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

Факультет информационных технологий и управления

Кафедра систем управления

С. И. Городко, С. В. Снисаренко

СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для студентов специальности 1-53 01 07 «Информационные технологии и управление в технических системах»

Минск БГУИР 2017

УДК 004.424(076)
ББК 32.973.26-018.2я73
Г70

Р е ц е н з е н т ы:

кафедра информационных систем и технологий Белорусского национального технического университета (протокол №1 от 06.09.2015);

начальник кафедры радиолокации и приемопередающих устройств учреждения образования «Военная академия Республики Беларусь», кандидат технических наук, доцент В. А. Кондратёнок

Городко, С. И.

Г70 Современные технологии программирования : учеб.-метод. пособие / С. И. Городко, С. В. Снисаренко. – Минск : БГУИР, 2017. – 67 с. : ил.

ISBN 978-985-543-252-5.

Приведены сведения, методика анализа программных систем на основе построения визуальных моделей в рамках унифицированного процесса разработки программного обеспечения с использованием унифицированного языка моделирования UML. Даны примеры и конкретные рекомендации, помогающие лучше усвоить соответствующий материал и выполнить задание по контрольной работе.

УДК 004.424(076)
ББК 32.973.26-018.2я73

ISBN 978-985-543-252-5

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ 4
1 ТЕХНОЛОГИИ WEB-ПРОГРАММИРОВАНИЯ 6
И КОНЦЕПЦИЯ HTML 6
1.1 Серверные web-приложения 6
1.2 Клиентские приложения 8
1.3 Отображение страницы в окне браузера 9
1.4 Графика в web 11
1.5 Текст в web. Два комплекта шрифтов 12
1.6 Структура HTML-страницы 12
1.7 Раздел заголовка 14
1.8 Раздел тела документа 15
1.9 Управление отображением текста 15
1.10 Таблицы 16
1.11 Гиперссылки 17
1.12 Списки стилей 18
2 ЯЗЫК UML 21
2.1 Структура и компоненты языка UML 22
2.2 Диаграммы вариантов использования (use case diagram) 29
2.3 Диаграммы последовательности (sequence diagram) 34
2.4 Диаграммы кооперации (collaboration diagram) 39
2.5 Диаграммы классов (class diagram) 43
2.6 Диаграммы состояний (statechart diagram) 51
2.7 Диаграммы деятельности (activity diagram) 54
2.8 Диаграммы компонентов (component diagram) 58
2.9 Диаграммы развертывания (deployment diagram) 61
ЗАКЛЮЧЕНИЕ 65
СПИСОК ЛИТЕРАТУРЫ 66

ПРЕДИСЛОВИЕ

Под современными технологиями программирования сегодня понимают в основном интернет-технологии, включающие в себя концептуальные знания www и HTML, Java, клиентских и серверных скриптов и языков запросов к базам данных, основы web-дизайна. Однако наиболее важной частью профессиональной подготовки специалиста является умение работать над большим проектом, быть в «команде» и доводить проект от замысла до реализации.

В силу специфичности производства ПО (практически нулевая стоимость тиражирования, очень быстрый процесс устаревания и т. д.) технология его создания очень сильно завязана на человеческий ресурс и поэтому должна включать в себя организационный и управленческий аспекты. На сегодняшний день в мире существует огромное количество различных процессов для создания ПО. Тем не менее именно технологий, рассматривающих полный жизненный цикл проекта разработки ПО, сочетающих в себе научный подход, серьезную базу исследований и имеющих историю реального использования и адаптации, относительно немного.

Нас интересует разработка того, что мы будем называть промышленными программными продуктами. Они применяются для решения самых разных задач, таких, например, как системы с обратной связью, которые управляют или сами управляются событиями физического мира и для которых ресурсы времени и памяти ограничены; задачи поддержания целостности информации объемом в сотни тысяч записей при параллельном доступе к ней с обновлениями и запросами; системы управления и контроля за реальными процессами (например, диспетчеризация воздушного или железнодорожного транспорта). Системы подобного типа обычно имеют большое время жизни, и большое количество пользователей оказывается в зависимости от их нормального функционирования. В мире промышленных программ мы также встречаем среды разработки, которые упрощают создание приложений в конкретных областях, и программы, которые имитируют определенные стороны человеческого интеллекта.

Существенная черта промышленной программы – уровень сложности: один разработчик практически не в состоянии охватить все аспекты такой системы. Грубо говоря, сложность промышленных программ превышает возможности человеческого интеллекта. Увы, но сложность, о которой мы говорим, по-видимому, присуща всем большим программным системам. Говоря «присуща», мы имеем в виду, что эта сложность здесь неизбежна: с ней можно справиться, но избавиться от нее нельзя.

Цель данного учебно-методического пособия – обучение методике анализа программных систем на основе построения визуальных моделей в рамках унифицированного процесса разработки ПО (The Unified Software Development Process) [2] и с использованием унифицированного языка моделирования UML (The Unified Modeling Language) [4].

В последние годы прошлого века появился и очень быстро завоевал огромную популярность новый класс приложений – так называемые web-

приложения. Обеспечивающие доступ через Интернет или интрасеть к информационным системам и базам данных web-приложения стали одним из наиболее эффективных инструментов современного бизнеса.

Для разработки web-серверов, являющихся web-приложениями, широко используется язык разметки гипертекста HTML (HyperText Markup Language). Фактически все страницы, которые видят посетители web-сервера, составлены на языке HTML и содержат объекты различных типов (изображения, анимацию, формы для ввода информации и т. д.). Если web-сервер содержит только статическую информацию, изменяющуюся эпизодически, ее можно представить в виде набора документов HTML. Для их создания подходит практически любой текстовый редактор (даже простейший Notepad), хотя лучше воспользоваться специальными средствами визуального проектирования страниц HTML, такими как Microsoft FrontPage.

Сам по себе язык HTML несложен, однако эта простота обманчива. В силу ограниченности его возможностей и ряда других обстоятельств приходится немало потрудиться, чтобы получить желаемый результат. Одна и та же страница может по-разному отображаться в различных браузерах, поэтому при проектировании web-страниц вопросам совместимости с браузерами приходится уделять особое внимание. Чтобы ускорить загрузку страниц, необходимо минимизировать общий объем расположенных на них иллюстраций.

Существует достаточное количество программных оболочек, в которых реализована графическая нотация и стереотипы языка UML. Наиболее популярными из них являются Rational Rose (разработчик Rational Software), в которой используется версия 1.5 языка UML и 5 Enterprise Architect (от Sparx Systems), где уже используется версия 2.0 языка UML. Базовая нотация языка UML имеется также в графическом редакторе Visio.

1 ТЕХНОЛОГИИ WEB-ПРОГРАММИРОВАНИЯ И КОНЦЕПЦИЯ HTML

1.1 Серверные web-приложения

Различают пассивные и активные серверы web. Если страницы сервера содержат только статическую текстовую и мультимедийную информацию, а также гипертекстовые ссылки на другие страницы, то сервер называется *пассивным*. Когда же страницы сервера ведут себя аналогично окнам обычных интерактивных приложений, вступая в диалог с пользователем, пользователь имеет дело с *активным* сервером. Очевидно, статический сервер web не может служить основой для создания интерактивных приложений в сети Интернет с базами данных, так как он не предусматривает никаких средств ввода и обработки запросов.

Программы CGI. Для того чтобы сервер web мог вести диалог с пользователем, разработан механизм программных расширений сервера, основанный на применении так называемого стандартного шлюзового интерфейса (*Common Gateway Interface, CGI*). Программы CGI пользуются этим интерфейсом для получения сведений от пользователя, для их обработки и отправки обратно в виде нового документа HTML, ссылки на существующий документ или на другой объект.

При этом для ввода информации пользователем в документ HTML встраиваются формы, содержащие различные органы управления. Заполнив всю форму, пользователь нажимает кнопку ввода, и данные из полей формы передаются программе CGI (рисунок 1.1). Обработав данные, программа CGI динамически формирует новый документ HTML с результатами обработки и отправляет его обратно пользователю. При необходимости программа CGI обращается к СУБД или другим программным системам, работающим на сервере.

Программы CGI можно составлять на различных языках программирования – C, C++, Perl, Pascal, Java и т. д. Perl особенно удобен для создания программ CGI, так как он содержит соответствующие функции и доступен в различных операционных системах, в том числе Linux и Solaris.

Программа CGI – это консольное приложение, работающее в среде операционной системы сервера web и осуществляющее обмен данными через стандартные потоки ввода и вывода. Такое приложение запускается только по запросу пользователя, когда к нему выполняется обращение из документа HTML. Окончив обработку запроса пользователя, программа CGI завершается.

Расширения ISAPI. Другая технология расширения сервера web – программный интерфейс сервера Microsoft IIS – ISAPI *(Internet information Server Application Program Interface)*. По своим функциональным возможностям модули ISAPI аналогичны программам CGI, однако они работают быстрее за счет того, что приложение не завершается после обработки данных, а постоянно работает в виде процесса. Для CGI программ для каждого пользователя приходится запускать отдельный процесс, что занимает время, а приложение ISAPI обрабатывает запросы от всех пользователей. С другой стороны, так как ISAPI работает в адресном пространстве сервера web, ошибка в приложении ISAPI

способна вызвать аварийное завершение работы сервера web. Ошибки в программе CGI менее значимы, так как авария произойдет в том процессе, в котором работает эта программа.

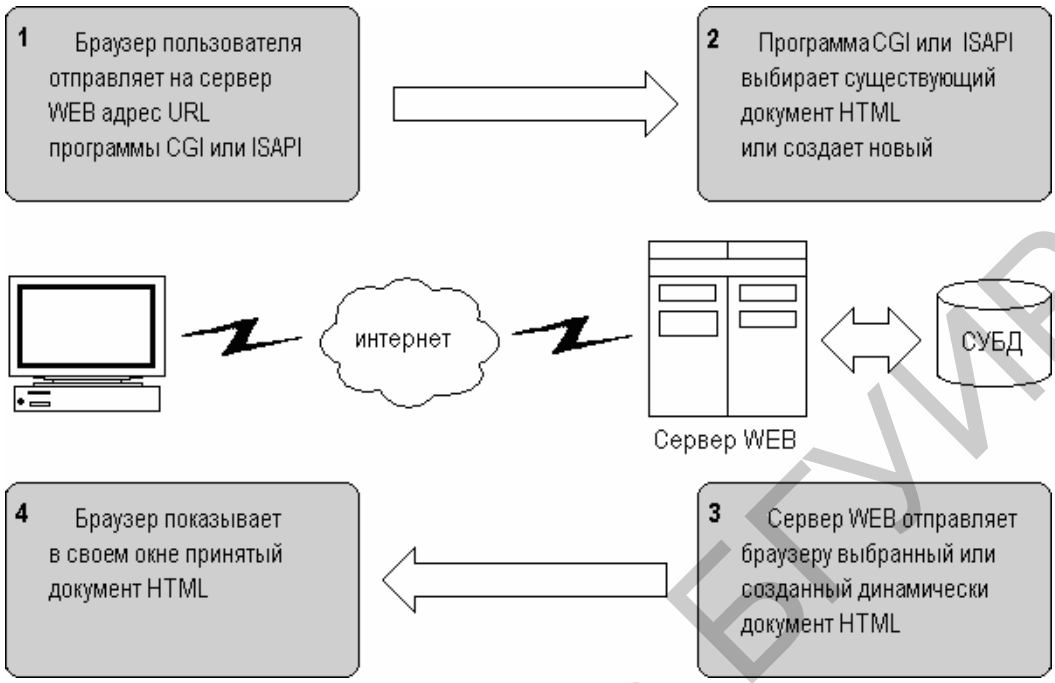


Рисунок 1.1 – Взаимодействие клиентского браузера и программного расширения

Хотя технология ISAPI изначально предназначалась только для сервера Microsoft IIS, сейчас ее можно использовать и на платформе Linux. Для создания расширения ISAPI используются языки C и C++, а также функции программного интерфейса Windows.

Приложения ASP. Технология *Active Server Pages* (ASP) предполагает использование на сервере Internet Information Server текстовых файлов с расширением asp, содержащих операторы языка HTML, и сценарии на JScript или VB Script. Когда пользователь обращается к странице ASP, сервер web интерпретирует расположенный в ней сценарий. При этом анализируются параметры, переданные этой странице. Далее страница модифицируется (или создается заново), а затем отправляется обратно пользователю. Сервер web отправляет не саму страницу, а результат ее интерпретации, а логика работы страницы скрыта от пользователей.

Приложения PHP. Еще один способ создания активных серверов web – использование технологии предварительной обработки гипертекста PHP (сокращение от «Php: *Hypertext Preprocessor*»). В то время как ASP предполагает активное использование модели компонентного объекта COM и элементов управления ActiveX, технология PHP базируются на классических библиотеках объектных модулей. Разработанная для платформы Unix и ее клонов PHP сегодня доступна и на платформе Microsoft Windows.

1.2 Клиентские приложения

Целесообразно разделять работу между клиентом и сервером, чтобы добиться оптимальной производительности в условиях низкоскоростных каналов Интернета. Предварительную обработку введенных данных, отправляемых серверу, имеет смысл выполнять на стороне клиента. Это позволит исключить повторные передачи неправильно заполненных форм. А вот выборку из базы данных должен выполнять сервер.

Браузер, отображающий содержимое страниц, играет роль «интеллектуального» терминала. Помимо показа текста и графических изображений, браузер представляет собой среду, в которой работают активные объекты, встроенные в страницы web. Это сценарии JavaScript, VB Script, аплеты Java, элементы управления ActiveX и некоторые другие.

Клиентские сценарии JavaScript. Язык сценариев JavaScript разработан фирмой Netscape Communication Corporation и первоначально назывался LiveScript. Язык JavaScript не имеет никакого отношения к языку Java, созданному Sun Microsystems.

Конструкции языка JavaScript встраиваются в страницы HTML и интерпретируются под управлением браузера при загрузке страниц, а также при совершении пользователем определенных действий над объектами, расположенными в этих страницах. Сценарии JavaScript способны обрабатывать данные, введенные пользователями в полях форм, а также события, возникающие в процессе манипуляций пользователя с мышью, копировать в окно браузера другие страницы HTML или изменять содержимое уже загруженных страниц.

Сценарии JavaScript широко применяются для создания различных визуальных эффектов, таких, например, как изменение внешнего вида элементов управления, над которыми установлен курсор мыши, анимация графических изображений, создание звуковых эффектов и т. д.

Механизм локальной памяти Cookie позволяет сценариям JavaScript сохранять на компьютере локальную информацию, введенную пользователем. Например, в Cookie может храниться список товаров из интернет-магазина, отобранных для покупки.

Для обеспечения совместимости с различными браузерами приходится учитывать такие особенности, что например, браузер IE реализует собственную версию JavaScript, называемую JScript.

Клиентские сценарии VB Script. Помимо JScript, браузер IE способен работать с VB Script, являющимся подмножеством Visual Basic и функционально равноценным языку JavaScript. Так как не все в Интернете работают с IE, применение VB Script для создания страниц, расположенных в Интернете, неоправдано. Ситуация меняется, если эта технология применяется в корпоративной интрасети. Когда администратор может установить на компьютеры всех пользователей IE, а в штате компании есть программисты, имеющие большой опыт работы с Visual Basic, то применение VB Script вместо JavaScript сокращает сроки разработки.

Аплеты Java. Подмножество приложений Java, называемых аплетами Java, используют наряду с клиентскими сценариями для организации активности на стороне клиента. Аплеты Java могут применяться для получения визуальных и звуковых эффектов, организации ввода и предварительной обработки данных перед отправкой их на сервер, а также для представления полученных от сервера данных в графическом, табличном или ином виде.

Создание аплетов во многом упрощается благодаря наличию обширных библиотек классов Java. В этих библиотеках есть мощные и удобные средства для организации пользовательского интерфейса, работы с графическими изображениями, передачи данных по сети и др. Это позволяет больше внимания уделить логике работы аплета, а не реализации типовых задач программирования вроде организации динамических массивов или загрузки графических изображений. Аплеты выполняются под управлением браузера и не имеют доступа к ресурсам компьютера.

Элементы управления ActiveX. Элементы управления ActiveX (ActiveX control) основаны на модели компонентных объектов (Component Object Model, COM), их применяют для решения тех же задач, что и аплеты Java, однако предоставляют полный доступ к ресурсам компьютера. Это уменьшает привлекательность элементов ActiveX для оформления страниц серверов web, так как загрузка на компьютер неизвестных программ небезопасна. Элементы управления ActiveX можно использовать как на стороне сервера, так и на стороне клиента. Чтобы убедить пользователей в том, что предлагаемый для загрузки элемент ActiveX безопасен, используется технология цифровых сертификатов. Чтобы получить цифровой сертификат и подписать свой элемент управления ActiveX, разработчик должен внести единовременный денежный взнос, а затем производить ежегодную плату.

Заметим, что при разработке элементов управления ActiveX для web-сервера их можно не подписывать, так как они не загружаются клиентами, а выполняются непосредственно на сервере.

1.3 Отображение страницы в окне браузера

При создании гипертекстовых страниц необходимо учитывать многообразие браузеров и платформ, каждая из которых по-разному поддерживает HTML и сценарии. Большая часть используемых современных браузеров – это Internet Explorer. Он интегрирован в операционную систему, поэтому пользователи Windows используют его умолчанию. Альтернативой IE является Opera, которая за короткое время превратилась из небольшой и простой программы, созданной норвежской компанией Opera Software, в серьезного конкурента IE. Этот браузер обладает исключительно малым временем загрузки и минимальными требованиями к объему диска. Достоинством Opera является полное соответствие стандартам HTML.

Одним из неприятных аспектов разработки гипертекстовых страниц является зависимость их внешнего представления от конфигурации программного и аппаратного обеспечения каждого отдельного пользователя. Страница, которая правильно выглядит на одной машине, может совершенно иначе выглядеть на экране другого пользователя. Это зависит как от возможностей браузера, так и предпочтений пользователя (размер шрифта, цвета и т. д.).

При разработке страницы следует учитывать рабочее пространство в окне браузера, поскольку операционная система и сам браузер занимают на экране некоторое пространство. При проектировании страницы следует фиксировать горизонтальный размер рабочей области окна (правилом хорошего тона является отсутствие горизонтальной полосы прокрутки). На практике размеры окна браузера варьируются. Рабочее пространство в Internet Explorer 4.0 распределяется следующим образом (рисунок 1.2):

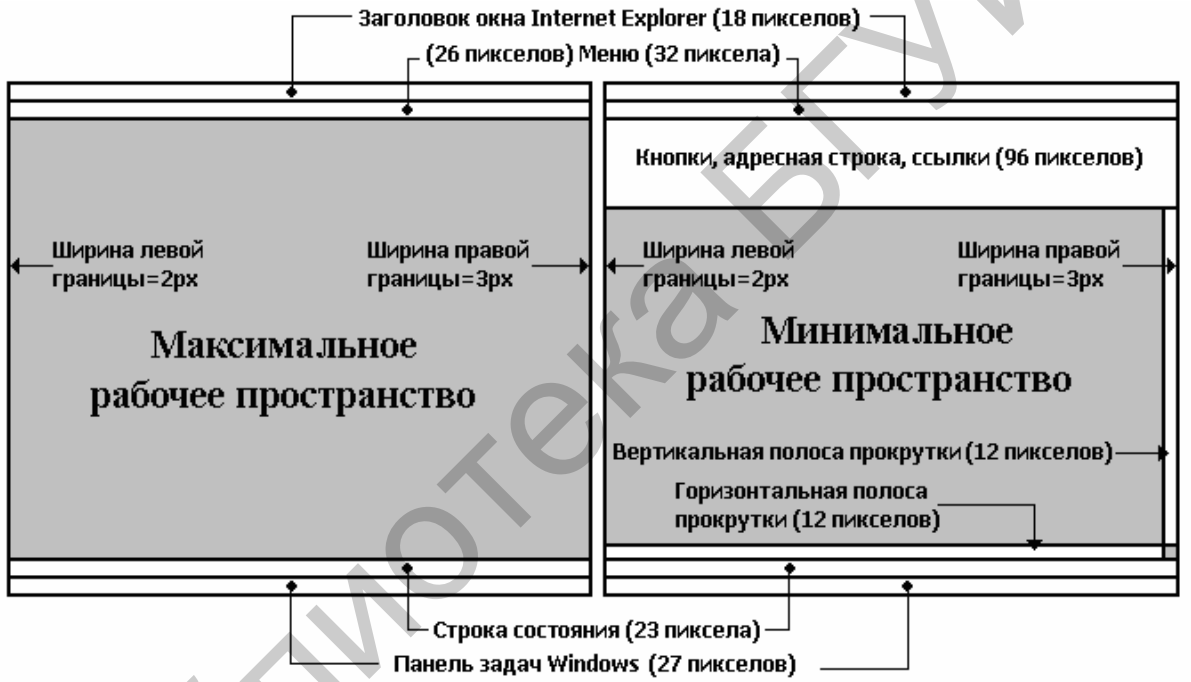


Рисунок 1.2 – Рабочее пространство в Internet Explorer

По умолчанию web-страницы *гибкие*. При этом текст и элементы HTML-файла попадают в окно браузера, заполняя все доступное пространство, вне зависимости от размеров монитора. Если размер окна браузера изменяется, элементы повторно выводятся, чтобы настроиться на новые размеры. Проблема состоит в непредсказуемости места появления элементов. Такие страницы хорошо отображаются на мониторах с разным разрешением, заполняя все пространство монитора. Однако на больших разрешениях длина строки может оказаться чрезмерной, а длинные строки неудобны для чтения с экрана. Кроме того, на больших мониторах элементы расположены гармонично, а на маленьких – они скучены.

Для структурирования гибкого документа используются таблицы и фреймы. При использовании процентных значений размеров для таблиц или фреймов размер будет изменяться в соответствии с окном браузера. К примеру, два столбца с шириной по 25 и 75 % от размера окна браузера всегда сохраняют эти пропорции независимо от разрешения.

Страница *фиксированного размера* останется постоянной независимо от размеров окна – это ее достоинство, также обеспечивается лучшее управление длинами строк. Чтобы строки не становились слишком длинными при просмотре на больших мониторах, используют таблицы. Главный недостаток, – если размер окна меньше сетки страницы, ее части не будут видны и потребуется горизонтальная прокрутка, что воспринимается как помеха. Для создания фиксированной web-страницы необходимо помесить ее содержимое в структурную таблицу с абсолютными размерами, заданными в пикселах.

1.4 Графика в web

Важным элементом при создании web-страницы является ее графическое наполнение, которое может быть передано цветовой палитрой и изображениями. Для задания цвета используются числовые RGB-значения в диапазоне от 0 до 255. Например, значения RGB для темно-оранжевого цвета равны R:198, G:83, B:52. Существуют три системы для задания цветов. Шестнадцатеричная (3333FF); проценты (процентный эквивалент 51-51-255 равен 20-20-100 %); десятичная, в диапазоне от 0 до 255 (51-51-255, что означает значение красного – 51, зеленого – 51, голубого – 255).

Большинсво изображений, существующих в web, представлены в трех растровых форматах: GIF, JPEG и PNG.

Формат GIF (Grafic Interchange Format) был первым форматом файлов, который поддерживался web-браузерами, и по сей день продолжает оставаться основным. GIF представляет собой файлы индексированных 8-разрядных цветов, что определяет максимум 256 цветов. Поскольку этот формат сжимает информацию о цветах по строкам пикселов, GIF-файлы лучше всего подходят для графики, которая содержит области равномерного цвета. Другим достоинством этого формата является возможность анимации графических изображений, состоящих из нескольких сменяющих друг друга кадров.

Формат JPEG. Вторым наиболее популярным графическим форматом в web является JPEG (Joint Photographic Experts Group). Он содержит 24-разрядную информацию о цвете. Это миллионы цветов в отличие от 256 цветов формата GIF. В JPEG используется так называемое сжатие с потерями. Это означает, что часть информации об изображении в процессе сжатия теряется, но в большинстве случаев ухудшение качества не заметно. В этом формате лучше всего сохранять фотографии или любые изображения с плавными градациями цветов, так как он предлагает более высокое качество изображения в файле меньшего объема. Однако JPEG не является лучшим решением для графических изображений с одноцветными областями, поскольку этот формат имеет тенденцию

изменять цвета, и конечный файл будет несколько больше, чем GIF-файл для того же изображения.

Формат PNG. Третий графический формат – это PNG (Portable Network Graphic), который постепенно становится очень популярным в Web. PNG может поддерживать 8-разрядные индексированные цвета, 16-разрядные полутона или 24-разрядные полноцветные изображения, используя схему сжатия без потерь. Это обеспечивает более высокое качество изображений, а иногда и меньший объем файлов по сравнению с форматом GIF за счет использования различных алгоритмов сжатия (Selective (Селективный), Adaptive (Адаптивный) и др.) Кроме того, файлы PNG имеют функции, позволяющие показывать рисунок фона сквозь отбрасываемые мягкие тени.

1.5 Текст в web. Два комплекта шрифтов

При создании web-страницы заранее неизвестно, как именно будет выглядеть текст на экране, поэтому используются два комплекта шрифтов:

– *пропорциональный шрифт* (иначе «шрифт переменной ширины») для каждого символа выделяет разное количество места в зависимости от его начертания. Например, в пропорциональном шрифте заглавная «W» занимает больше места в строке по горизонтали, чем прописная «I», Times, Helvetica и Arial являются примерами пропорциональных шрифтов. Большие отрывки текста удобнее читать, когда они напечатаны пропорциональными шрифтами. С большой вероятностью текст на странице будет отображен пропорциональным шрифтом Times;

– *шрифт с фиксированной шириной* (также известный как шрифт «постоянная ширина», или «равноширинный») предоставляет одинаковое место для всех символов шрифта. Заглавная «W» занимает столько же места, что и прописная «I». Примерами шрифтов фиксированной ширины являются Courier и Monaco.

Другим надежным способом представления текстовой информации является его оформление *в виде графического изображения*. При этом страница состоит из множества графических элементов, содержащих заголовки, подзаголовки и объявления. Многие web-страницы представлены исключительно в графике, которая содержит внутри себя весь текст страницы. При этом страница одинакова при выводе во всех графических браузерах. Однако следует учитывать, что при этом изображения загружаются дольше, чем текст, а в неграфических браузерах содержание утрачивается.

1.6 Структура HTML-страницы

По современным представлениям электронный документ – это некоторая информационная сущность, у которой можно выделить четыре аспекта: содержание, структуру, стиль, поведение. Содержание определяет информационное наполнение документа, его ценность как источника информации. Структура

определяет элементы содержания (абзац, список, таблица, раздел, картинка, объект) и связи между ними (предок-потомок, целое-часть). Стиль задает внешнее оформление документа (цвет, гарнитуру и размер шрифта, графические эффекты, выравнивание абзацев). Поведение определяет реакцию документа на события, инициируемые пользователем (нажатие клавиш клавиатуры и мыши, наведение мыши, перемещение фокуса ввода) и программой просмотра (начало или окончание загрузки документа, переход к другому документу).

Так, структура и содержание документа описываются средствами HTML. Стиль документа описывается средствами языка CSS, а поведение – средствами скриптов, фрагментов кода (например JavaScript). Использование CSS позволяет облегчить сопровождение документа, сделав его менее громоздким и более структурированным.

Структура HTML-документа описывается с помощью тегов, имеющих имя, которыми они идентифицируются. Тег (tag) – это элемент разметки, который представляет собой текст, заключенный в угловые скобки < >. Теги управляют отображением информации, но при этом сами не выводятся на экран. Теги бывают одиночными, открывающими и закрывающими:

```
<имя_элемента> отображаемое содержимое </имя_элемента>
```

Пара из открывающего и закрывающего тега называется *контейнером*.

Тег может описывать сложный фрагмент структуры, и для определения его параметров используются атрибуты, имеющие имя и значение, в виде строки в кавычках:

```
<имя_элемента имя_атрибута = "значение атрибута">
```

У некоторых атрибутов значение отсутствует.

Корневым элементом любого HTML-документа является контейнер HTML, в котором размещается все содержимое документа. Оно включает две обязательные части: Head (заголовок) и Body (тело), следующие в указанном порядке.

```
<HTML>
  <Head>
  </Head>
   <Body>
   </Body>
</HTML>
```

Элементы HTML делятся на три группы: заголовочные, блоковые и текстовые. *Заголовочные* располагаются в разделе заголовка, *блоковые* описывают структуру документа и содержат текст или другие блоковые либо текстовые элементы. *Текстовые* элементы содержат непосредственно текст документа и

другие текстовые элементы. Текстовые элементы не могут содержать блоковые элементы.

1.7 Раздел заголовка

Из элементов, которые могут употребляться в разделе заголовка документа, рассмотрим теги Base, BaseFont, Meta и Title.

Base. Указывает базовый URL, относительно которого будут разрешаться все относительные URL, встречающиеся в этом документе. URL указывается атрибутом Href (*hyper reference* – гиперссылка), а имя целевого фрейма, в который будут загружаться соответствующие документы, – атрибутом Target (необязательный атрибут).

```
<HTML>
   <Head>
   <Base Href = "http://msdn.microsoft.com/library/index.html">
</Head>
<Body>
   <A Href = "../others/toc.html">Click me</A>
</Body> </HTML>
```

Basefont. Указывает параметры отображения текста в случае, если они не заданы явно. Имеет следующие атрибуты: Size (размер) обязательный атрибут, Color (цвет), Face (гарнитура).

```
   <BaseFont color = "black" Font = "Arial, Tahoma, Verdana" Size
= "4">
```

Meta. Предназначен для внедрения в документ информации о нем самом (метаинформации), которая может быть использована службами поиска документов в Internet.

```
   <Meta http-equiv = "Content-Type" Content = "text/html; CharSet
= Windows-1251">
```

Этот элемент указывает, что тип документа – text/html, а его содержимое представлено в кодировке windows-1251, принятой в русскоязычных версиях ОС Windows. В результате документ после загрузки будет отображен правильно даже в нерусифицированных браузерах (если, конечно, в системе установлены соответствующие шрифты).

Title. Указывает строку, выводимую в заголовке окна браузера.

```
<Title>An introduction to HTML</Title>
```

Link. Определяет соотношение между текущим документом и другими документами. Используется для связывания документов со списками стилей:

```
<link Rel="stylesheet" Type="text/css" Href="acad.css">
```

1.8 Раздел тела документа

Тело оформляется элементом Body, который является контейнером. Для него определены следующие необязательные атрибуты: ALink (цвет гиперссылок при выборе их пользователем), BgColor (фоновый цвет), BackGround (фоновый рисунок), Text (цвет текста по умолчанию), TopMargin (верхнее поле документа в пикселах), BottomMargin (нижнее поле документа в пикселах), LeftMargin (левое поле документа в пикселах), RightMargin (правое поле документа в пикселах), link (цвет ссылок, не посещенных пользователем), Vlink (цвет ссылок, посещенных пользователем), Scroll (значение "yes" или "no" указывает, отображать полосы прокрутки или нет).

```
   <Body Alink = "#FF0000" BgColor = "#FFFFFF" Link = "#0000FF"
Text = "#000000" Vlink = "#800080">
```

1.9 Управление отображением текста

В HTML предусмотрено 6 встроенных стилей для текстовых заголовков, они оформляются элементами H1, H2, ..., H6 (сокр. от *heading* – заголовок, по аналогии с уровнями заголовков Word). Заголовки верхнего уровня по умолчанию форматируются более крупным шрифтом. Заголовок является контейнером.

Абзацы оформляются элементом P (сокр. от *paragraph*). По умолчанию абзацы выравниваются влево, для выравнивания по обоим краям можно использовать значение "justify". Закрывающий тег – необязательный, абзац завершается, когда внутри него встречается первый блоковый элемент.

```
   <H1 Align = "Center">Введение в HTML</H1>
<P Align = "Justify">HTML (Hypertext Markup Language)
```

Текст внутри заголовка или абзаца можно форматировать с помощью текстовых элементов логического и физического стиля.

К элементам **логического** стиля относятся такие, как address (оформление контактной информации), cite (оформление цитат), code (оформление фрагмента кода, вставленного в текст абзаца) и др.

К элементам **физического** стиля относятся: Font (определяет параметры шрифта: цвет, гарнитуру и размер, и др.), B (полужирный), I (курсив), U (подчеркнутый), s (перечеркнутый), Sub (нижний индекс), Sup (верхний индекс).

```
<P>Текст можно выделять <I>курсивом</I>, <B>полужирным</B></P>
<P>Вот формула полной энергии тела: E = mc<Sup>2</Sup>.</P>
```

Указанных тегов часто недостаточно для полноценного форматирования текста, поэтому многие элементы физического стиля формируются только с использованием таблицы стилей.

Необходимо отметить, что в тексте нельзя употреблять некоторые символы, которые используются для оформления элементов HTML. Например, символы < и >. Их следует заменять соответствующими *esc*-последовательностями. Также *esc*-последовательностями обозначаются символы copyright и registered trade mark, специальные символы некоторых европейских алфавитов и др.

1.10 Таблицы

Таблица – наиболее важный элемент, который используется не столько для представления табличной информации, сколько для управления размещением фрагментов документа на экране. Таблица оформляется контейнером Table и содержит группы строк трех видов: *заголовка* (THead), *тела* (TBody) и *подвала* (TFoot). Первая и последняя – необязательны, тело – обязательно. Строки оформляются элементом Tr и содержат ячейки двух видов: *заголовка* (Th) и ячейки *данных* (Td). Пример таблицы с заголовком, подвалом и подписью представлен на рисунке 1.3.

Наименование продукции	Количество	Стоимость
Процессор Intel Pentium IV 1500	1,000	80,000.00
Процессор AMD Athlon 1400	1,300	84,500.00
Итого		**164,500.00**

Рисунок 1.3 – Сведения о продажах на 08.11.2006

```
<Table Align = "Center" Border=1>
    <Caption>Сведения о продажах на 08.11.2006</caption>
    <THead>
        <Tr>
            <Th>Наименование продукции</Th>
            <Th>Количество</Th>
            <Th>Стоимость</Th>
        </Tr>
    </THead>
    <TBody>
```

```
        <Tr>
           <Td>Процессор Intel Pentium IV 1500</Td>
           <Td>1,000</Td>
           <Td>80,000.00</Td>
        </Tr>
        <Tr>
           <Td>Процессор AMD Athlon 1400</Td>
           <Td>1,300</Td>
           <Td>84,500.00</Td>
        </Tr>
      <TFoot>
        <Tr>
           <Th>Итого</Th>
           <Th></Th>
           <Th>164,500.00</Th>
        </Tr>
      </TFoot>
      </TBody>
  </Table>
```

Для контейнера Table определены следующие необязательные атрибуты: Align (позволяет управлять горизонтальным выравниванием – "Left", "Center", "Right"), Border (толщина внешней границы таблицы в пикселах), CellSpacing (промежуток между ячейками, заполняемый цветом фона), CellPadding (отступы от всех четырех границ содержимого ячеек от границ ячеек), Height и Width (высота и ширина таблицы, размеры можно в пикселах и в процентах от высоты и ширины окна), BgColor (цвет фона), BackGround (картинка фона таблицы), BorderColor (цвет границы таблицы).

Теги THead, TBody, TFoot и Tr имеют следующие необязательные атрибуты: Align (выравнивание по горизонтали всех ячеек всех строк группы), Valign (выравниванием по вертикали – "Top", "Middle", "Bottom", "BaseLine") и др.

1.11 Гиперссылки

Существует три основных типа гиперссылок: внутренние, внешние и относительные.

Внутренние ссылки (internal links) – это ссылки на объекты в пределах одного документа. С их помощью пользователь может перемещаться по одной и той же web-странице между ее разделами.

Внешние (external links), или *удаленные (distant links) ссылки* – это ссылки на другие web-серверы.

Относительные (relative links), или *локальные (local links) ссылки* – это ссылки на другие web-страницы (или службы Internet), расположенные на одном сервере со страницей, содержащей ссылки.

В каждой ссылке содержится URL (Uniform Resource Locator), или *унифицированный локатор ресурсов,* являющийся адресом web-страницы, который отображается в адресном поле. Гиперссылка оформляется контейнером A с атрибутами Href (адрес гиперссылки), Target (имя окна или фрейма, в который будет загружен ресурс). Внутри контейнера можно указать объект (текст или картинку), по которому необходимо щелкнуть мышью для перехода по ссылке.

```
<A Href = "http://www.microsoft.com" target = "article> Домашняя страница Microsoft </A>
```

1.12 Списки стилей

Каскадные таблицы стилей (CSS) были реализованы в 1997 году. В качестве применяемой HTML-разметки они стали доступны с версии Internet Explorer 4.0. Стили основаны на разметке, существующей в языке HTML, и не являются заменой HTML. Назначение списка стилей – определить оформление того или иного элемента или набора элементов.

Существует три способа добавления информации о стиле в HTML-документ. Первый – использование внешнего списка стилей, который либо импортируется, либо связывается с текущим документом. Второй – включение информации о стилях, действующих в данном документе, в элемент <Head>. И третий – вставка информации о стиле непосредственно в том месте, где этот стиль должен действовать.

Внешний список стилей представляет собой простой текстовый файл, в котором записаны определения стилей для HTML-тегов или классов. Как правило, для обозначения, что данный файл является списком стилей, используется расширение .css (для стилей CSS1) либо .jss (для стилей JSS – JavaScript style sheet, не часто поддерживается браузерами).

Описание стилей или список стилей является набором правил, состоящих из HTML-элемента, класса, или идентификатора. HTML-элемент называется *селектором*. Ему присваивается определенное свойство, например Font-Family, после которого через двоеточие записывается значение этого свойства. В описании стиля могут использоваться несколько правил, которые отделяются друг от друга точкой с запятой:

```
{Font-Family: Impact; Font-Size: 28pt}
```

С появлением таблиц стилей в языке появилось три новых контейнера: Style, Link, Span. Контейнер Style (<Style Type="...">.....</Style>) служит для определения таблицы описания стилей и применяется внутри контейнера Head:

```
<HTML>
   <Head>
   <Style>
    St1{ Font-Family: Sans-Serif }
    St2{ Font-Family: Serif }
   </Style> </Head>
<Body>
 <St1>Sample text 1.</St1><Br>
 <St2>Sample text 2.</St2>
</Body> </HTML>
```

Контейнер Link в контексте описателей стилей применяется для определения внешнего файла с описаниями стилей для данного документа. Ниже представлен пример включения внешнего файл в документ.

Внешний файл имеет название StyleSheet.CSS и включает следующее описание стилей:

```
P {color:blue; Font-Family: Times; Font-Size: 10pt;}
H1 {color:black; Font-Size: 12pt; Font-Style: Arial;
Text-Align: Center;}
```

Для применения этого описателя стилей в заголовок документа необходимо включить тег <Link>:

```
<HTML>
<Head>
 <Link Rel=StyleSheet Type="text/css" Hr ef=css.htm>
</Head>
<Body>
 <P> Sample text 1.</P>
 <H1> Sample text 2.</H1>
</Body> </HTML>
```

Контейнер Span применяется для переопределения стиля отображения текущего фрагмента текста. Часто Span применяют для достижения типографских эффектов, таких, например, как выделение заглавной буквы абзаца:

```
<HTML>
<head>
   <Style Type="text/css">
    H1 {Color:navy; Text-transform: UpperCase; Font-Size: 18pt;
    Font-Weight: Bold; Font-Family: Times;}
    P {Color:navy; Font-Size: 12pt; Font-Family: Times; Text-
    Align: Justify}
   </Style> </Head>
<Body>
 <Center> <H1>Информационные Ресурсы Internet</h1> </Center>
 <P><Span>О</Span>Сети Internet исполнилось 25 лет.</P>
</Body> </HTML>
```

В данном примере контейнер Span применен сразу после тега начала параграфа <P>, что позволяет выделить первую букву в отображаемом абзаце. Можно также определить класс стилей и использовать его при помощи атрибута Class:

```
   <Style Type="text/css">
    H3.Class1 {Font-Size:12pt; Color= Blue}
   </Style>
.....
   <H3 Class="class1">This is a blue text</H3>
```

В данном примере определен класс заголовков третьего уровня, но можно определить класс, который можно будет применять к любым контейнерам, а не только к заголовкам:

```
   <Style Type="text/css">
    all.class1 {Font-Size:12pt; Color= Blue}
   </Style>
```

Кроме определения классов существует возможность создания поименованных стилей, которые создаются как уточнение какого-либо класса:

```
   <Style Type="text/css">
    all.class1 {Font-Size: 12pt; Color= Blue}
    #C1 {Font-Size: 20}
   </Style>
....
   <H3 Class="class1">This is a blue text</H3>
   <H3 Class="class1" Id="C1">This is a blue text</H3>
```

Таким образом, атрибуты контейнеров позволяют связать описатели стилей с содержанием контейнеров и управлять формой отображаемой информации.

2 ЯЗЫК UML

Унифицированный язык моделирования (UML) является стандартным инструментом для создания «чертежей» программного обеспечения. С помощью UML можно визуализировать, специфицировать, конструировать и документировать артефакты программных систем.

UML пригоден для моделирования любых систем: от информационных систем масштаба предприятия до распределенных web-приложений и даже встроенных систем реального времени. UML – это язык для визуализации, специфицирования, конструирования и документирования артефактов программных систем. Язык моделирования, подобный UML, является стандартным средством для составления «чертежей» программного обеспечения.

Для понимания любой нетривиальной системы приходится разрабатывать большое количество взаимосвязанных моделей. В применении к программным системам это означает, что необходим язык, с помощью которого можно с различных точек зрения описать представления архитектуры системы на протяжении цикла ее разработки.

UML – это графический язык специфицирования, что означает построение точных и полных графических моделей, касающихся анализа, проектирования и реализации, которые должны приниматься в процессе разработки и развертывания системы программного обеспечения.

UML – это язык конструирования, и модели, созданные с его помощью, могут быть непосредственно переведены на различные языки программирования. Иными словами, UML-модель можно отобразить на такие языки, как Java, C++, Visual Basic, и даже на таблицы реляционной базы данных.

Такое отображение модели на язык программирования позволяет осуществлять прямое проектирование: генерацию кода из модели UML в какой-то конкретный язык. Можно решить и обратную задачу: реконструировать модель по имеющейся реализации.

UML позволяет решить проблему документирования системной архитектуры и всех ее деталей, предлагает язык для формулирования требований к системе и определения тестов и, наконец, предоставляет средства для моделирования работ на этапе планирования проекта и управления версиями.

Использование UML эффективно в:

– информационных системах масштаба предприятия;
– банковских и финансовых услугах;
– телекоммуникациях;
– на транспорте;
– оборонной промышленности, авиации и космонавтике;
– розничной торговле;
– медицинской электронике;
– науке;
– распределенных web-системах.

Сфера применения UML не ограничивается моделированием ПО. Он позволяет моделировать, например, документооборот в юридических системах, структуру и функционирование системы обслуживания пациентов в больницах, осуществлять проектирование аппаратных средств.

2.1 Структура и компоненты языка UML

Общие принципы. Конструктивное использование языка UML основывается на понимании общих принципов моделирования сложных систем и особенностей процесса *объектно-ориентированного анализа и проектирования* (ООАП). Выбор выразительных средств для построения моделей сложных систем основывается на нескольких принципах.

Первым является принцип абстрагирования, который предписывает включать в модель только те аспекты проектируемой системы, которые имеют непосредственное отношение к выполнению системой своих функций или своего целевого предназначения. При этом все второстепенные детали опускаются, чтобы чрезмерно не усложнять процесс анализа и исследования полученной модели.

Вторым принципом построения моделей сложных систем является принцип многомодельности. Это значит, что никакое единственное представление сложной системы не является достаточным для адекватного выражения всех ее особенностей.

Еще одним принципом прикладного системного анализа является принцип иерархического построения моделей сложных систем. Этот принцип предписывает рассматривать процесс построения модели на разных уровнях абстрагирования или детализации в рамках фиксированных представлений.

Таким образом, процесс ООАП можно представить как поуровневый спуск от наиболее общих моделей и представлений концептуального уровня к более частным и детальным представлениям логического и физического уровня. 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.

Объектно-ориентированный анализ и проектирование системы предусматривает использование словаря языка UML, включающего три вида строительных блоков: сущности, отношения, диаграммы.

Сущности. Основными объектно-ориентированными блоками являются сущности. В языке UML имеется четыре вида сущностей: структурные, поведенческие, группирующие, аннотационные.

Структурные сущности – это имена существительные в моделях на языке UML. Они представляют собой статические части модели, соответствующие концептуальным или физическим элементам системы. Существует пять разновидностей концептуальных и логических сущностей.

Класс (Class) – это описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой (рисунок 2.1). Класс реализует один или несколько интерфейсов.

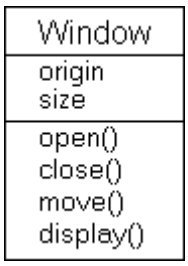


Рисунок 2.1 – Классы

Интерфейс (Interface) – это совокупность операций, которые определяют набор услуг, предоставляемый классом или компонентом. Интерфейс описывает видимое извне поведение элемента (рисунок 2.2). Интерфейс редко существует сам по себе – обычно он присоединяется к реализующему его классу или компоненту.

Рисунок 2.2 – Интерфейс

Кооперация (Collaboration) определяет структуру поведения системы в терминах взаимодействия участников этой кооперации и служит для обозначения множества взаимодействующих объектов в моделируемой системе. Кооперация имеет как структурный, так и поведенческий аспект, – один и тот же класс может принимать участие в нескольких кооперациях (рисунок 2.3).

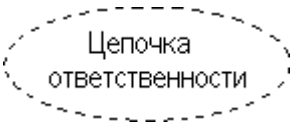


Рисунок 2.3 – Кооперация

Прецедент (Use case) – это описание последовательности выполняемых системой действий, которая производит наблюдаемый результат, значимый для какого-то определенного *актера* (Actor). Прецедент применяется для структурирования поведенческих сущностей модели и реализуется посредством кооперации (рисунок 2.4).

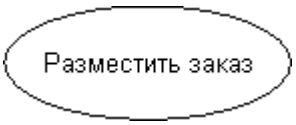


Рисунок 2.4 – Прецедент

Активным классом (Active class) называется класс, объекты которого вовлечены в один или несколько процессов (Threads), и могут инициировать управляющее воздействие. Активный класс отличается от обычного класса тем, что деятельность его объектов осуществляется одновременно с деятельностью других элементов. Графически активный класс изображается так же, как простой класс, но ограничивающий прямоугольник рисуется жирной линией и обычно включает имя, атрибуты и операции (рисунок 2.5).

Рисунок 2.5 – Активный класс

Компоненты и узлы соответствуют физическим сущностям системы.

Компонент (Component) – это физическая заменяемая часть системы, соответствующая некоторому набору интерфейсов и обеспечивающая его реализацию (рисунок 2.6). Компонент – это физическая упаковка логических элементов, (классов, интерфейсов и кооперации), например компоненты COM+ или Java Beans, а также файлы исходного кода.

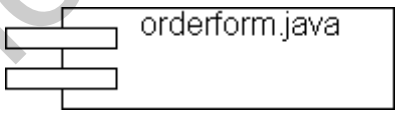


Рисунок 2.6 – Компонент

Узел (Node) – это элемент, представляющий вычислительный ресурс, обладающий памятью и способностью обработки. Совокупность компонентов может размещаться в узле, а также мигрировать с одного узла на другой (рисунок 2.7).

Рисунок 2.7 – Узел

Существуют также разновидности этих семи сущностей: актеры, сигналы, утилиты (виды классов), процессы и нити (виды активных классов), приложения, документы, файлы, библиотеки, страницы и таблицы (виды компонентов).

Поведенческие сущности (Behavioral things) являются динамическими составляющими модели UML. Это глаголы языка: они описывают поведение модели. Существует всего два типа поведенческих сущностей.

Взаимодействие (Interaction) – это поведение, суть которого заключается в обмене сообщениями между объектами для достижения определенной цели. С помощью взаимодействия описывается как отдельная операция, так и поведение совокупности объектов. Взаимодействие предполагает ряд других элементов, таких, как сообщение (рисунок 2.8), последовательность действий (поведение, инициированное сообщением) и связь (между объектами).

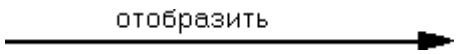


Рисунок 2.8 – Сообщение

Автомат (State machine) – это алгоритм поведения, определяющий последовательность состояний, через которые объект или взаимодействие проходят на протяжении своего жизненного цикла в ответ на различные события, а также реакции на эти события (рисунок 2.9). С помощью автомата можно описать поведение отдельного класса или кооперации классов. С автоматом связан ряд других элементов: состояния, переходы (из одного состояния в другое), события (сущности, инициирующие переходы) и виды действий (реакция на переход).

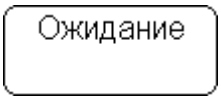


Рисунок 2.9 – Состояние

Взаимодействия и автоматы семантически связаны с различными структурными элементами, такими, как классы, кооперации и объекты.

Группирующие сущности являются организующими частями модели, это блоки, на которые можно разложить модель. Основной группирующей сущностью является пакет.

Пакет (Package) – это универсальный механизм организации элементов в группы (рисунок 2.10). В пакет можно поместить структурные, поведенческие и даже другие группирующие сущности. В отличие от компонентов, существующих во время работы программы, пакеты носят чисто концептуальный характер, т. е. существуют только во время разработки.

Рисунок 2.10 – Пакеты

Существуют также вариации пакетов, например, каркасы (Frameworks), модели и подсистемы.

Аннотационные сущности – пояснительные части модели UML. Это комментарии для дополнительного описания, разъяснения или замечания к любому элементу модели. Имеется только один базовый тип аннотационных элементов – примечание.

Примечание (Note) – это символ для изображения комментариев, присоединенных к элементу или группе элементов (рисунок 2.11).

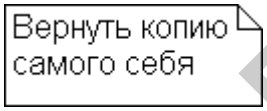


Рисунок 2.11 – Примечание

Примечания предназначены для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта. В качестве такой информации могут быть комментарии разработчика, ограничения и помеченные значения.

Отношения. В языке UML определены четыре типа **отношений**: зависимость, ассоциация, обобщение, реализация. Эти отношения являются основными связующими строительными блоками в UML.

Зависимость (Dependency) – это семантическое отношение между двумя сущностями, при котором изменение одной из них, независимой, может повлиять на семантику другой, зависимой (рисунок 2.12).

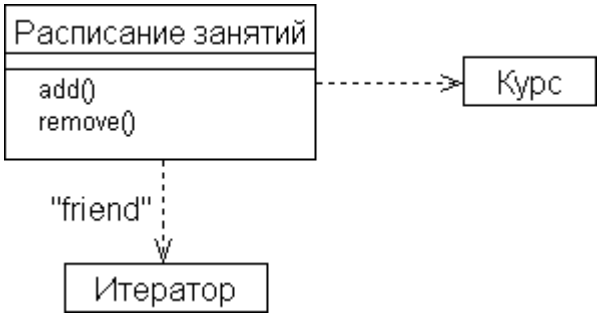


Рисунок 2.12 – Отношения зависимости

Ассоциация (Association) – отношение, описывающее совокупность связей между объектами. Разновидностью ассоциации является *агрегирование* (Aggregation) – структурное отношение между целым и его частями (рисунок 2.13). Графическое изображение ассоциации может включать кратность и имена ролей (рисунок 2.14).

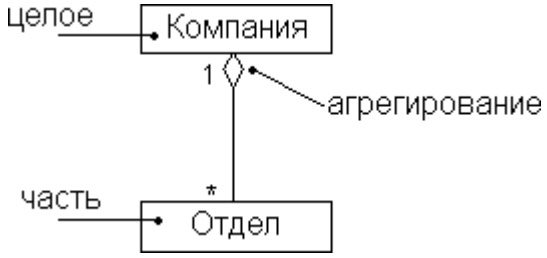


Рисунок 2.13 – Агрегирование

Рисунок 2.14 – Имена ассоциаций

Обобщение (Generalization) – это отношение «специализация/обобщение» (рисунок 2.15), при котором объект специализированного элемента (потомок) может быть подставлен вместо объекта обобщенного элемента (родителя или предка). Таким образом, потомок (Child) наследует структуру и поведение своего родителя (Parent).

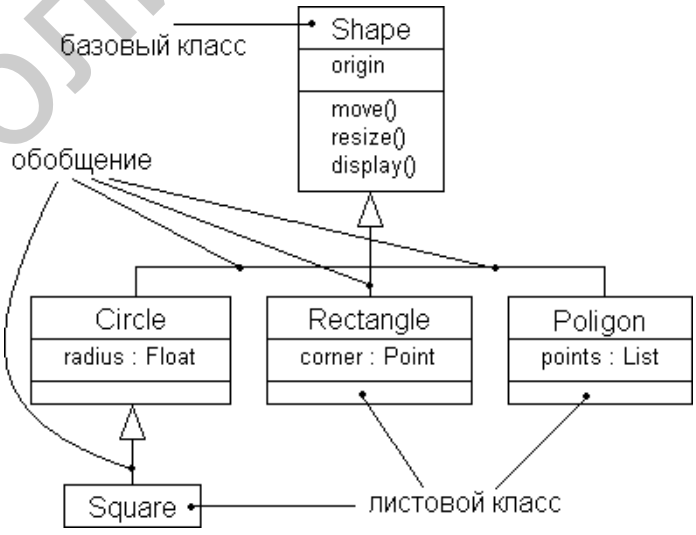


Рисунок 2.15 – Обобщение

Наконец, *реализация* (Realization) – это отношение между классификаторами, при котором один классификатор определяет «контракт», а другой гарантирует его выполнение (рисунок 2.16).

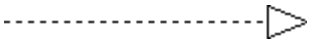


Рисунок 2.16 – Реализация

Отношения реализации встречаются в двух случаях: во-первых, между интерфейсами и реализующими их классами или компонентами, а во-вторых, между прецедентами и реализующими их кооперациями.

Диаграммы. **Диаграмма** в UML – это графическое представление набора элементов, изображаемое в виде связанного графа с вершинами (сущностями) и ребрами (отношениями), используемое для визуализации системы с разных точек зрения. В UML выделяют 8 типов диаграмм (рисунок 2.17).

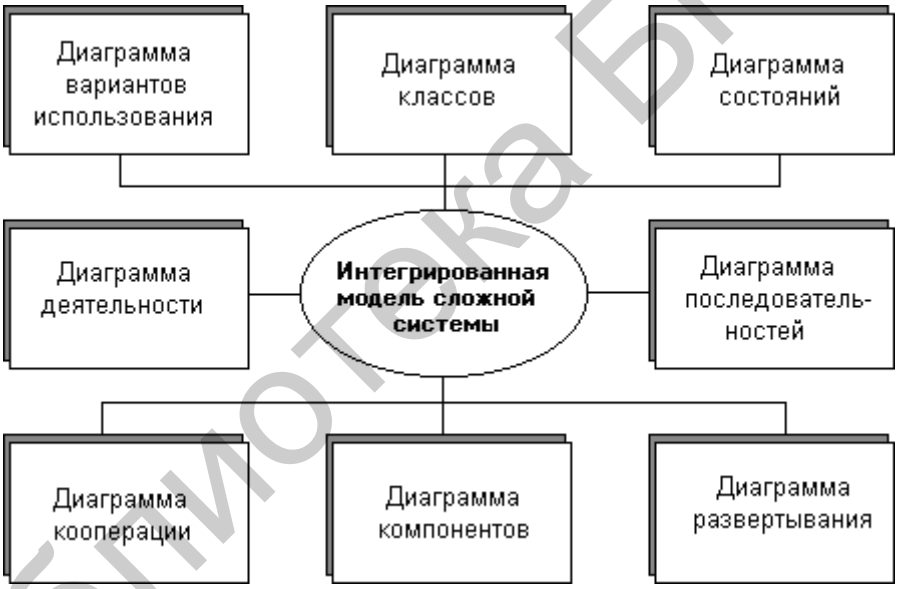


Рисунок 2.17 – Интегрированная модель сложной системы в нотации UML

На *диаграмме классов* (Class diagram) изображаются классы, интерфейсы, объекты и кооперации, а также их отношения. Используется при моделировании объектно-ориентированных систем.

На *диаграмме вариантов использования* (Use case diagram) представлены прецеденты и актеры (частный случай классов), а также отношения между ними. Они используются при моделировании поведения системы.

Диаграммы *последовательностей* (Sequence diagram) и *кооперации* (Collaboration diagram) являются частными случаями диаграмм взаимодействия. На диаграммах взаимодействия представлены связи между объектами (сообще-

ния, которыми объекты могут обмениваться). Диаграммы взаимодействия относятся к динамическому виду системы. При этом диаграммы последовательности отражают временную упорядоченность сообщений, а диаграммы кооперации – структурную организацию обменивающихся сообщениями объектов. Эти диаграммы могут быть преобразованы друг в друга.

На *диаграммах состояний* (Statechart diagrams) представлен автомат, включающий состояния, переходы, события и виды действий. Диаграммы состояний используются при моделировании поведения интерфейса, класса или кооперации, зависящем от последовательности событий.

Диаграмма деятельности (Activity diagram) представляет переходы потока управления между объектами от одной деятельности к другой внутри системы.

Диаграмма компонентов (Component diagram) представляет зависимости между компонентами. Диаграммы компонентов отображаются на один или несколько классов, интерфейсов или кооперацией.

На *диаграмме развертывания* (Deployment diagram) представлена конфигурация обрабатывающих узлов системы и размещенных в них компонентов.

2.2 Диаграммы вариантов использования (use case diagram)

На диаграммах вариантов использования отображается взаимодействие между вариантами использования, представляющими функции системы, и действующими лицами, представляющими людей или системы, получающие или передающие информацию в данную систему. Из диаграмм вариантов использования можно получить довольно много информации о системе. Этот тип диаграмм описывает общую функциональность системы. Пользователи, менеджеры проектов, аналитики, разработчики, специалисты по контролю качества и все, кого интересует система в целом, могут, изучая диаграммы вариантов использования, понять, что система должна делать.

Базовые элементы диаграммы вариантов использования. К базовым элементам рассматриваемой диаграммы относятся *вариант использования*, *актер* и *интерфейс*.

Вариант использования (рисунок 2.18) применяется для спецификации общих особенностей поведения системы или другой сущности без рассмотрения ее внутренней структуры (например, оформление заказа на покупку товара, получение информации о кредитоспособности клиента, отображение графической формы на экране монитора).

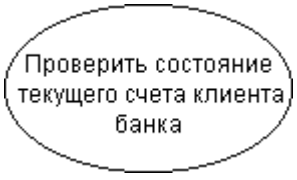


Рисунок 2.18 – Графическое обозначение варианта использования

Актер – это внешняя по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для решения определенных задач (рисунок 2.19). При этом актеры служат для обозначения согласованного множества ролей, которые могут играть пользователи в процессе взаимодействия с проектируемой системой.

Рисунок 2.19 – Графическое обозначение актера

Имя актера должно быть достаточно информативным с точки зрения семантики, например, клиент банка, продавец магазина, пассажир авиарейса, водитель автомобиля, сотовый телефон.

Так как в общем случае актер всегда находится вне системы, его внутренняя структура никак не определяется. Для актера имеет значение только его внешнее представление, т. е. то, как он воспринимается со стороны системы. Актеры взаимодействуют с системой посредством передачи и приема сообщений от вариантов использования. Сообщение представляет собой запрос актером сервиса от системы и получение этого сервиса. Это взаимодействие может быть выражено посредством ассоциаций между отдельными актерами и вариантами использования или классами. Кроме этого, с актерами могут быть связаны интерфейсы, которые определяют, каким образом другие элементы модели взаимодействуют с этими актерами.

Интерфейс служит для спецификации параметров модели, которые видимы извне без указания их внутренней структуры (рисунок 2.20). В диаграммах вариантов использования интерфейсы определяют совокупность операций, обеспечивающих необходимый набор сервисов или функциональности для актеров. Интерфейсы не могут содержать ни атрибутов, ни состояний, ни направленных ассоциаций. Они содержат только операции без указания особенностей их реализации. Формально интерфейс эквивалентен абстрактному классу без атрибутов и методов с наличием только абстрактных операций.

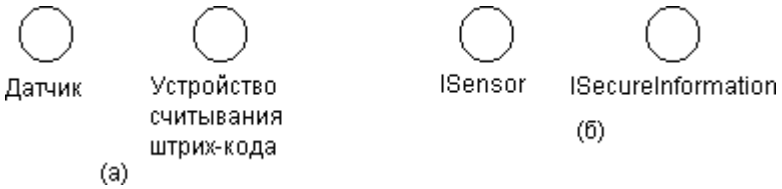


Рисунок 2.20 – Графическое изображение интерфейсов на диаграммах вариантов использования

Интерфейс соединяется с вариантом использования сплошной линией, если он реализует все операции, необходимые для данного интерфейса (рисунок 2.21, а). Если же вариант использования определяет только тот сервис, который необходим для реализации данного интерфейса, используется пунктирная стрелка (рисунок 2.21, б).

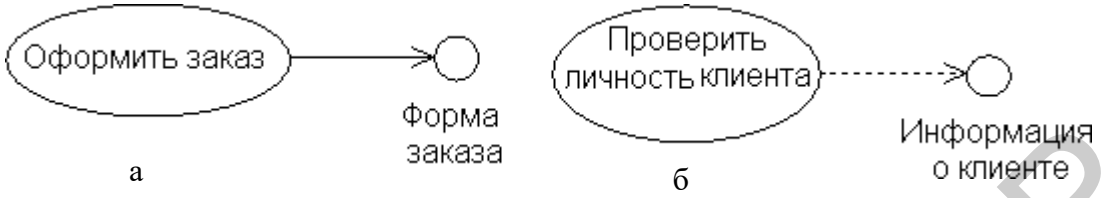


Рисунок 2.21 – Графическое изображение взаимосвязей интерфейсов с вариантами использования

Важность интерфейсов заключается в том, что они определяют стыковочные узлы в проектируемой системе, что совершенно необходимо для организации коллективной работы над проектом. Более того, спецификация интерфейсов способствует «безболезненной» модификации уже существующей системы при переходе на новые технологические решения. В этом случае изменению подвергается только реализация операций, но никак не функциональность самой системы. А это обеспечивает совместимость последующих версий программ с первоначальными при спиральной технологии разработки программных систем.

Примечания в языке UML предназначены для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта (рисунок 2.22). В качестве такой информации могут быть комментарии разработчика (например, дата и версия разработки диаграммы или ее отдельных компонентов), ограничения (например, на значения отдельных связей или экземпляры сущностей) и помеченные значения.

Рисунок 2.22 – Примеры примечаний в языке UML

Применительно к диаграммам вариантов использования примечание может носить самую общую информацию, относящуюся к общему контексту системы.

Отношения на диаграмме вариантов использования. Для выражения отношений между актерами и вариантами использования применяются стандартные виды отношений, описанные в разделе 2.1.

Отношение ассоциации применительно к диаграммам вариантов использования служит для обозначения специфической роли актера в отдельном варианте использования (рисунок 2.23). Другими словами, ассоциация определяет семантические особенности взаимодействия актеров и вариантов использования в графической модели системы. Таким образом, это отношение устанавливает, какую конкретную роль играет актер при взаимодействии с экземпляром варианта использования. Графическое обозначение отношения ассоциации может включать дополнительные условные обозначения (имя и кратность).

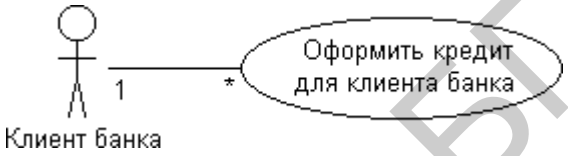


Рисунок 2.23 – Отношение ассоциации между актером
и вариантом использования

Отношение расширения между вариантами использования обозначается пунктирной линией со стрелкой (вариант отношения зависимости), направленной от того варианта использования, который является расширением для исходного варианта использования. Данная линия со стрелкой помечается ключевым словом “extend” («расширяет»), как показано на рисунке 2.24.

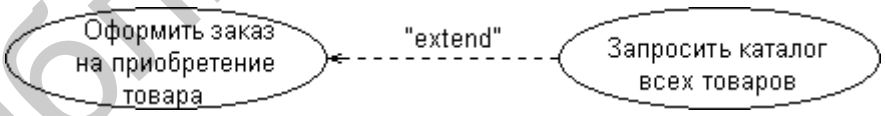


Рисунок 2.24 – Отношение расширения между вариантами использования

Отношение расширения отмечает тот факт, что один из вариантов использования может присоединять к своему поведению некоторое дополнительное поведение, определенное для другого варианта использования.

Отношение обобщения графически обозначается сплошной линией со стрелкой, которая указывает на родительский вариант использования (рисунок 2.25).

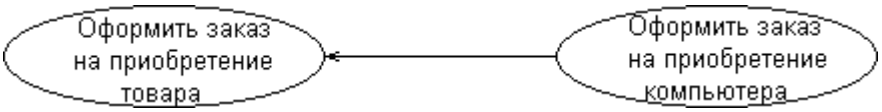


Рисунок 2.25 – Отношение обобщения между вариантами использования

Отношение обобщения между вариантами использования применяется в том случае, когда необходимо отметить, что дочерние варианты использования обладают всеми атрибутами и особенностями поведения родительских вариантов. При этом дочерние варианты использования участвуют во всех отношениях родительских вариантов. В свою очередь дочерние варианты могут наделяться новыми свойствами поведения, которые отсутствуют у родительских вариантов использования, а также уточнять или модифицировать наследуемые от них свойства поведения.

Отношение включения между двумя вариантами использования указывает, что поведение одного варианта использования включается в качестве составного компонента в последовательность поведения другого варианта использования. Графически данное отношение обозначается пунктирной линией со стрелкой (вариант отношения зависимости), направленной от базового варианта использования к включаемому. При этом данная линия со стрелкой помечается ключевым словом “include” («включает»), как показано на рисунке 2.26.

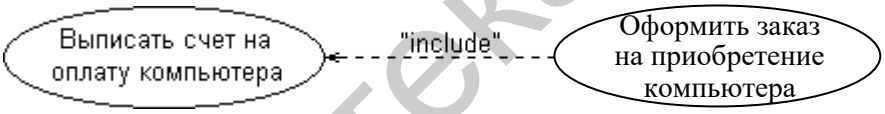


Рисунок 2.26 – Отношение включения между вариантами использования

При моделировании возникает необходимость в указании количества объектов, связанных посредством одного экземпляра ассоциации. Это число называется *кратностью* (Multiplicity) роли ассоциации и записывается либо как выражение, значением которого является диапазон значений, либо в явном виде (рисунок 2.27). Кратность указывает на то, столько объектов должно соответствовать каждому объекту на противоположном конце. Кратность можно задать равной единице (1), указать диапазон: «ноль или единица» (0…1), «много» (0…*), «единица или больше» (1…*). Разрешается также указывать определенное число (например, 3).

Рисунок 2.27 – Кратность

Пример диаграммы вариантов использования. Рассмотрим диаграмму вариантов использования, отражающую систему работы банковского автомата (Automated Teller Machine, ATM) (рисунок 2.28).

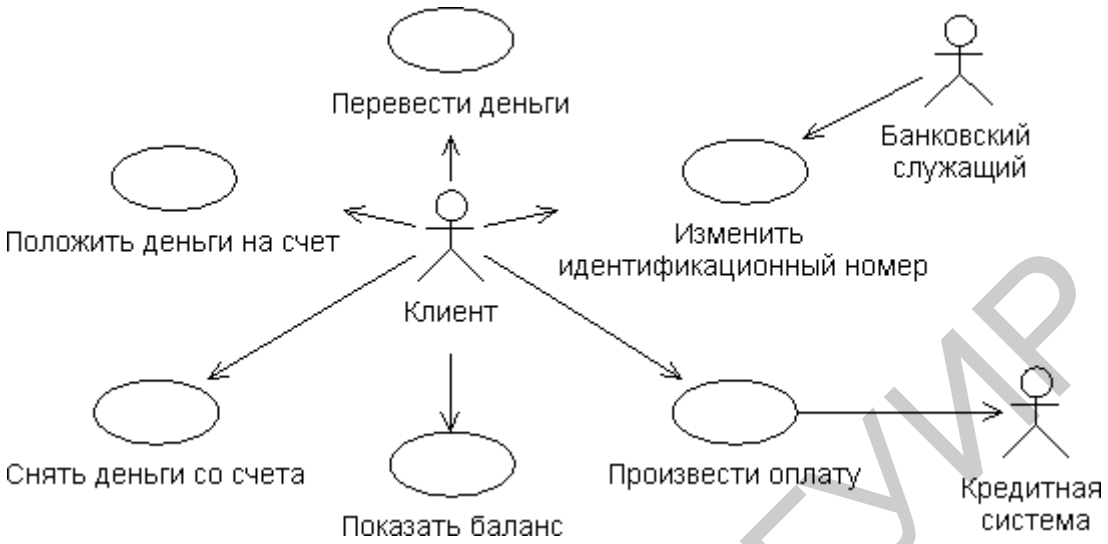


Рисунок 2.28 – Диаграмма вариантов использования для ATM

Клиент банка инициирует различные варианты использования: снять деньги со счета, перевести деньги, положить деньги на счет, показать баланс, изменить идентификационный номер, произвести оплату. Банковский служащий может инициировать вариант использования «Изменить идентификационный номер». Действующими лицами могут быть и внешние системы, в данном случае кредитная система показана именно как действующее лицо – она является внешней для системы ATM. Стрелка, направленная от варианта использования к действующему лицу, показывает, что вариант использования предоставляет некоторую информацию действующему лицу. В данном случае вариант использования «Произвести оплату» предоставляет кредитной системе информацию об оплате по кредитной карточке.

2.3 Диаграммы последовательности (sequence diagram)

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы взаимодействия. При этом учитываются два аспекта: во-первых, взаимодействия объектов можно рассматривать во времени, и тогда для представления временных особенностей передачи и приема сообщений между объектами используется *диаграмма последовательности*. Во-вторых, можно рассматривать структурные особенности взаимодействия объектов. Для представления структурных особенностей передачи и приема сообщений между объектами используется диаграмма кооперации.

Объекты диаграммы последовательности. Диаграммы последовательности отражают поток событий, происходящих в рамках варианта использования. На этих диаграммах изображаются только те объекты, которые непосредственно участвуют во взаимодействии т. к. ключевым моментом является именно динамика взаимодействия объектов во времени и не используются возможные статические ассоциации с другими объектами. При этом диаграмма последовательности имеет два измерения (рисунок 2.29). Одно – слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии. Второе измерение – вертикальная временная ось, направленная сверху вниз. При этом взаимодействия объектов реализуются посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения и также образуют порядок по времени своего возникновения. Другими словами, сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже.

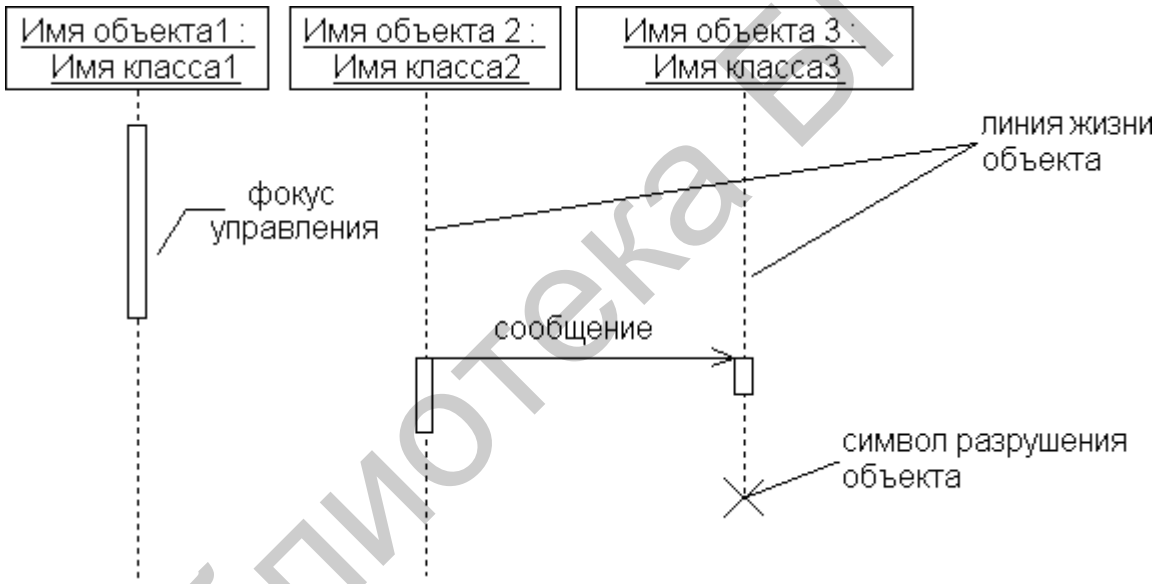


Рисунок 2.29 – Графические примитивы диаграммы последовательности

Линия жизни объекта (object lifeline) изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности. Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе и, следовательно, может потенциально участвовать во всех ее взаимодействиях. Если объект существует в системе постоянно, то его линия жизни должна начинаться в верхней части диаграммы и заканчиваться в нижней части (объекты 1 и 2 на рисунке 2.30). Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены, чтобы освободить занимаемые ими ресурсы. Для обозначения момента уничтожения объекта в языке UML используется специальный символ в форме

латинской буквы “X” (объект 3 на рисунке 2.30). Ниже этого символа пунктирная линия не изображается, поскольку соответствующего объекта в системе уже нет, и этот объект должен быть исключен из всех последующих взаимодействий.

Отдельные объекты в системе могут создаваться по мере необходимости, существенно экономя ресурсы системы и повышая ее производительность (объект 6 на рисунке 2.31).

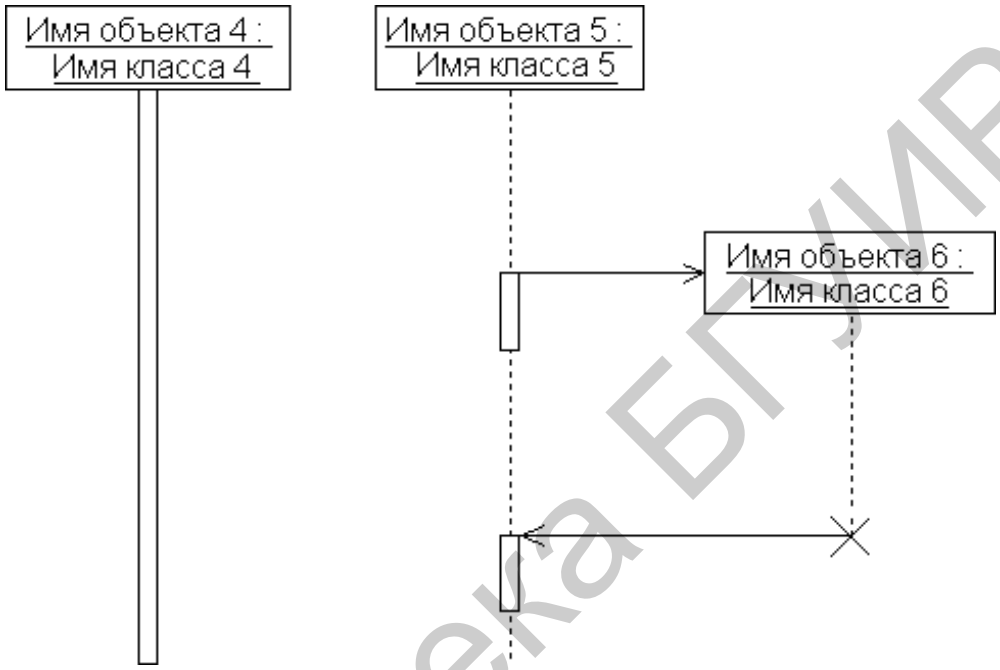


Рисунок 2.30 – Варианты линий жизни и фокусов управления объектов

Комментарии или примечания уже рассматривались ранее при изучении других видов диаграмм. Они могут включаться и в диаграммы последовательности, ассоциируясь с отдельными объектами или сообщениями.

Как уже отмечалось выше, взаимодействия объектов реализуются с помощью сообщений. У каждого сообщения должно быть имя, соответствующее его цели. Существует несколько видов сообщений: простое, синхронное, с отказом становиться в очередь и др. (рисунок 2.31).

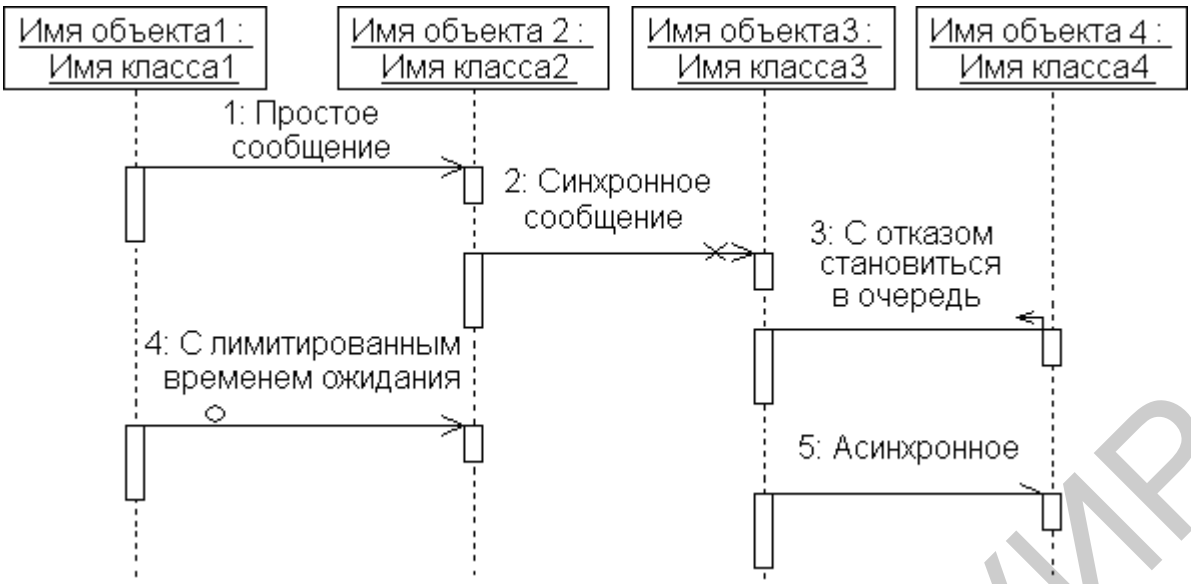


Рисунок 2.31– Примеры сообщений

Простое сообщение используется по умолчанию. Означает, что все сообщения выполняются в одном потоке управления (рисунок 2.31, 1).

Синхронное (synchronous) применяется, когда клиент посылает сообщение и ждет ответа пользователя (рисунок 2.31, 2).

Сообщение с отказом становиться в очередь (balking): клиент посылает сообщение серверу и, если сервер не может немедленно принять сообщение, оно отменяется (рисунок 2.31, 3).

Сообщение с лимитированным временем ожидания (timeout): клиент посылает сообщение серверу, а затем ждет указанное время; если в течение этого времени сервер не принимает сообщение, оно отменяется (рисунок 2.31, 4).

Асинхронное сообщение (asynchronous): клиент посылает сообщение серверу и продолжает свою работу, не ожидая подтверждения о получении (рисунок 2.31, 5).

Пример диаграммы последовательности. Пример сценария снятия 20 дол. со счета (при отсутствии таких проблем, как неправильный идентификационный номер или недостаток денег на счете) показан на рисунке 2.32.

Эта диаграмма последовательности отображает поток событий в рамках варианта использования «Снять деньги». В верхней части диаграммы показаны все действующие лица и объекты, требуемые системе для выполнения варианта использования «Снять деньги». Стрелки соответствуют сообщениям, передаваемым между действующим лицом и объектом или между объектами для выполнения требуемых функций. Следует отметить также, что на диаграмме «Последовательности» показаны именно объекты, а не классы. Классы представляют собой типы объектов. Объекты конкретны; вместо класса «Клиент» на диаграмме «Последовательности» представлен конкретный клиент Джо.

Вариант использования начинается, когда клиент вставляет свою карточку в устройство для чтения – этот объект показан в прямоугольнике в верхней

части диаграммы. Он считывает номер карточки, открывает объект «счет» (account) и инициализирует экран АТМ. Экран запрашивает у клиента его регистрационный номер. Клиент вводит число 1234. Экран проверяет номер у объекта «счет» и обнаруживает, что он правильный. Затем экран предоставляет клиенту меню для выбора, и тот выбирает пункт *«Снять деньги»*. Экран запрашивает, сколько он хочет снять, и клиент указывает 20 дол. Экран снимает деньги со счета. При этом он инициирует серию процессов, выполняемых объектом «счет». Во-первых, осуществляется проверка, что на этом счете лежат, по крайней мере, 20 дол. Во-вторых, из счета вычитается требуемая сумма. Затем кассовый аппарат получает инструкцию выдать чек и 20 дол. наличными. Наконец все тот же объект «счет» дает устройству для чтения карточек инструкцию вернуть карточку.

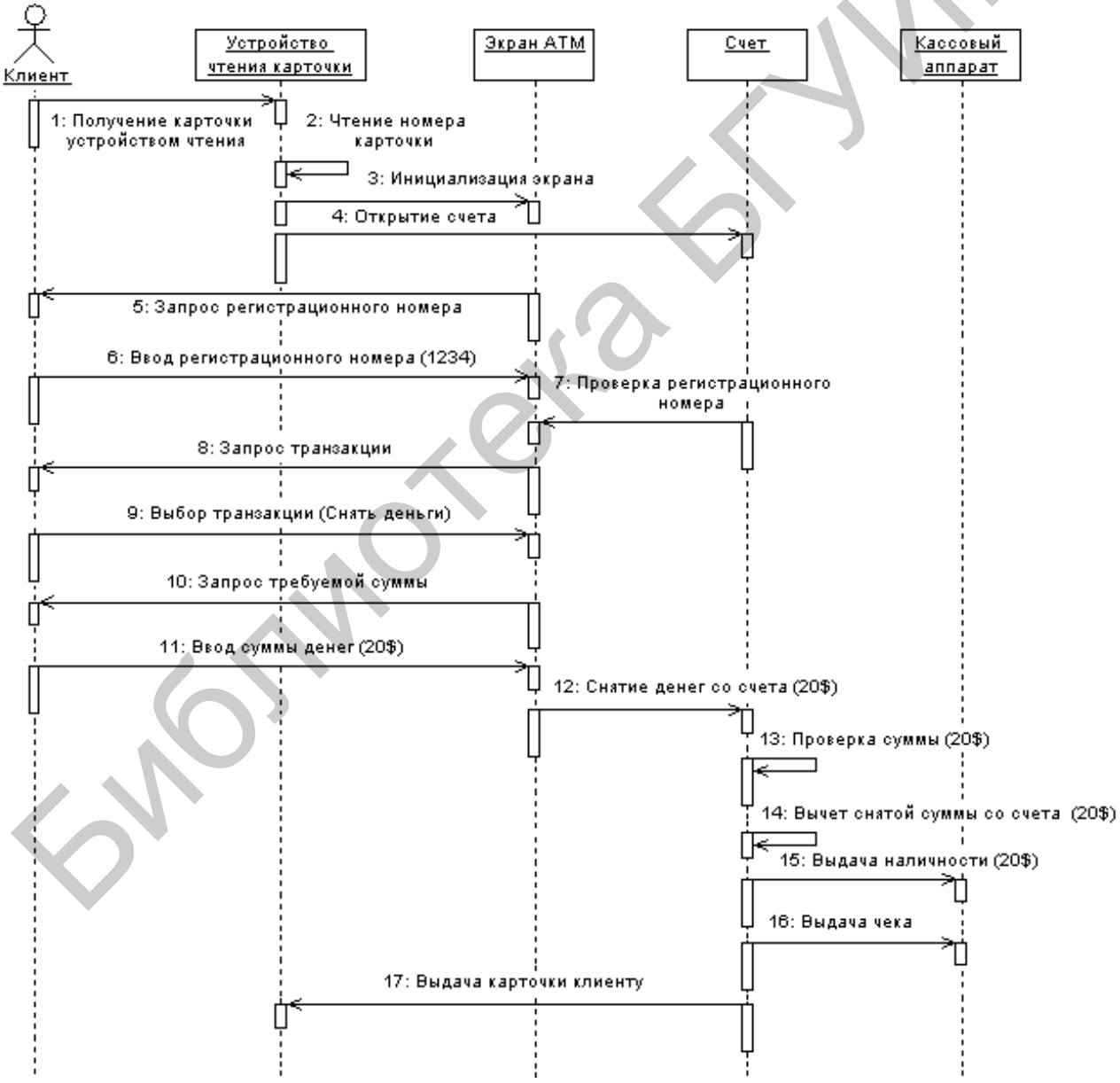


Рисунок 2.32 – Диаграмма последовательности для снятия клиентом 20 дол.

Таким образом, диаграмма последовательности иллюстрирует последовательность действий, реализующих вариант использования «Снять деньги со счета» на примере снятия клиентом 20 дол. Глядя на эту диаграмму, пользователи знакомятся со спецификой своей работы. Аналитики видят последовательность (поток) действий, разработчики – объекты, которые надо создать, и их операции. Специалисты по контролю качества поймут детали процесса и смогут разработать тесты для их проверки. Таким образом, диаграммы последовательности полезны всем участникам проекта.

2.4 Диаграммы кооперации (collaboration diagram)

Подобно диаграммам последовательности *диаграммы кооперации* отображают поток событий в конкретном сценарии варианта использования. Главная особенность диаграммы кооперации заключается в возможности графически представить не только последовательность взаимодействия, но и все структурные отношения между объектами, участвующими в этом взаимодействии.

Прежде всего на диаграмме кооперации в виде прямоугольников изображаются участвующие во взаимодействии объекты, содержащие имя объекта, его класс и, возможно, значения атрибутов. Далее, как и на диаграмме классов, указываются ассоциации между объектами в виде различных соединительных линий. При этом можно явно указать имена ассоциации и ролей, которые играют объекты в данной ассоциации. Дополнительно могут быть изображены динамические связи – потоки сообщений. Они представляются также в виде соединительных линий между объектами, над которыми располагается стрелка с указанием направления, имени сообщения и порядкового номера в общей последовательности инициализации сообщений.

В отличие от диаграммы последовательности на диаграмме кооперации изображаются только отношения между объектами, играющими определенные роли во взаимодействии, а последовательность взаимодействий и параллельных потоков определяется с помощью порядковых номеров.

Объекты диаграммы кооперации. Отдельные аспекты спецификации объектов как элементов диаграмм уже рассматривались ранее при описании диаграмм последовательности. Эти же объекты являются основными элементами, из которых строится диаграмма кооперации. Для графического изображения объектов используется такой же символ прямоугольника, что и для классов.

Объект является отдельным экземпляром класса, который создается на этапе выполнения программы. Он может иметь свое собственное имя и конкретные значения атрибутов. Для обозначения роли классификатора необходимо указать либо имя класса (вместе с двоеточием), либо имя роли (вместе с наклонной чертой). В противном случае прямоугольник будет соответствовать обычному классу. Если роль, которую должен играть объект, наследуется от нескольких классов, то все они должны быть указаны явно и разделяться запятой и двоеточием.

Отдельные примеры изображения объектов и классов на диаграмме кооперации приводятся на рисунок 2.33. В первом случае (рисунок 2.33, а) обозначен объект с именем «клиент», играющий роль «инициатор запроса». Далее (рисунок 2.33, б) следует обозначение анонимного объекта, который играет роль «инициатор запроса». В обоих случаях не указан класс, на основе которого будут созданы эти объекты. Обозначение класса присутствует в следующем варианте записи (рисунок 2.33, в), причем объект также анонимный.

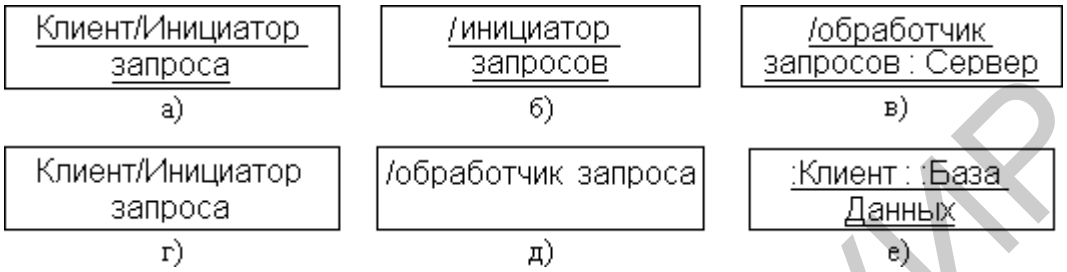


Рисунок 2.33 – Варианты записи имен объектов, ролей и классов на диаграммах кооперации

Применительно к уровню спецификации на диаграммах кооперации могут присутствовать именованные классы с указанием роли класса в кооперации (рисунок 2.33, г) или анонимные классы, когда указывается только его роль (рисунок 2.33, д). Последний случай характерен для ситуации, когда в модели могут присутствовать несколько классов с именем «клиент», поэтому требуется явно указать имя соответствующего пакета «База данных» (рисунок 2.33, е).

Мультиобъект (multiobject) представляет собой целое множество объектов на одном из концов ассоциации (рисунок 2.34, а). На диаграмме кооперации мультиобъект используется, чтобы показать операции и сигналы, адресованные всему множеству объектов, а не только одному. При этом стрелка сообщения относится ко всему множеству объектов, которые обозначают данный мультиобъект. На диаграмме кооперации может быть явно указано отношение композиции между мультиобъектом и отдельным объектом из его множества (рисунок 2.34, б).

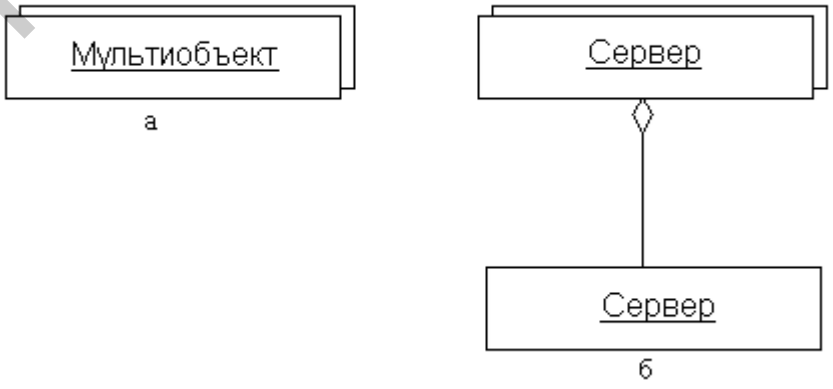


Рисунок 2.34 – Графическое изображение мультиобъектов на диаграмме кооперации

В контексте языка UML все объекты делятся на две категории: *пассивные* и *активные*. Пассивный объект оперирует только данными и не может инициировать деятельность по управлению другими объектами. В то же время пассивные объекты могут посылать сигналы в процессе выполнения запросов, которые они получают.

Активный объект имеет свою собственную нить управления и может инициировать деятельность по управлению другими объектами. При этом под нитью понимается поток управления, который может выполняться параллельно с другими вычислительными нитями или нитями управления в пределах одного вычислительного процесса.

В приведенном фрагменте диаграммы кооперации (рисунок 2.35) активный объект «a: Вызывающий абонент» является инициатором процесса установления соединения для обмена информацией с другим абонентом.

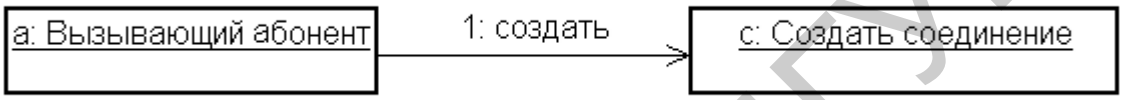


Рисунок 2.35 – Активный объект (слева) на диаграмме кооперации

Составной объект (composite object) или объект-контейнер предназначен для представления объекта, имеющего собственную структуру и внутренние потоки (нити) управления. Составной объект является экземпляром составного класса (класса-контейнера), который связан отношением агрегации или композиции со своими частями. Аналогичные отношения связывают между собой и соответствующие объекты.

На диаграммах кооперации составной объект состоит из двух секций: верхней и нижней. В верхней секции записывается имя составного объекта, а в нижней – его элементы (рисунок 2.36), которые могут быть составными объектами.

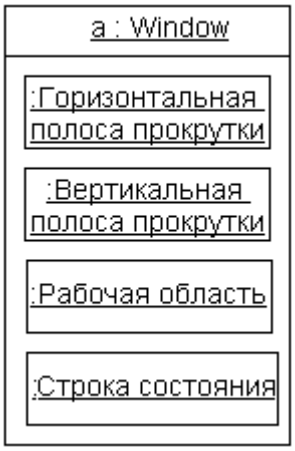


Рисунок 2.36 – Составной объект на диаграмме кооперации

Связь (link) является экземпляром или примером произвольной ассоциации. Связь как элемент языка UML может иметь место между двумя и более объектами. Связь на диаграмме кооперации изображается отрезком прямой линии, соединяющей два прямоугольника объектов. На каждом из концов этой линии могут быть явно указаны имена ролей данной ассоциации. Рядом с линией в ее средней части может записываться имя соответствующей ассоциации. Связи не имеют собственных имен, поскольку полностью идентичны как экземпляры ассоциации. Для связей не указывается также и кратность.

Применительно к диаграммам кооперации сообщения имеют некоторые дополнительные семантические особенности. Они определяют коммуникацию между двумя объектами, один из которых передает другому некоторую информацию. При этом первый объект ожидает, что после получения сообщения вторым объектом последует выполнение некоторого действия. Таким образом, именно сообщение является причиной или стимулом для начала выполнения операций, отправки сигналов, создания и уничтожения отдельных объектов. Связь обеспечивает канал для направленной передачи сообщений между объектами от объекта-источника к объекту-получателю.

Пример диаграммы кооперации. На рисунке 2.37 приведена кооперативная диаграмма, описывающая, как клиент снимает со счета 20 дол.

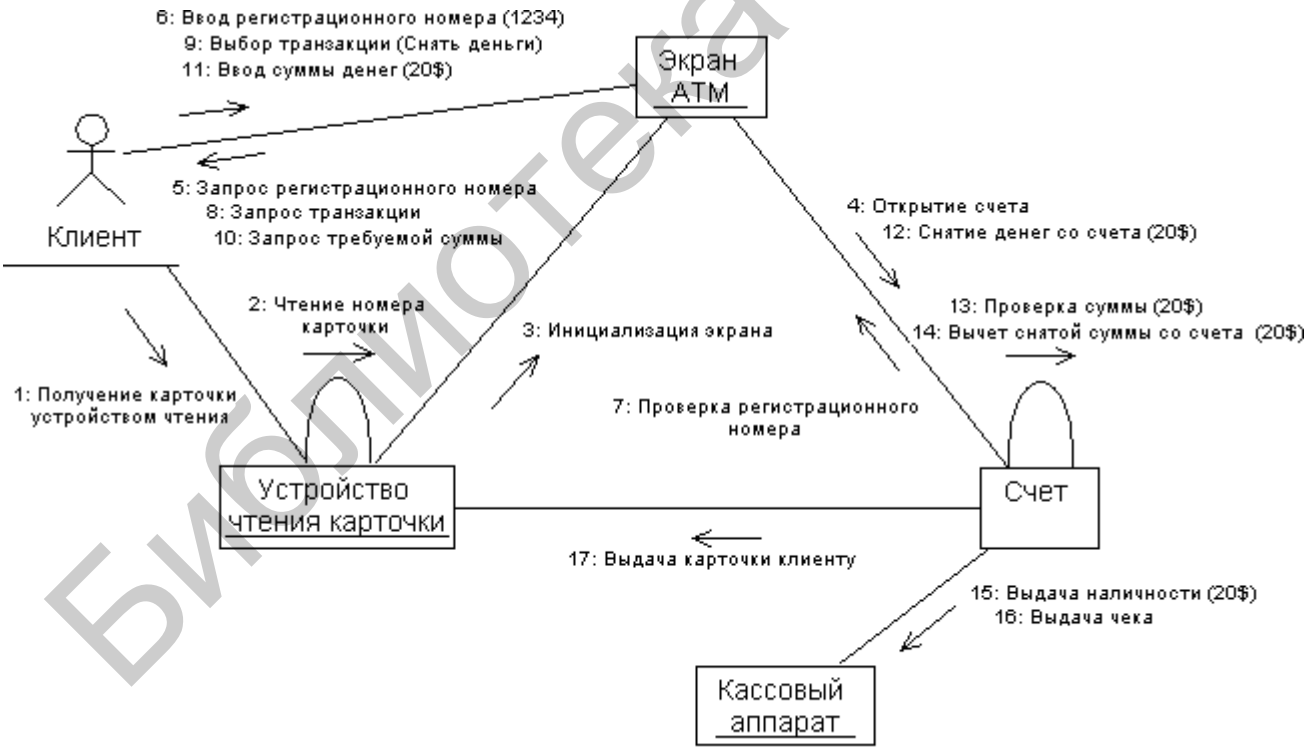


Рисунок 2.37 – Диаграмма кооперации для снятия клиентом 20 дол.

Из диаграммы кооперации легче понять поток событий и отношения между объектами, однако труднее уяснить последовательность событий, поэтому для сценария создают диаграммы обоих типов.

2.5 Диаграммы классов (class diagram)

Диаграммы классов при моделировании объектно-ориентированных систем встречаются чаще других. На таких диаграммах отображается множество классов, интерфейсов, кооперации и отношений между ними. Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Кроме того, диаграммы классов составляют основу еще двух диаграмм – компонентов и развертывания.

Диаграмма классов может отражать различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы.

Компоненты диаграммы классов. Классом называется описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Например, класс «стена» описывает объекты с общими свойствами: высотой, длиной, толщиной и т. д. При этом конкретные стены будут рассматриваться как отдельные экземпляры класса «стена». У каждого класса есть имя, (простое или составное, к которому спереди добавлено имя пакета, в который входит класс) (рисунок 2.38). Имя класса в пакете должно быть уникальным. Класс реализует один или несколько интерфейсов.

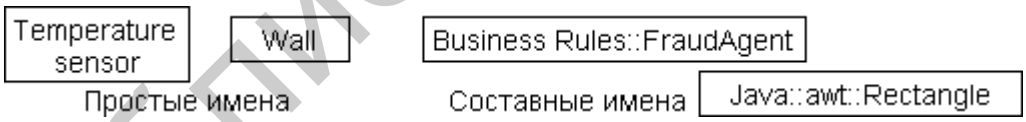


Рисунок 2.38 – Простые и составные имена

Атрибут – это именованное свойство класса, включающее описание множества значений, которые могут принимать экземпляры этого класса. Класс может иметь любое число атрибутов или не иметь их вовсе. В языках высокого уровня, таких, как C++, Java, атрибуты соответствуют переменным, объявленным в классе. Например, у любой стены есть высота, ширина и толщина. Атрибуты представлены в разделе, расположенном под именем класса; при этом указываются их имена, и иногда начальное значение (рисунок 2.39).

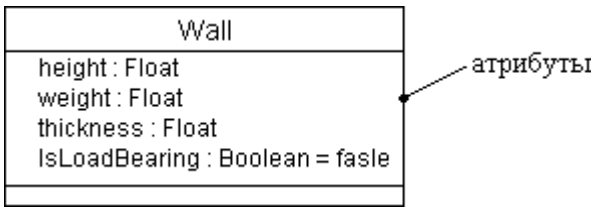


Рисунок 2.39 – Атрибуты и их класс

Операция – это некоторый сервис, который предоставляет экземпляр или объект класса по требованию своих клиентов (других объектов, в том числе и экземпляров данного класса). Класс может содержать любое число операций или не содержать их вовсе. В языках высокого уровня, таких, как C++, Java, операции соответствуют функциям, объявленным в классе. Операцию можно описать более подробно, указав имена и типы параметров, их значения, принятые по умолчанию, а также тип возвращаемого значения (рисунок 2.40).

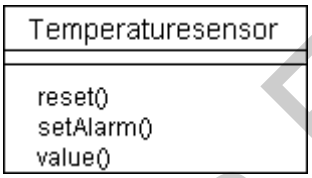


Рисунок 2.40 – Операции

При изображении класса необязательно сразу показывать все его атрибуты и операции. Их может быть много, однако для данного представления системы лишь небольшое подмножество атрибутов и операций имеет значение. В это случае класс сворачивают и изображают только некоторые из атрибутов и операций, а дополнительные атрибуты или операции обозначают многоточием. Для лучшей организации списков атрибутов и операций можно снабдить каждую группу дополнительным описанием (стереотипами).

Обязанности класса – это своего рода контракт, которому он должен подчиняться. Атрибуты и операции являются свойствами, посредством которых выполняются обязанности класса. Например, класс FraudAgent (агент по предотвращению мошенничества), который встречается в приложениях по обработке кредитных карточек, отвечает за оценку платежных требований – законные, подозрительные или подложные (рисунок 2.41). Моделирование классов лучше всего начинать с определения обязанностей сущностей, которые входят в словарь системы. Число обязанностей класса может быть произвольным, но на практике хорошо структурированный класс имеет по меньшей мере одну обязанность; с другой стороны, их не должно быть и слишком много. При уточнении модели обязанности класса преобразуются в совокупность атрибутов и операций, которые должны наилучшим образом обеспечить их выполнение.

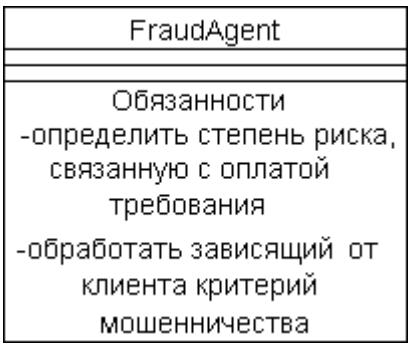


Рисунок 2.41 – Обязанности

Классы редко существуют автономно, они взаимодействуют между собой. Это значит, что при моделировании системы необходимо идентифицировать не только сущности, составляющие ее словарь, но и описать, как они соотносятся друг с другом. Существует три основных вида отношений между классами: *зависимости*, *обобщения, ассоциации* (рисунок 2.42).

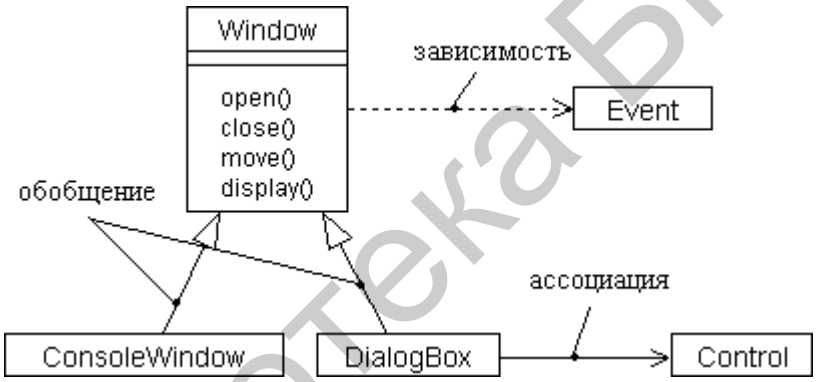


Рисунок 2.42 – Отношения

Кроме перечисленных отношений на диаграммах классов также применяются отношения агрегации и композиции.

Отношение агрегации применяется для представления системных взаимосвязей типа «часть – целое». Например, деление персонального компьютера на составные части: системный блок, монитор, клавиатуру и мышь (рисунок 2.43).

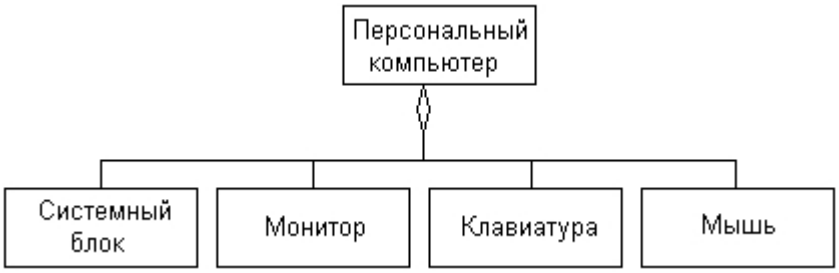


Рисунок 2.43 – Отношения агрегации

Отношение композиции является частным случаем агрегации. Оно служит для выделения специальной формы отношения «часть – целое», при которой составляющие части в некотором смысле находятся внутри целого. Специфика взаимосвязи между ними заключается в том, что части не могут выступать в отрыве от целого, т. е. с уничтожением целого уничтожаются и все его составные части. Например – окно интерфейса программы, состоящее из строки заголовка, кнопок управления размером, полос прокрутки, главного меню, рабочей области и строки состояния (рисунок 2.44).

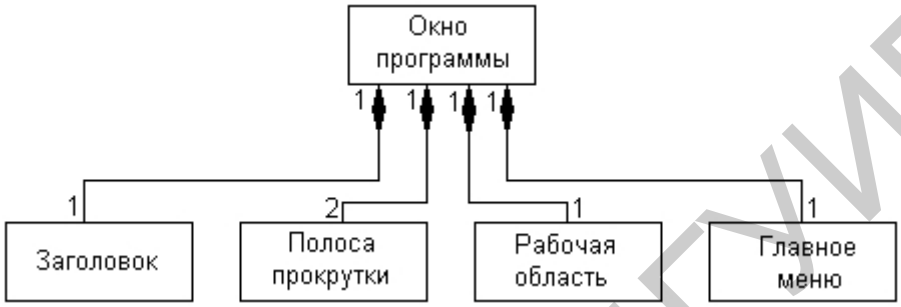


Рисунок 2.44 – Отношения композиции

Кроме описанных выше элементов на диаграмме классов также могут присутствовать примечания, дополнения, стереотипы, помеченные значения, ограничения.

Стереотипом называют расширение словаря UML, позволяющее создавать новые виды строительных блоков, аналогичные существующим, но специфичные для данной задачи. Стереотип представлен в виде имени, заключенного в кавычки и расположенного над именем другого элемента. С помощью стереотипа создается новый строительный блок, который напоминает существующий, но обладает новыми свойствами (рисунок 2.45).

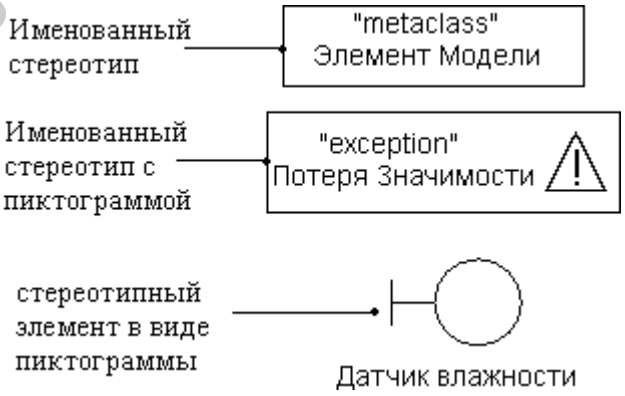


Рисунок 2.45 – Стереотипы

Помеченное значение позволяет включать новую информацию в спецификацию элемента. Например, при создании нескольких версий ПО необходимо отслеживать версию и автора какой-нибудь важной абстракции. Ни версия, ни автор не являются первичными концепциями UML, но их можно добавить к любому блоку, например к классу, задавая для него новые помеченные значения (рисунок 2.46).

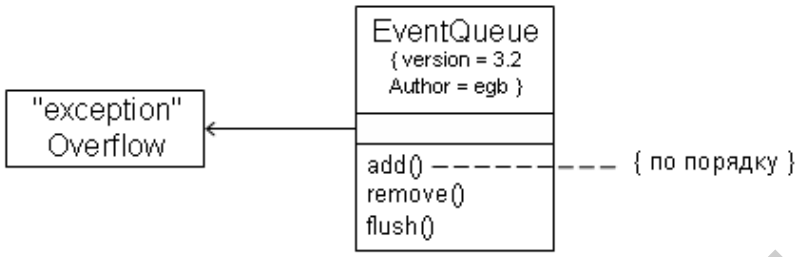


Рисунок 2.46 – Помеченное значение

Ограничение – это расширение семантики элемента UML, позволяющее создавать новые или изменять существующие правила (рисунок 2.47).

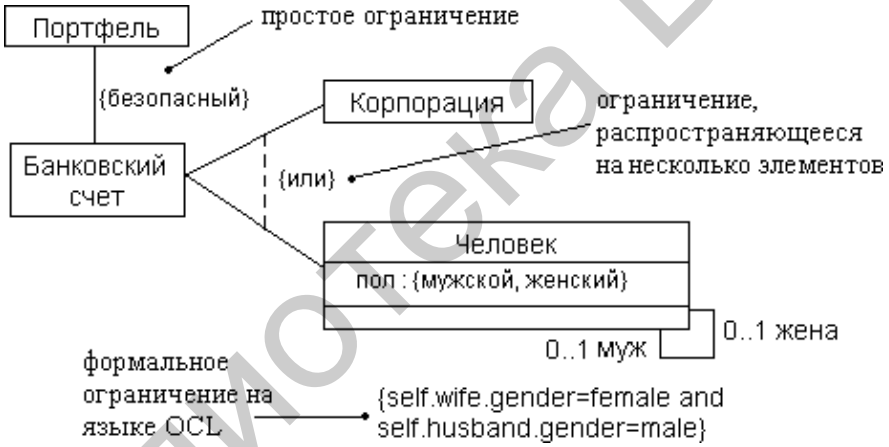


Рисунок 2.47 – Ограничения

Каждый элемент языка UML имеет свою семантику. С помощью ограничений можно создавать новую семантику или изменять существующие семантические правила. Например, на рисунке 2.47 показано, что информация, передаваемая между классами «портфель» и «банковский счет», зашифрована. При моделировании систем реального времени часто используются временные и пространственные ограничения.

Прямое и обратное проектирование. Основным результатом деятельности группы разработчиков являются не диаграммы, а программное обеспечение, поэтому модели и основанные на них реализации должны соответствовать друг другу с минимальными затратами по поддержанию синхронизации между ними. Чаще всего разработанные модели преобразуются в программный код. Хотя UML не определяет конкретного способа отображения на какой-либо объ-

ектно-ориентированный язык, он проектировался с учетом этого требования. В наибольшей степени это относится к диаграммам классов, содержание которых без труда отображается на такие известные объектно-ориентированные языки программирования, как Java, C++, ObjectPascal, Visual Basic и др.

Прямым проектированием (Forward engineering) называется процесс преобразования модели в код путем отображения на некоторый язык реализации. Процесс прямого проектирования приводит к потере информации, поскольку написанные на языке UML модели семантически богаче любого из существующих объектно-ориентированных языков. Фактически именно это различие и является основной причиной необходимости моделей. Некоторые структурные свойства системы, такие как кооперации или ее поведенческие особенности, например взаимодействия, могут быть легко визуализированы в UML, но в чистом коде наглядность теряется.

При прямом проектировании диаграммы классов следует учитывать, что так как модели зависят от семантики выбранного языка программирования, вероятно, придется отказаться от использования некоторых возможностей UML. Например, язык UML допускает множественное наследование, а язык программирования Smalltalk – только одиночное. В связи с этим можно запретить авторам моделей пользоваться множественным наследованием. Для специфицирования языка программирования применяются помеченные значения как на уровне индивидуальных классов (если нужна тонкая настройка), так и на более высоком уровне, например для пакетов или кооперации.

На рисунке 2.48 изображена диаграмма классов, на которой проиллюстрирована реализация образца цепочки обязанностей. В данном примере представлены три класса: Client (Клиент), EventHandler (ОбработчикСобытий) и GUIEventHandler (ОбработчикСобытийГИП). Первые два из них являются абстрактными, а последний – конкретным. Класс EventHandler содержит обычную для данного образца операцию handleRequest (ОбработатьЗапрос), хотя в рассматриваемом случае было добавлено два скрытых атрибута.

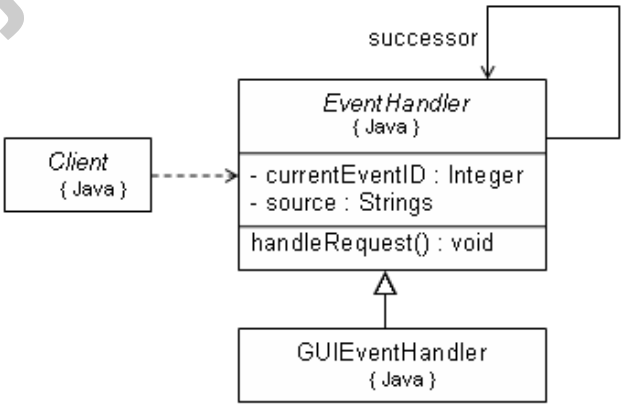


Рисунок 2.48 – Прямое проектирование

Определенное для каждого класса помеченное значение показывает, что они будут преобразованы в код на языке Java. Прямое проектирование в данном случае легко осуществимо с помощью специального инструмента. Так, для класса EventHandler будет сгенерирован следующий код:

```
public abstract class EventHandler {
  EventHandler successor;
  private Integer currentEventID;
  private String source;
  EventHandler()   {}
  public void handleRequest ()   {}
}
```

Обратным проектированием (Reverse engineering) называется процесс преобразования в модель кода, записанного на каком-либо языке программирования. В результате этого процесса вы получаете огромный объем информации, часть которой находится на более низком уровне детализации, чем необходимо для построения полезных моделей. В то же время обратное проектирование никогда не бывает полным. Как уже упоминалось, прямое проектирование ведет к потере информации, так что полностью восстановить модель на основе кода не удастся, если только инструментальные средства не включали в комментариях к исходному тексту информацию, выходящую за пределы семантики языка реализации.

Обратное проектирование диаграммы классов осуществляется следующим образом:

– идентифицируются правила для преобразования из выбранного языка реализации. Это можно сделать на уровне проекта или организации в целом;

– с помощью инструментального средства указывается код, который будет подвергнут обратному проектированию;

– используя инструментальные средства, создается диаграмма классов путем опроса полученной модели. Следует начать, например, с одного или нескольких классов, а затем расширить диаграмму, следуя вдоль некоторых отношений или добавив соседние классы. При этом можно раскрыть или спрятать детали содержания диаграммы в зависимости от ваших намерений.

Примеры диаграмм классов. На рисунке 2.49 показана совокупность классов, взятых из информационной системы вуза. Этот рисунок содержит достаточное количество деталей для конструирования физической базы данных. В нижней части диаграммы расположены классы «студент», «курс» и «преподаватель». Между классами «студент» и «курс» есть ассоциация, показывающая, что студенты могут посещать курсы. Более того, каждый студент может посещать любое количество курсов и на каждый курс может записаться любое число студентов.

Два класса «вуз» и «факультет» содержат несколько операций, позволяющих манипулировать их частями. Эти операции включены из-за их важности для поддержания целостности данных (например, добавление или удаление «факультета» затрагивает целый ряд других объектов).

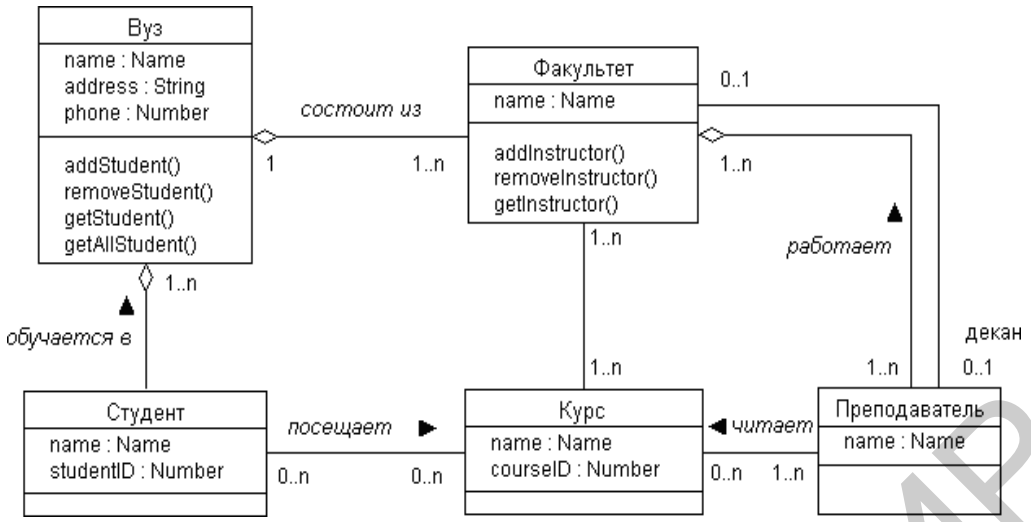


Рисунок 2.49 – Диаграмма классов для информационной системы вуза

Существует много других операций, которые стоило бы рассмотреть при проектировании этих и иных классов, например, запрос о необходимых предварительных знаниях перед зачислением студента на курс. Но это, скорее, бизнес-правила, а не операции по поддержанию целостности данных, поэтому лучше располагать их на более высоком уровне абстракции.

На рисунке 2.50 представлен пример диаграммы классов структуры компании.

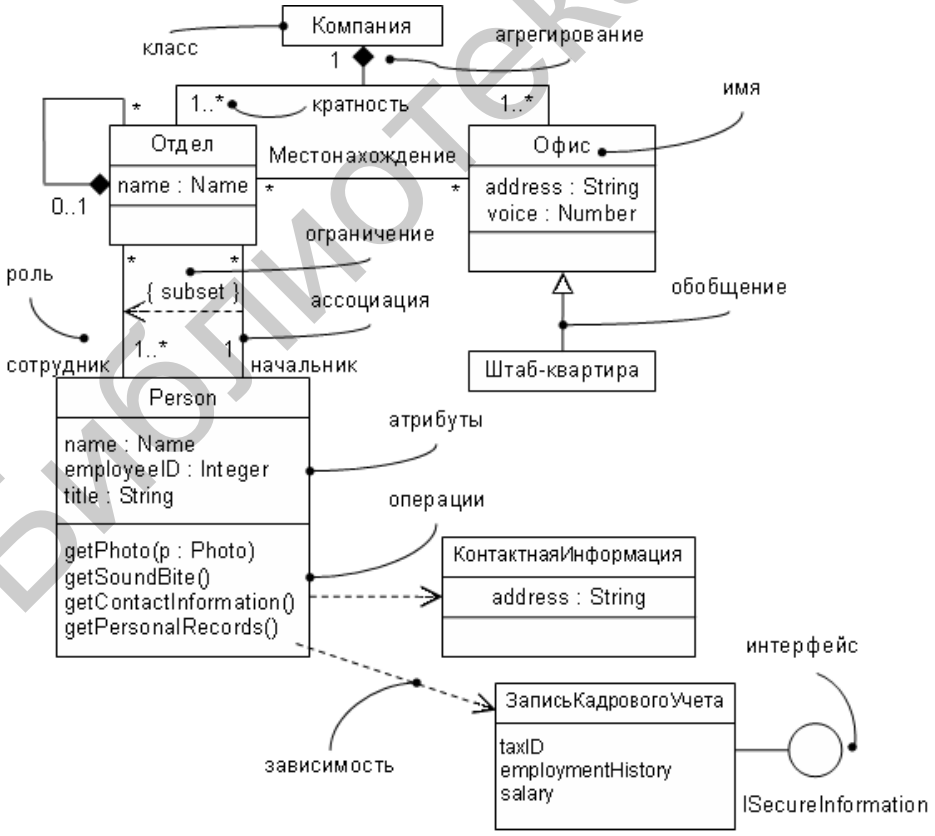


Рисунок 2.50 – Диаграмма классов

2.6 Диаграммы состояний (statechart diagram)

Главное предназначение этой диаграммы – описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Чаще всего диаграммы состояний используются для описания поведения отдельных экземпляров классов (объектов), но они также могут быть применены для спецификации функциональности других компонентов моделей, таких, как варианты использования, актеры, подсистемы, операции и методы.

Диаграмма состояний по существу является графом специального вида, который представляет некоторый автомат. Вершинами этого графа являются состояния и некоторые другие типы элементов автомата (псевдосостояния), которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Для понимания семантики конкретной диаграммы состояний необходимо представлять не только особенности поведения моделируемой сущности, но и знать общие сведения по теории автоматов.

Автомат (state machine) в языке UML представляет собой некоторый формализм для моделирования поведения элементов модели и системы в целом. Автомат описывает поведение отдельного объекта в форме последовательности состояний, которые охватывают все этапы его жизненного цикла, начиная от создания объекта и заканчивая его уничтожением. Каждая диаграмма состояний представляет некоторый автомат.

Простейшим примером визуального представления состояний и переходов на основе формализма автоматов может служить ситуация с исправностью технического устройства, такого как компьютер. В этом случае вводятся в рассмотрение два самых общих состояния: «исправен» и «неисправен» и два перехода: «выход из строя» и «ремонт». Графически эта информация может быть представлена в виде изображенной ниже диаграммы состояний компьютера (рисунок 2.51).

Рисунок 2.51 – Простейший пример диаграммы состояний для технического устройства типа компьютер

Основными понятиями, входящими в формализм автомата, являются состояние и переход. Главное различие между ними заключается в том, что длительность нахождения системы в отдельном состоянии существенно превышает время, которое затрачивается на переход из одного состояния в другое. Предполагается, что в пределе время перехода из одного состояния в другое равно нулю (если дополнительно ничего не сказано). Другими словами, переход объекта из состояния в состояние происходит мгновенно.

В языке UML под *состоянием* понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия. Состояние может быть задано в виде набора конкретных значений атрибутов класса или объекта, при этом изменение их отдельных значений будет отражать изменение состояния моделируемого класса или объекта.

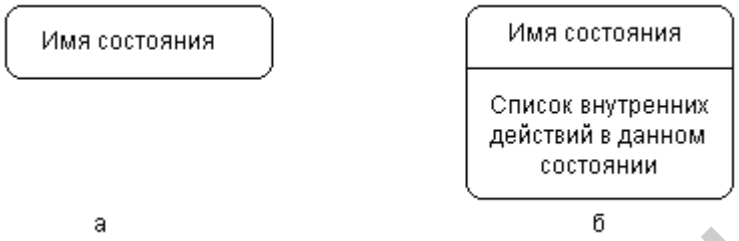


Рисунок 2.52 – Графическое изображение состояний на диаграмме состояний

Состояние на диаграмме изображается прямоугольником со скругленными вершинами (рисунок 2.52). Этот прямоугольник, в свою очередь, может быть разделен на две секции горизонтальной линией. Если указана лишь одна секция, то в ней записывается только имя состояния (рисунок 2.52, а). В противном случае в первой из них записывается имя состояния, а во второй – список некоторых внутренних действий или переходов в данном состоянии (рисунок 2.52, б).

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий. В этом состоянии находится объект по умолчанию в начальный момент времени. Графически начальное состояние в языке UML обозначается в виде закрашенного кружка (рисунок 2.53, а), из которого может только выходить стрелка, соответствующая переходу.

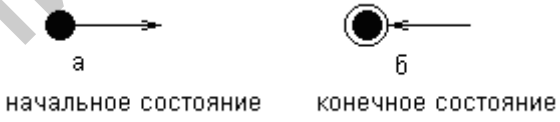


Рисунок 2.53 – Графическое изображение начального и конечного состояний на диаграмме состояний

Конечное (финальное) состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних действий. В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени. Графически конечное состояние в языке UML обозначается в виде закрашенного кружка, помещенного в окружность (рисунок 2.53, б), в которую может только входить стрелка, соответствующая переходу.

Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим. Пребывание моделируемого объекта в первом состоянии может сопровождаться выполнением некоторых действий, а переход во второе состояние будет возможен после завершения этих действий, а также после удовлетворения некоторых дополнительных условий. На диаграмме состояний переход изображается сплошной линией со стрелкой, которая направлена в целевое состояние (рисунок 2.54).

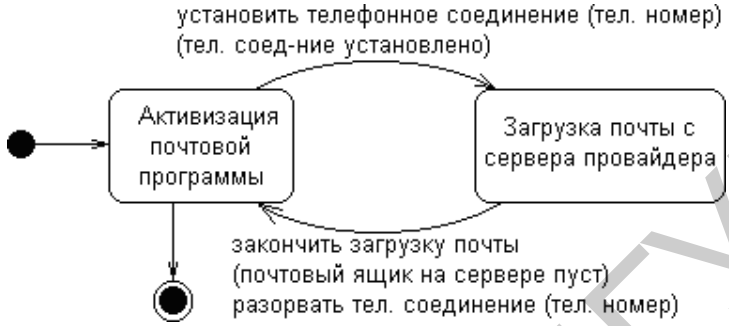


Рисунок 2.54 – Диаграмма состояний для моделирования почтовой программы-клиента

Пример диаграммы состояний. Рассмотрим пример диаграммы состояний для моделирования поведения банкомата (рисунок 2.55).

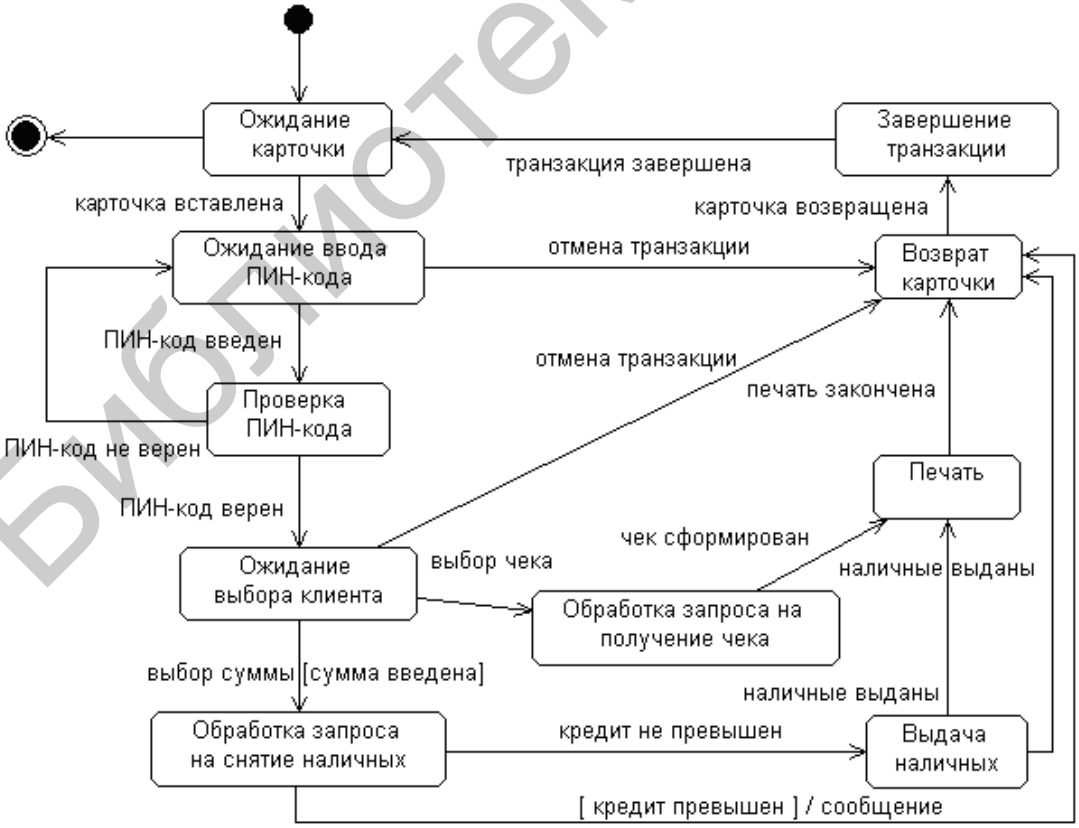


Рисунок 2.55 – Диаграмма состояний для моделирования поведения банкомата

Следует заметить, что в разрабатываемой модели диаграмма состояний является единственной и описывает поведение системы управления банкоматом в целом. Главное достоинство данной диаграммы состояний – возможность моделировать условный характер реализации всех вариантов использования в форме изменения отдельных состояний разрабатываемой системы. Иногда разработку диаграммы состояний, особенно в условиях дефицита времени, отпущенного на выполнение проекта, опускают, т. к. часто происходит дублирование информации, представленной на диаграммах кооперации и последовательности.

2.7 Диаграммы деятельности (activity diagram)

При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций.

Для моделирования процесса выполнения операций в языке UML используются так называемые *диаграммы деятельности*. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления не деятельностей, а действий, и в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами – переходы от одного состояния действия к другому.

В контексте языка UML деятельность (activity) представляет собой некоторую совокупность отдельных вычислений, выполняемых автоматом. При этом отдельные элементарные вычисления могут приводить к некоторому результату или действию (action). На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения.

Основные элементы диаграммы деятельности. Состояние действия (action state) является специальным случаем состояния с некоторым входным действием и по крайней мере одним выходящим из состояния переходом. Графически состояние действия изображается фигурой, напоминающей прямоугольник, боковые стороны которого заменены выпуклыми дугами (рисунок 2.56). Внутри этой фигуры записывается выражение действия (action-expression), которое должно быть уникальным в пределах одной диаграммы деятельности.

а б

а – простое действие; б – выражение

Рисунок 2.56 – Графическое изображение состояния действия

Действие может быть записано на естественном языке, некотором псевдокоде или языке программирования. Никаких дополнительных или неявных ограничений при записи действий не накладывается.

При построении диаграммы деятельности используются только те переходы, которые переводят деятельность в последующее состояние сразу, как только закончится действие в предыдущем состоянии (нетриггерные). На диаграмме такой переход изображается сплошной линией со стрелкой.

Ветвление на диаграмме деятельности обозначается небольшим ромбом, внутри которого нет никакого текста (рисунок 2.57).

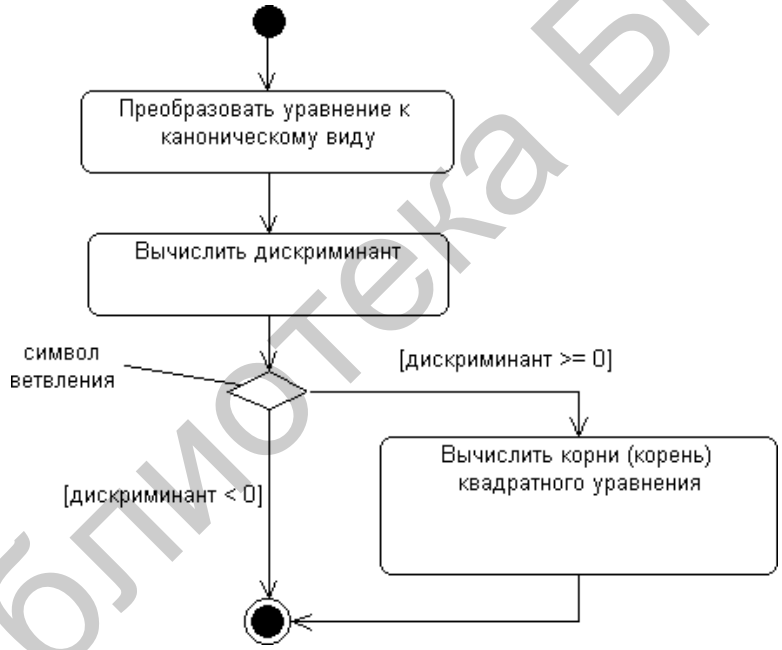
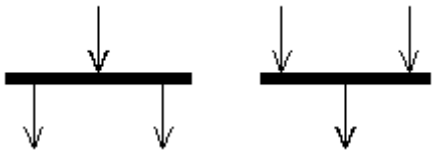


Рисунок 2.57 – Фрагмент диаграммы деятельности для алгоритма нахождения корней квадратного уравнения

В качестве примера рассмотрим фрагмент алгоритма нахождения корней квадратного уравнения. В общем случае после приведения уравнения второй степени к каноническому виду: $a*x*x + b*x + c = 0$ необходимо вычислить его дискриминант. Причем в случае отрицательного дискриминанта уравнение не имеет решения на множестве действительных чисел, и дальнейшие вычисления должны быть прекращены. При неотрицательном дискриминанте уравнение имеет решение, корни которого могут быть получены на основе конкретной расчетной формулы.

Процедуру вычисления корней квадратного уравнения можно представить в виде диаграммы деятельности с тремя состояниями действия и ветвлением (рисунок 2.57). Каждый из переходов, выходящих из состояния «Вычислить дискриминант», имеет сторожевое условие, определяющее единственную ветвь, по которой может быть продолжен процесс вычисления корней в зависимости от знака дискриминанта.

В языке UML для распараллеливания вычислений используется специальный символ для разделения (рисунок 2.58, а) и слияния (рисунок 2.58, б) параллельных вычислений или потоков управления.



а – раздел; б – слияние

Рисунок 2.58 – Разделения и слияния параллельных потоков управления

Диаграммы деятельности могут быть использованы не только для спецификации алгоритмов вычислений или потоков управления в программных системах. Не менее важная область их применения связана с моделированием бизнес-процессов. Для моделирования этих особенностей в языке UML используется специальная конструкция, получившая название *дорожки* (swimlanes). Имеется в виду визуальная аналогия с плавательными дорожками в бассейне, если смотреть на соответствующую диаграмму. При этом все состояния действия на диаграмме деятельности делятся на отдельные группы, которые отделяются друг от друга вертикальными линиями. Две соседние линии и образуют дорожку, а группа состояний между этими линиями выполняется отдельным подразделением (отделом, группой, отделением, филиалом) компании (рисунок 2.59).

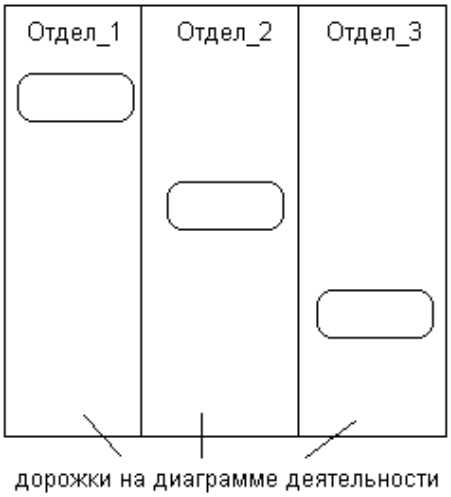


Рисунок 2.59 – Вариант диаграммы деятельности с дорожками

В общем случае действия на диаграмме деятельности выполняются над теми или иными объектами. Эти объекты либо инициируют выполнение действий, либо определяют некоторый результат этих действий. При этом действия специфицируют вызовы, которые передаются от одного объекта графа деятельности к другому. Поскольку в таком ракурсе объекты играют определенную роль в понимании процесса деятельности, иногда возникает необходимость явно указать их на диаграмме деятельности.

Для графического представления объектов используются прямоугольник класса, с тем отличием, что имя объекта подчеркивается. Далее после имени может указываться характеристика состояния объекта в прямых скобках. Такие прямоугольники объектов присоединяются к состояниям действия отношением зависимости пунктирной линией со стрелкой. Соответствующая зависимость определяет состояние конкретного объекта после выполнения предшествующего действия.

Пример диаграммы деятельности. В качестве примера рассмотрим фрагмент диаграммы деятельности торговой компании, обслуживающей клиентов по телефону. Подразделениями компании являются отдел приема и оформления заказов, отдел продаж и склад.

Этим подразделениям будут соответствовать три дорожки на диаграмме деятельности, каждая из которых специфицирует зону ответственности подразделения (рисунок 2.60).

В данном случае диаграмма деятельности заключает в себе не только информацию о последовательности выполнения рабочих действий, но и о том, какое из подразделений торговой компании должно выполнять то или иное действие. Кроме того, центральным объектом процесса продажи является заказ или, вернее, состояние его выполнения. Вначале до звонка от клиента заказ как объект отсутствует и возникает лишь после такого звонка. Однако этот заказ еще не заполнен до конца, поскольку требуется еще подобрать конкретный товар в отделе продаж. После его подготовки он передается на склад, где вместе с отпуском товара заказ окончательно дооформляется. Наконец, после получения подтверждения об оплате товара эта информация заносится в заказ, и он считается выполненным и закрытым.

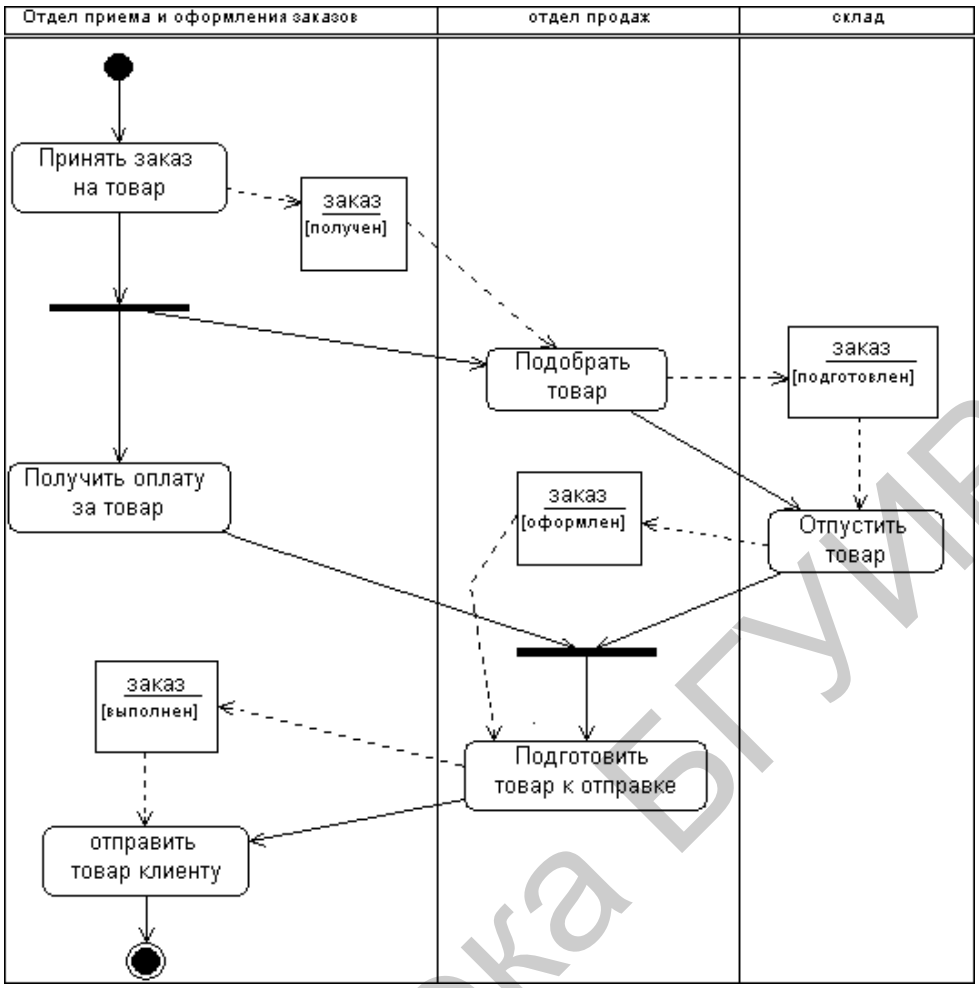


Рисунок 2.60 – Диаграмма деятельности торговой компании с объектом-заказом

2.8 Диаграммы компонентов (component diagram)

Все рассмотренные ранее диаграммы отражали концептуальные аспекты построения модели системы и относились к логическому уровню представления. *Диаграмма компонентов* описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

Диаграмма компонентов разрабатывается для следующих целей:

– визуализации общей структуры исходного кода программной системы;

– спецификации исполнимого варианта программной системы;

– обеспечения многократного использования отдельных фрагментов программного кода;

– представления концептуальной и физической схем баз данных.

Диаграмма компонентов обеспечивает согласованный переход от логического представления к конкретной реализации проекта в форме программного кода. Одни компоненты могут существовать только на этапе компиляции программного кода, другие – на этапе его исполнения. Диаграмма компонентов отражает общие зависимости между компонентами, рассматривая последние в качестве классификаторов.

Основные графические элементы диаграммы компонентов. Для представления физических сущностей в языке UML применяется специальный термин – *компонент* (component). Компонент реализует некоторый набор интерфейсов и служит для общего обозначения элементов физического представления модели. Для графического представления компонента может использоваться специальный символ – прямоугольник со вставленными слева двумя более мелкими прямоугольниками (рисунок 2.61). Внутри объемлющего прямоугольника записывается имя компонента и, возможно, некоторая дополнительная информация.

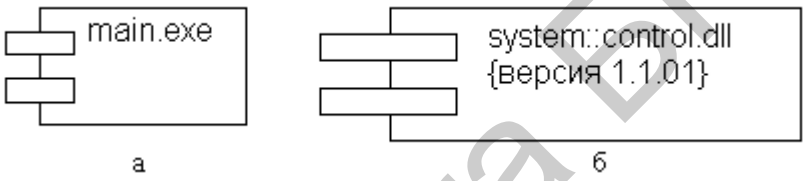


Рисунок 2.61 – Графическое изображение компонента в языке UML

В первом случае (рисунок 2.61, а) с компонентом уровня экземпляра связывается только его имя, а во втором (рисунок 2.61, б) – дополнительно имя пакета и помеченное значение.

В языке UML выделяют три вида компонентов:

– *компоненты развертывания*, которые обеспечивают непосредственное выполнение системой своих функций: динамически подключаемые библиотеки с расширением dll, web-страницы на языке разметки гипертекста с расширением html и файлы справки с расширением hlp.

– *компоненты – рабочие продукты*: файлы с исходными текстами программ, например, с расширениями h или cpp для языка C++;

– *компоненты исполнения*, представляющие исполнимые модули – файлы с расширением exe.

Следующим элементом диаграммы компонентов является *интерфейс*. Этот элемент уже рассматривался ранее, поэтому отметим только его особенности, которые характерны для представления на диаграммах компонентов. В общем случае интерфейс графически изображается окружностью, которая соединяется с компонентом отрезком линии без стрелок (рисунок 2.62, а). Семантически линия означает реализацию интерфейса, а наличие интерфейсов у ком-

понента означает, что данный компонент реализует соответствующий набор интерфейсов.

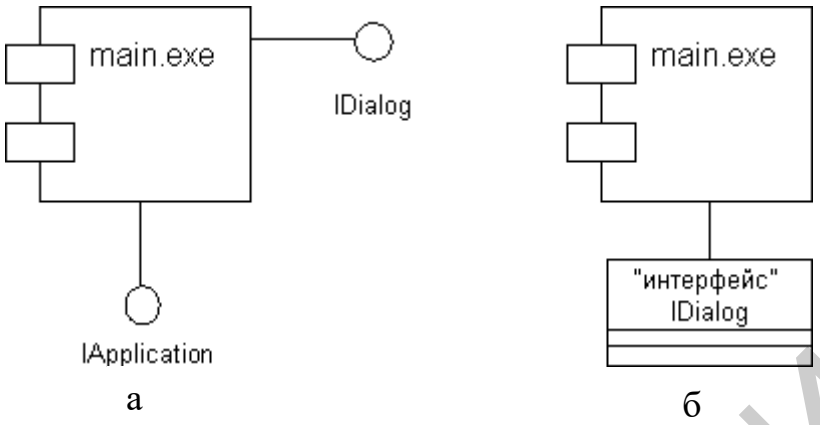


Рисунок 2.62 – Графическое изображение интерфейсов на диаграмме компонентов

Другим способом представления интерфейса на диаграмме компонентов является его изображение в виде прямоугольника класса со стереотипом «интерфейс» и возможными секциями атрибутов и операций (рисунок 2.62, б). Как правило, этот вариант обозначения используется для представления внутренней структуры интерфейса, которая может быть важна для реализации.

Применительно к диаграмме компонентов зависимости могут связывать компоненты и импортируемые этим компонентом интерфейсы, а также различные виды компонентов между собой.

В первом случае рисуют стрелку от компонента-клиента к импортируемому интерфейсу (рисунок 2.63). Наличие такой стрелки означает, что компонент не реализует соответствующий интерфейс, а использует его в процессе своего выполнения. Причем на этой же диаграмме может присутствовать и другой компонент, который реализует этот интерфейс. Так, например, изображенный ниже фрагмент диаграммы компонентов представляет информацию о том, что компонент с именем «main.exe» зависит от импортируемого интерфейса IDialog, который, в свою очередь, реализуется компонентом с именем «image.java». Для второго компонента этот же интерфейс является экспортируемым.

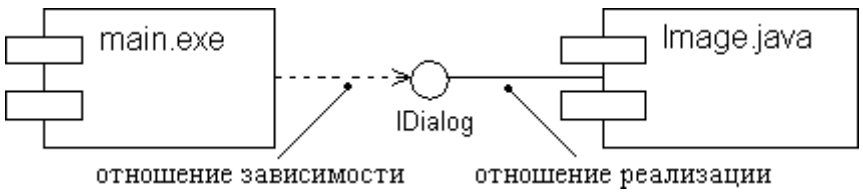


Рисунок 2.63 – Фрагмент диаграммы компонентов с отношением зависимости

На диаграмме компонентов также могут быть представлены отношения зависимости между компонентами и реализованными в них классами (рисунок 2.64). Эта информация имеет важное значение для обеспечения согласования логического и физического представлений модели системы.

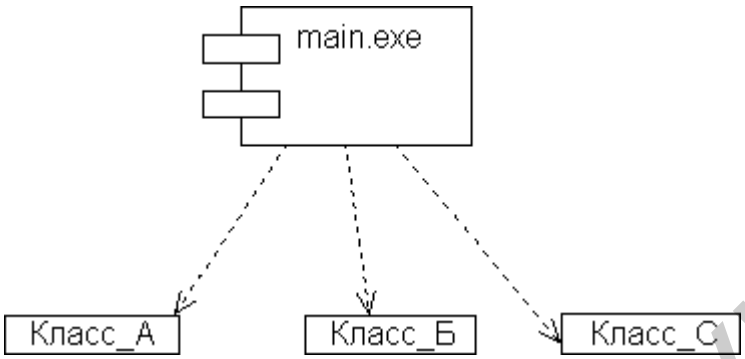


Рисунок 2.64 – Графическое изображение зависимости между компонентом и классами

2.9 Диаграммы развертывания (deployment diagram)

Физическое представление программной системы не может быть полным, если отсутствует информация о том, на какой платформе и на каких вычислительных средствах она реализована. *Диаграмма развертывания* предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения (runtime). При этом представляются только компоненты-экземпляры программы, являющиеся исполнимыми файлами или динамическими библиотеками. Те компоненты, которые не используются на этапе исполнения, на диаграмме развертывания не показываются. Так, компоненты с исходными текстами программ могут присутствовать только на диаграмме компонентов. На диаграмме развертывания они не указываются.

Диаграмма развертывания содержит графические изображения процессоров, устройств, процессов и связей между ними. В отличие от диаграмм логического представления диаграмма развертывания является единой для системы в целом, поскольку должна всецело отражать особенности ее реализации. Эта диаграмма, по сути, завершает процесс ООАП для конкретной программной системы и ее разработка, как правило, является последним этапом спецификации модели.

Элементы диаграммы развертывания. К основным элементам диаграммы развертывания относятся узлы и соединения.

Узел (node) представляет собой некоторый физически существующий элемент системы, обладающий некоторым вычислительным ресурсом. В качестве вычислительного ресурса узла может рассматриваться наличие по меньшей мере некоторого объема электронной или магнитооптической памяти

и/или процессора. Понятие узла также может включать в себя и другие механические или электронные устройства, такие, как датчики, принтеры, модемы, цифровые камеры, сканеры и манипуляторы.

Графически на диаграмме развертывания узел изображается в форме трехмерного куба. Узел имеет собственное имя, которое указывается внутри этого графического символа. Сами узлы могут представляться как в качестве типов (рисунок 2.65, а), так и в качестве экземпляров (рисунок 2.65, б).

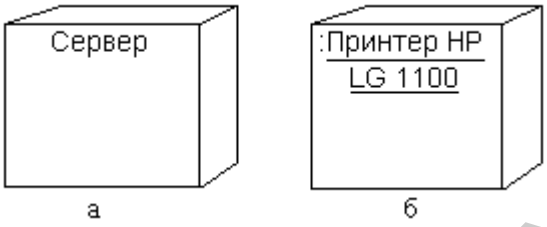


Рисунок 2.65 – Графическое изображение узла на диаграмме развертывания

Помеченное значение – это расширение свойств элемента UML, позволяющее вводить новую информацию в его спецификацию. У каждой сущности в UML есть фиксированный набор свойств: классы имеют имена, атрибуты и операции; ассоциации-имена и концевые точки (каждая со своими свойствами) и т. д. Помеченные значения позволяют добавлять новые свойства.

Например, как показано на рисунке 2.66, в диаграмме развертывания можно указать число процессоров, установленных на узле каждого вида, или потребовать, чтобы каждому компоненту был приписан стереотип библиотеки, если его предполагается развернуть на клиенте или сервере.

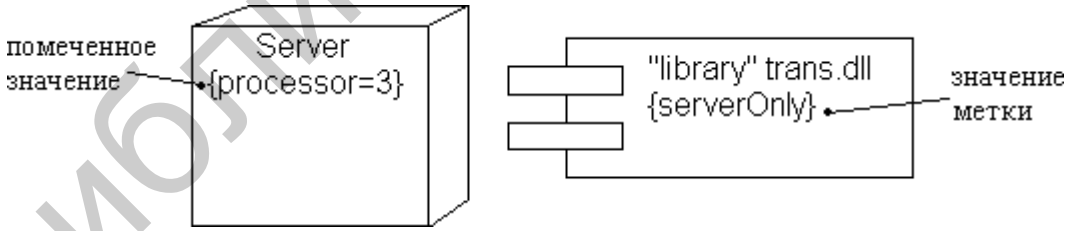


Рисунок 2.66 – Помеченные значения

Так же как и на диаграмме компонентов, изображения узлов могут расширяться, чтобы включить некоторую дополнительную информацию о спецификации узла. Если дополнительная информация относится к имени узла, то она записывается под этим именем в форме помеченного значения (рисунок 2.67).

Рисунок 2.67 – Графическое изображение узла-экземпляра с дополнительной информацией в форме помеченного значения

Соединения указывают отношения между узлами и являются разновидностью ассоциации. Изображаются отрезками линий без стрелок. Наличие такой линии указывает на необходимость организации физического канала для обмена информацией между соответствующими узлами. Характер соединения может быть дополнительно специфицирован примечанием, помеченным значением или ограничением (рисунок 2.68). В рассмотренном примере явно определены не только требования к скорости передачи данных в локальной сети с помощью помеченного значения, но и рекомендации по технологии физической реализации соединений в форме примечания.

Рисунок 2.68 – Фрагмент диаграммы развертывания с соединениями между узлами

Кроме соединений на диаграмме развертывания могут присутствовать *отношения зависимости* между узлом и развернутыми на нем компонентами. Подобный способ является альтернативой вложенному изображению компонентов внутри символа узла, что не всегда удобно, поскольку делает этот символ излишне объемным (рисунок 2.69).

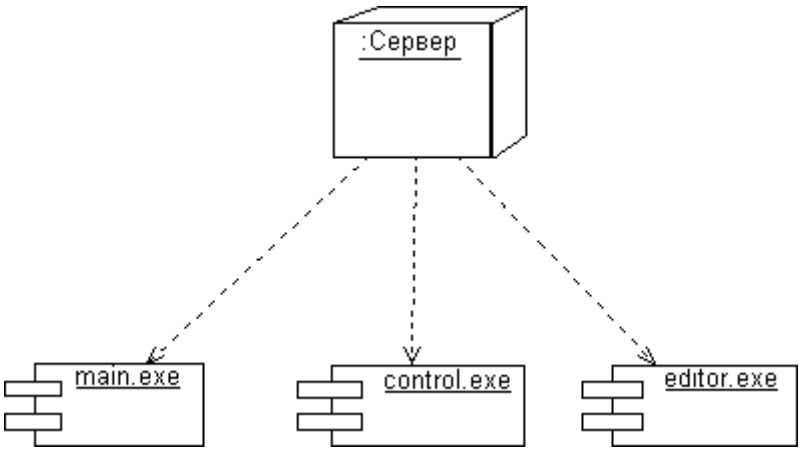


Рисунок 2.69 – Диаграмма развертывания с отношением зависимости между узлом и развернутыми на нем компонентами

Пример диаграммы развертывания. Рассмотрим фрагмент физического представления системы удаленного обслуживания клиентов банка (рисунок 2.70).

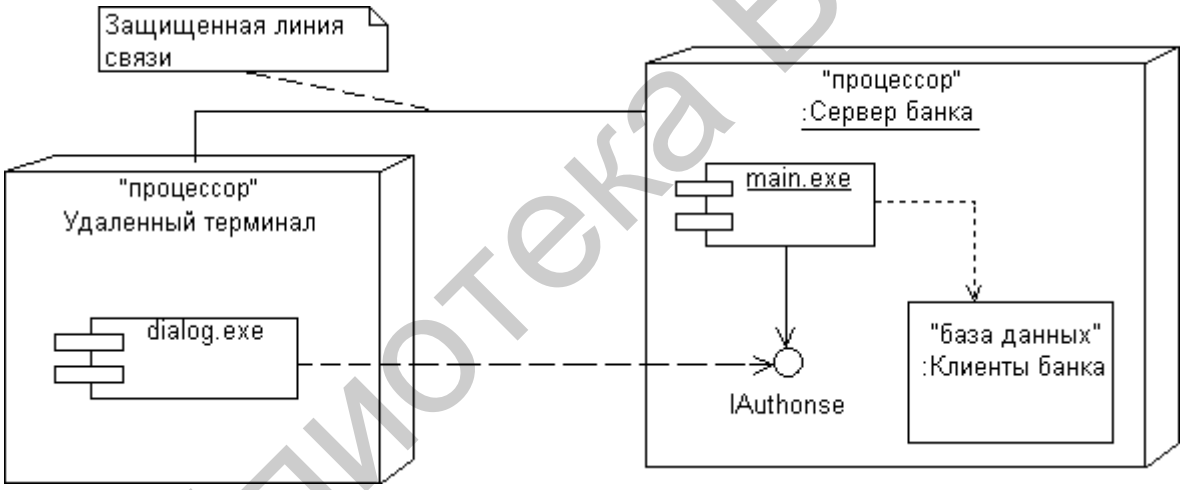


Рисунок 2.70 – Диаграмма развертывания для системы удаленного обслуживания клиентов банка

На диаграмме развертывания узлами системы являются удаленный терминал (узел-тип) и сервер банка (узел-экземпляр). Указана зависимость компонента реализации диалога «dialog.exe» на удаленном терминале от интерфейса lAuthorise, реализованного компонентом «main.exe», который, в свою очередь, развернут на анонимном узле-экземпляре «Сервер банка». Последний зависит от компонента базы данных «Клиенты банка», который развернут на этом же узле. Примечание указывает на необходимость использования защищенной линии связи для обмена данными в данной системе. Другой вариант записи этой информации заключается в дополнении диаграммы узлом со стереотипом «закрытая сеть».

ЗАКЛЮЧЕНИЕ

Пособие охватывает наиболее важные вопросы разработки программного обеспечения современных систем управления.

Основное внимание уделяется изучению методики анализа и проектирования программных систем на основе построения визуальных моделей в рамках унифицированного процесса разработки программного продукта (The Unified Software Development Process) с использованием унифицированного языка моделирования UML (The Unified Modeling Language).

Надеемся, что весьма сжатые сведения в учебно-методическом пособии являются достаточными для того, чтобы без затруднений перейти к более глубокому изучению соответствующих тем в опубликованных источниках, например, перечисленных ниже в списке литературы.

Пособие может быть использовано для индивидуального изучения и самостоятельного выполнения индивидуального задания по контрольной работе.

СПИСОК ЛИТЕРАТУРЫ

1 Трофимов, С. А. CASE-технологии: практическая работа в Rational Rose / С. А. Трофимов. – М. : Бином-Пресс, 2002. – 288 с.

2 Леоненков, А. Самоучитель UML / А. Леоненков. – СПб. : Питер, 2004. – 278 с.

3 Ноутон, П. Java 2 / П. Ноутон, Г. Шилдт. – СПб. : БХВ-Петербург, 2008. – 1072 с.

4 Пауэлл, Т. А. Полное руководство по HTML / Т. А. Пауэлл. – Минск : ООО «Попурри», 2001. – 576 с.

5 Фаулер, М. UML. Основы. Краткое руководство по унифицированному языку моделирования / М. Фаулер, К. Скотт. – СПб. : Символ-Плюс, 2005. – 192 с.

6 UML. Руководство пользователя / Г. Буч [и др.]. – М. : ДМК-Пресс, 2007. – 496 с.

7 CASE-технологии. Практикум / Д. Э. Федотова [и др.]. – М. : Горячая Линия – Телеком, 2005. – 160 с.

8 Эккель, Б. Философия Java / Б. Эккель. – СПб. : Питер, 2009. – 640 с.

9 Вагнер, Р. JavaScript. Энциклопедия пользователя / Р. Вагнер, А. Вайк. – М. : ДиаСофт, 2007. – 264 с.

10 Ноутон, П. Java 2 / П. Ноутон, Г. Шилдт. – СПб. : БХВ-Петербург, 2008. – 1072 с.

11 Мейнджер, Д. JAVA : основы программирования / Д. Мейнджер. – СПб. : BHV-Санкт-Петербург, 1997. – 320 с.

12 Буч, Гр. UML. Руководство пользователя / Гр. Буч, Дж. Рамбо, А. Джекобсон. – М. : ДМК, 2001. – 432 с.

13 Стелтинг, С. Применение шаблонов Java. Библиотека профессионалов/ С. Стелтинг, О. Маасен. – М. : Вильямс, 2002. – 576 с.

14 Спейнаур, С. Справочник web-мастера / С. Спейнаур, В. Куэрсиа ; пер. с англ. – Киев : BHV, 1997. – 386 с.

15 Вендров, А. М. CASE-технологии. Современные методы и средства проектирования информационных систем / А. М. Вендров. – М. : Финансы и статистика, 1998. – 176 с.

16 Пирумян, В. Платформа программирования J2ME для портативных устройств / В. Пирумян. – М. : КУДИЦ-Образ, 2002. – 352 с.

17 Маклаков, С. В. ERWin и BPWin. CASE-средства разработки информационных систем / С. В. Маклаков. – М. : Диалог-МИФИ, 2000. – 256 с.

18 Яворски, Дж. Подготовка web-страниц для Internet / Дж. Яворски. – М. : Лори, 1996. – 320 с.

Св. план 2013, поз. 20

Учебное издание

Городко Сергей Иванович
Снисаренко Светлана Валерьевна

СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *М. В. Касабуцкий*

Подписано в печать 30.10.2017. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 4,07. Уч-изд. л. 4,0. Тираж 100 экз. Заказ 10.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6