Изучить теоретические сведения по технологическому процессу анализа и проектирования в технологии RUP и разработке диаграмм классов в языке UML.

2. Проанализировать требования к функциональности системы, а также описания прецедентов, построить список кандидатов в классы и исходную диаграмму классов.

3. Выделить группы сходных по назначению классов, дополнить диаграмму классов соответствующими абстрактными классами, добавить в диаграмму отношения обобщения (или специализации) между абстрактными и конкретными классами.

4. Выявить отношения ассоциации между классами, определить их кратность, направленность и характер включения связанных объектов (другими словами, выделить среди ассоциаций отношения агрегации и композиции).

5. Добавить в диаграмму классов атрибуты и операции для всех включенных в нее классов.

3. Содержание отчета

1. Цель работы.
2. Краткое описание разрабатываемой системы (в соответствии с выданным индивидуальным заданием).
3. Диаграмма классов с краткими пояснениями.

4. Контрольные вопросы

1. Перечислите основные элементы процесса анализа и проектирования в технологии RUP.
2. Назовите виды структурных диаграмм в языке UML.
3. Дайте определение класса и его свойств (атрибутов и операций).
4. Дайте определение отношения обобщения между классами.
5. Дайте определение отношения ассоциации между классами и перечислите его основные характеристики.

Практическое занятие 3: ПРОЕКТИРОВАНИЕ РЕАЛИЗАЦИИ ФУНКЦИЙ СИСТЕМЫ С ПОМОЩЬЮ ДИАГРАММ ПОВЕДЕНИЯ

1. Цель

Изучить методы динамического моделирования систем в языке UML, освоить построение диаграммы видов деятельности, обзорной диаграммы взаимодействия и диаграммы последовательностей.

Задание: уточнить архитектуру проектируемой системы путем моделирования поведения информационной системы в соответствии с выбранной предметной областью с помощью диаграммы видов деятельности, обзорной диаграммы взаимодействия и диаграммы последовательностей.

2. Порядок выполнения

1. Изучить теоретические сведения по разработке диаграмм поведения в языке UML.

2. Построить диаграммы видов деятельности (не менее двух) для прецедентов разрабатываемой системы, в которых должны быть использованы ветвления и параллельные потоки управления.

3. Построить диаграммы последовательностей (не менее двух) для отдельных действий одной из диаграмм видов деятельности. В данных диаграммах необходимо использовать не менее двух комбинированных фрагментов с различными операторами.

4. Построить обзорную диаграмму взаимодействия на основе диаграммы видов деятельности, разработанной в п. 2. Обзорная диаграмма взаимодействия должна содержать ссылки на построенные диаграммы последовательностей, а остальные действия должны быть представлены фрагментами взаимодействия непосредственно на диаграмме.

5. Кратко описать построенные диаграммы.

3. Содержание отчета

1. Цель работы.
2. Краткое описание разрабатываемой системы (в соответствии с выданным индивидуальным заданием).
3. Построенные диаграммы поведения с краткими пояснениями.

4. Контрольные вопросы

1. Перечислите виды диаграмм поведения в языке UML, поясните их основные отличия.
2. Дайте определение действия и деятельности, назовите их основные отличия.
3. Назовите и приведите условные обозначения основных элементов диаграмм последовательностей.
4. Поясните связь диаграмм поведения с моделью прецедентов.

Практическое занятие 4: ПРОЕКТИРОВАНИЕ ДИАГРАММ КОМПОНЕНТОВ И РАЗВЕРТЫВАНИЯ. ПЕРЕХОД ОТ ПРОЕКТА К РЕАЛИЗАЦИИ В РАМКАХ КОНЦЕПЦИИ МОДЕЛЬНО-ОРИЕНТИРОВАННОЙ АРХИТЕКТУРЫ

1. Цель

Изучить основные положения концепции модельно-ориентированной архитектуры (MOA) и диаграммы, присущие модели реализации; освоить процедуру перехода от проектной модели к модели реализации в рамках данной концепции.

Задание: реализовать прототип проектируемой системы путем перехода к платформо-зависимым моделям с последующей автоматической генерацией исходного кода системы.

2. Порядок выполнения

1. Изучить теоретические сведения по разработке систем на базе принципов модельно-ориентированной архитектуры. Ознакомиться с базовыми возможностями используемых инструментальных средств. Основным из них является iO-Software ArcStyler Tool Suite версии 3.1 или выше (далее – ArcStyler) – набор средств, обеспечивающих разработку ПО в соответствии с принципами MOA. В зависимости от используемой версии ArcStyler для построения UML-диаграмм используется Rational Rose 2000 либо MagicDraw UML 9.

2. Создать платформо-независимую модель. Для построения PIM-модели следует воспользоваться каркасом ArcStyler MDA. При этом в логическое представление модели добавляется пакет iOCATBase, содержа-

щий платформо-независимые примитивные типы, которые следует использовать в PIM-моделях, а также основные пакеты и классы для поддерживаемых платформ. Все используемые в модели типы полей, типы аргументов и возвращаемых значений методов должны браться из пакета «Logical View::iOCATBase::elementary_datatypes».

3. Создать новый ArcStyler-проект и включить в него построенную ранее PIM-модель.

4. Настроить ArcStyler-проект, то есть определить используемые картриджи и пути для генерации исходного кода.

5. Аннотировать платформо-независимую модель. Под аннотированием понимается задание стереотипов и установка помеченных значений (tagged values) для классов, атрибутов и методов UML-модели.

Например, стереотип класса ComponentSegment обозначает, что из данного класса будет сгенерирован компонент выбранной компонентной архитектуры (например EJB), который при этом будет входить в состав компонента в терминологии UML.

6. Определить платформо-зависимые элементы модели, то есть компоненты (в терминологии UML), которые определяют набор модулей приложения и их состав при реализации приложения на конкретной платформе. Компонент со стереотипом EJBArchive представляет собой архив EJB-компонентов. Состав архива определяется на закладке Realizes свойств компонента.

Созданные компоненты следует разместить на диаграмме компонентов. Кроме того, необходимо создать диаграмму развертывания, на которой следует привести размещение компонентов по отдельным узлам вычислительной сети.

7. Сгенерировать каркас приложения. ArcStyler добавляет в контекстное меню обозревателя модели Rational Rose свои пункты, объединенные в подменю «ArcStyler». Для генерации каркаса приложения нужно выполнить операцию Generate для логического представления UML-модели или для отдельного пакета. Обратите внимание, что перед генерацией автоматически выполняется проверка модели на соответствие требованиям картриджа (то есть специфической платформы).

8. Собрать, развернуть и протестировать приложение. Для каждого компонента создается файл build.xml, содержащий инструкции по сборке, развертыванию и тестированию, а также обеспечивающий запуск Ant ко-

мандный файл build.bat. При запуске build.bat в качестве параметра следует указывать название конкретной цели:

1) собрать компонент с помощью цели build;

2) запустить сервер приложений JBoss с помощью цели startServer. При этом JBoss обнаружит EAR-файл приложения и развернет его;

3) установить Web-сервис:
   - запустить сервер Tomcat (цель сборки – startSOAPServer);
   - развернуть Web-сервис путем задания цели deploysoap;
   - перезапустить сервер Tomcat с использованием комбинированной цели «stopSOAPServer startSOAPServer»;

4) для тестирования компонентов с использованием тестовой оболочки JUnit используется цель runAssemblyClient. Если тесты реализованы без использования JUnit, то выполнить их можно с помощью цели runClient;

5) последовательность остановки серверов выглядит следующим образом. Сначала останавливается сервер Tomcat, для чего служит цель сборки stopSOAPServer. Затем производится останов сервера приложений JBoss с использованием цели stopServer.

3. Содержание отчета

1. Цель работы.
2. Диаграмма классов платформо-независимой модели.
3. Перечень присвоенных стереотипов и таблицы установленных помеченных значений.
4. Диаграммы классов, компонентов и развертывания платформозависимой модели.
5. Журнал сборки, развертывания и тестирования системы.

4. Контрольные вопросы

1. Дайте определения модели и платформы в MOA и поясните их взаимосвязь.
2. Дайте определение отображения в MOA, приведите способы определения и выполнения отображений.
3. Назовите элементы диаграмм компонентов и развертывания.
4. Перечислите этапы перехода от проектной модели к модели реализации.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Александров, Д. В.* Системное моделирование бизнеса : учеб. пособие / Д. В. Александров ; Владим. гос. ун-т. – Владимир : Изд-во Владим. гос. ун-та, 2004. – 300 с. – ISBN 5-89368-501-6.
2. *Александров, Д. В.* Распределенные информационные системы. CASE-технологии реинжиниринга : учеб. пособие / Д. В. Александров, А. В. Костров ; Владим. гос. ун-т. – Владимир : Изд-во Владим. гос. ун-та, 2001. – 136 с. – ISBN 5-89368-241-6.
3. *Вендров, А. М.* CASE-технологии. Современные методы и средства проектирования информационных систем / А. М. Вендров. – М. : Финансы и статистика, 1998. – 176 с. – ISBN 5-279-02144-X.
4. *Ивлев, В. А.* Реорганизация деятельности предприятий : от структурной к процессной организации / В. А. Ивлев, Т. В. Попова. – М. : Научтехлитиздат, 2001. – 282 с. – ISBN 5-89638-040-2.
5. *Калянов, Г. Н.* Консалтинг при автоматизации предприятий / Г. Н. Калянов. – М. : СИНТЕГ, 1997. – 316 с. – ISBN 5-89638-002-X.
6. *Костров, А. В.* Уроки информационного менеджмента. Практикум : учеб. пособие / А. В. Костров, Д. В. Александров. – М. : Финансы и статистика, 2005. – 304 с. – ISBN 5-279-02573-9.
7. *Маклаков, С. В.* Bpwin и Erwin. CASE-средства разработки информационных систем : учебно-справ. издание / С. В. Маклаков. – М. : Диалог-МИФИ, 1999. – 256 с. – ISBN 5-86404-128-9.
8. *Черемных, С. В.* Структурный анализ систем : IDEF-технологии / С. В. Черемных, И. О. Семенов, В. С. Ручкин. – М. : Финансы и статистика, 2001. – 208 с. – ISBN 5-279-02433-3.
9. *Буч, Г.* Унифицированный процесс разработки программного обеспечения / Г. Буч, Дж. Рамбо, А. Якобсон – СПб. : Питер, 2002. – 496 с. – ISBN 5-318-00358-3.
10. *Крачтен, Ф.* Введение в Rational Unified Process / Ф. Крачтен ; пер. с англ. – 2-е изд. – М. : Вильямс, 2002. – 239 с. – ISBN 5-8459-0239-8.
11. *Мацяшек, Л. А.* Анализ требований и проектирование систем. Разработка информационных систем с использованием UML / Л. А. Мацяшек. – М. : Вильямс, 2002. – 432 с. – ISBN 5-8459-0276-2.

Учебное издание

АЛЕКСАНДРОВ Дмитрий Владимирович
ГРАЧЕВ Иван Викторович
ФАДИН Дмитрий Николаевич

CASE-ТЕХНОЛОГИИ

Учебное пособие

Редактор Е. В. Афанасьева
Технический редактор Н. В. Тупицына
Корректор Т. В. Климова
Компьютерный набор И. В. Грачев, Д. Н. Фадин
Компьютерная верстка Д. В. Александров, И. В. Грачев
Дизайн обложки М. А. Шакеров

Подписано в печать . .06.
Формат 60x84/16. Бумага для множит. техники. Гарнитура Таймс.
Печать на ризографе. Усл. печ. л. 3,72. Уч.-изд. л. 3,81. Тираж 100 экз.
Заказ
Издательство
Владимирского государственного университета.
600000, Владимир, ул. Горького, 87.