2.7.6. Диаграммы Состояний

Диаграмма Состояний (State Transition) содержит информацию о состояниях, в которых может находиться объект, о том, как он переходит из одного состояния в другое и каким образом он ведет себя в этих состояниях. На диаграмме Состояний отображают жизненный цикл одного объекта, начиная с момента его создания и заканчивая разрушением. С помощью таких диаграмм удобно моделировать динамику поведения класса. Диаграммы Состояний не нужно создавать для каждого класса, они применяются только в очень сложных случаях. Если объект класса может существовать в нескольких состояниях и в каждом из них ведет себя по-разному, тогда для него требуется такая диаграмма. Однако во многих проектах они вообще не используются.

Состоянием (state) называется одно из возможных условий, в которых может существовать объект. Находясь в конкретном состоянии, объект может выполнять определенные действия. Например, он может генерировать отчет, осуществлять некоторые вычисления или посылать сообщение другому объекту. Для выявления состояний объекта необходимо исследовать две области модели: значения атрибутов объекта и связи с другими объектами.

В нотации UML состояние изображают в виде прямоугольника с закругленными краями. На рис. 2.57 можно видеть следующие состояния: Открыт, Превышен счет, Закрыт.

С состоянием можно связать данные пяти типов: деятельность, входное действие, выходное действие, событие и история состояния. Рассмотрим каждый из них в контексте примера. На рис. 2.57 показана диаграмма Состояний для класса Account (Счет) системы ATM.

Деятельностью (activity) называется поведение, реализуемое объектом, когда он находится в определенном состоянии. Деятельность - это прерываемое поведение. Оно может выполняться до своего завершения, если объект находится в данном состоянии, или может быть прервано переходом объекта в другое состояние. Деятельность изображают внутри самого состояния, ей должны предшествовать слово do (делать) и двоеточие. На рис. 2.77 показано, что у состояния «Превышен счет» имеется деятельность «Послать уведомление клиенту».

Входным действием (entry action) называется поведение, которое выполняется, когда объект переходит в определенное состояние. Оно осуществляется не после того, как объект перешел в состояние, а является частью перехода. В отличие от деятельности, данное действие рассматривается как непрерываемое. Входное действие показывают внутри состояния, ему предшествуют слово entry (вход) и двоеточие. Например, при переходе объекта Счет в состояние «Превышен счет» (рис.2.57), выполняется действие «Вре-

менно заморозить счет» независимо от того, откуда объект переходит в это состояние.

Выходное действие (exit action) осуществляется как составная часть процесса выхода из состояния. Оно является частью процесса перехода. Как и входное, выходное действие является непрерываемым. Выходное действие изображают внутри состояния, ему предшествуют слово exit (выход) и двоеточие. При выходе объекта Счет из состояния «Превышен счет» (рис 2.57), независимо от того, куда он переходит, выполняется действие «Разморозить счет».

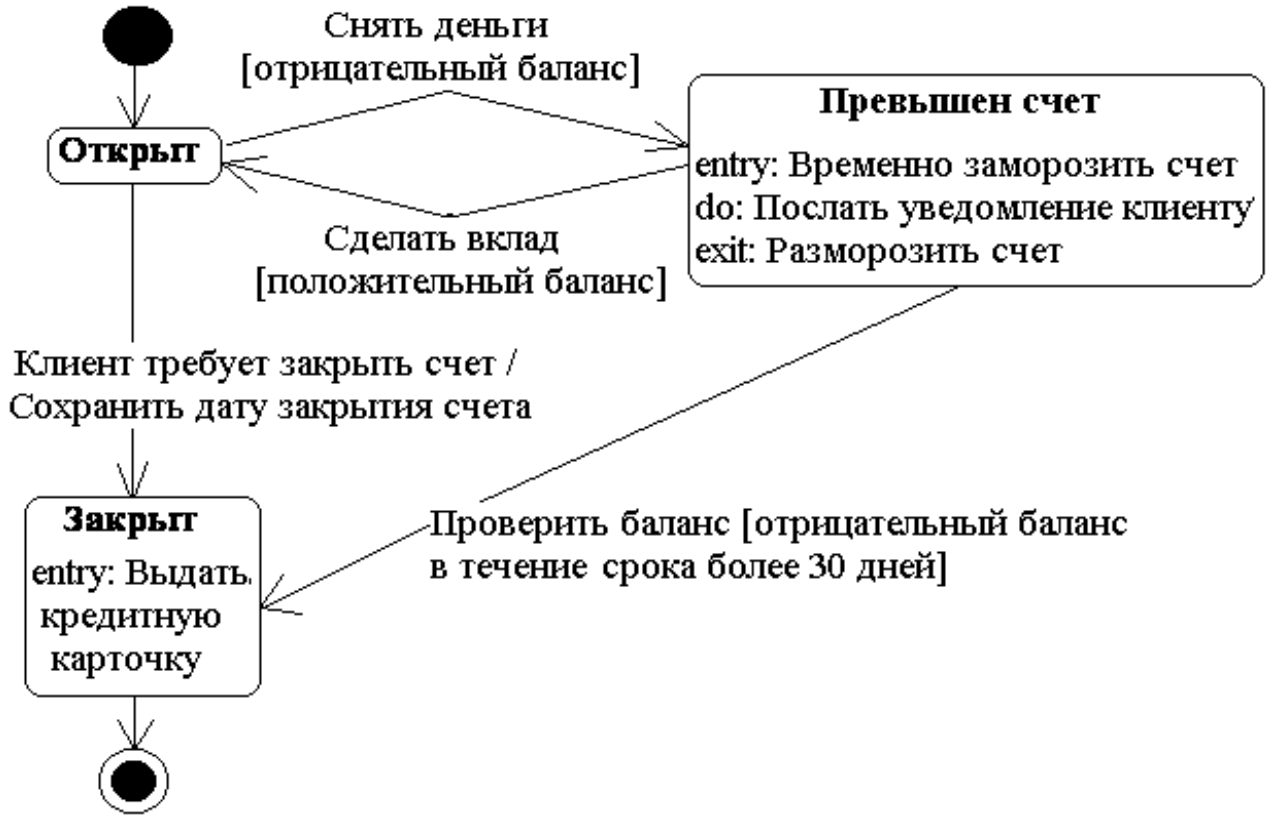


Рис. 2.57. Диаграмма Состояний для класса Account (Счет) системы ATM

Переходом (transition) называется перемещение объекта из одного состояния в другое. На диаграмме переходы изображают в виде стрелки, начинающейся в первоначальном и заканчивающейся в последующем состоянии. Переходы могут быть рефлексивными: объект переходит в то же состояние, в котором он в настоящий момент находится. Рефлексивные переходы изображают в виде стрелки, начинающейся и завершающейся на одном и том же состоянии. У перехода существует несколько спецификаций: события, аргументы, ограждающие условия, действия и посылаемые события. Рассмотрим эти

параметры в контексте примера АТМ (рис. 2.57).

Событие (event) - это то, что вызывает переход из одного состояния в другое. В нашем примере событие «Клиент требует закрыть счет» вызывает переход счета из открытого состояния в закрытое. Событие размещают на диаграмме вдоль линии перехода. Для отображения события на диаграмме можно использовать как имя операции, так и обычную фразу. Если использовать операции, то событие «Клиент требует закрыть счет» можно было бы назвать RequestClosure( ). У событий могут быть аргументы. Так, событие «Сделать вклад», вызывающее переход счета из состояния «Превышен счет» в состояние «Открыт», может иметь аргумент Amount (Количество), описывающий сумму депозита.

Большинство переходов должно иметь события, так как именно они инициируют переход. Тем не менее, бывают и ***автоматические переходы***, не имеющие событий. При этом объект сам перемещается из одного состояния в другое со скоростью, предусматривающей выполнение входных действий деятельности и выходных действий. В этом случае на диаграмме вдоль линии перехода нет никаких событий.

Ограждающее условие (guard conditions) определяет, когда переход может быть выполнен, а когда нет. На диаграмме ограждающие условия заключают в квадратные скобки и размещают, вдоль линии перехода после имени события. В нашем примере событие «Сделать вклад» переведет счет из состояния «Превышение счета» в состояние «Открыт», только если баланс больше нуля. В противном случае переход не осуществится. Ограждающие условия задавать необязательно. Однако если существует несколько автоматических переходов из состояния, необходимо определить для них взаимно исключающие ограждающие условия. Это поможет читателю диаграммы понять, какой путь перехода будет выбран автоматически.

Действием (action) является непрерываемое поведение, выполняющееся как часть перехода. Входные и выходные действия показывают внутри состояния, поскольку они определяют, что происходит, когда объект входит или выходит из состояния. Другие действия изображают вдоль линии перехода, так как они не должны осуществляться при входе или выходе из состояния. Действие размещают вдоль линии перехода после имени события, ему предшествует косая черта (/). Например, при переходе счета из открытого в закрытое состояние выполняется действие «Сохранить дату закрытия счета». Это непрерываемое поведение осуществляется только во время перехода из состояния «Открыт» в состояние «Закрыт».

Поведение объекта во время деятельности, входных и выходных действий может включать в себя отправку события другому объекту. Например, объект Account (Счет) может посылать событие объекту Card reader (Устрой-

ство чтения карты). В этом случае описанию деятельности, входного или выходного действия предшествует знак «^» (рис. 2.58). Здесь цель - это объект, получающий событие, событие - посылаемое сообщение, а аргументы являются параметрами посылаемого сообщения. При получении объектом события будет выполнена определенная деятельность.

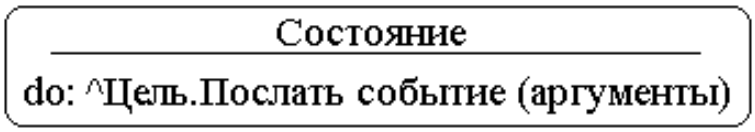


Рис. 2.58. Событие, посылаемое другому объекту

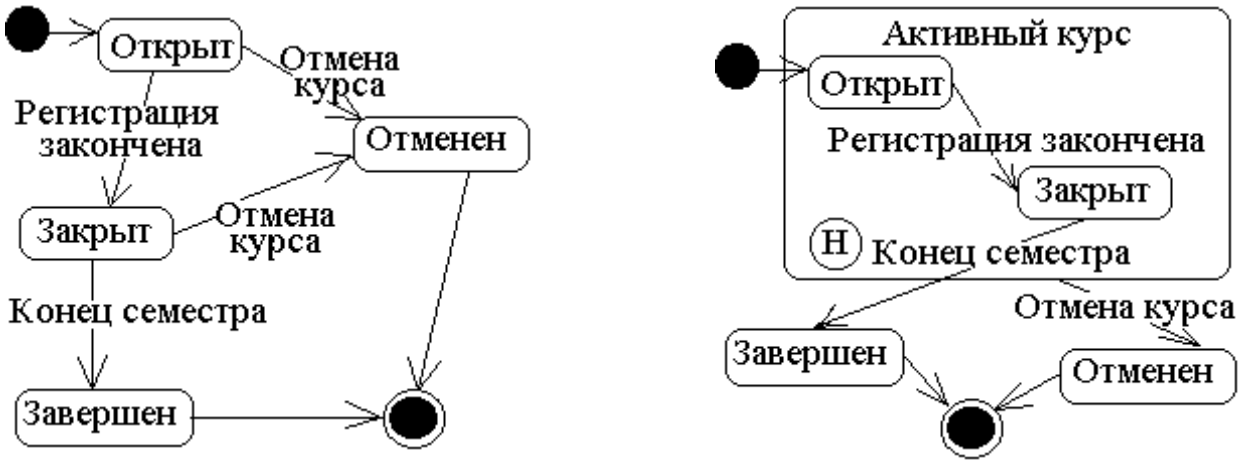
На диаграмму добавляют два специальных состояния объекта: начальное и конечное. ***Начальным*** (start) называется состояние, в котором объект находится сразу после своего создания. Начальное состояние обязательно - читатель должен знать, с чего начинается объект. На диаграмме может быть только одно начальное состояние, его изображают в виде закрашенного кружка. На рис. 2.57 оно показано до состояния «Открыт». ***Конечным*** (stop) называется состояние, в котором объект находится непосредственно перед уничтожением. Конечные состояния не являются обязательными, их может быть сколько угодно. Конечное состояние изображают в виде закрашенного кружка с ободком («бычий глаз»). На рис. 2.57 оно показано после состояния «Закрыт».

Для уменьшения беспорядка на диаграмме можно вкладывать состояния одно в другое. Вложенные состояния называются ***подсостояниями*** (substates), а те, в которые они вложены, - ***суперсостояниями*** (superstates).

Если у нескольких состояний имеются идентичные переходы, эти состояния можно сгруппировать вместе в суперсостояние. Затем, вместо того чтобы поддерживать одинаковые переходы (по одному на каждое состояние), их можно объединить, перенеся в суперсостояние. На рис. 2.59, а приведен пример диаграммы без вложенных состояний. На рис. 2.59, б изображена та же диаграмма с использованием вложенных состояний. Суперсостояния позволяют «навести порядок» на диаграмме Состояний.

Бывают случаи, когда система должна помнить, в каких состояниях она была в прошлом. Если, например, выход из суперсостояния с тремя подсостояниями, может потребовать, чтобы система запомнила, из какой точки суперсостояния произошел выход. Существует два способа решения этой проблемы. Во-первых, можно включить в суперсостояние начальное состояние. В таком случае будет понятно, где находится стартовая точка в суперсостоянии. Именно там окажется объект при входе в суперсостояние. Во-вторых, чтобы

запомнить, где был объект, можно использовать историю состояний (state history). В таком случае объект может выйти из суперсостояния, а затем вернуться точно в то место, откуда вышел. При подключении истории состояний в углу диаграммы располагается буква «H» в кружке (рис.2.60).



а) б)

Рис. 2.59. Диаграмма Состояний:
а) без вложенных состояний; б) суперсостояния позволили «внести порядок»

Рис. 2.60. История суперсостояния

2.7.7. Диаграммы Компонентов

Диаграммы Компонентов показывают, как выглядит модель на физическом уровне. На ней изображаются компоненты программного обеспечения системы и связи между ними.

Компонентом (component) называется физический модуль кода. Существуют два основных типа компонентов: библиотеки исходного кода и исполняемые компоненты. Например, для языка C++ файлы с расширением «cpp» и «h» будут отдельными компонентами. Получающийся при компиляции исполняемый файл с расширением «exe» также является компонентом системы.

Компонент (Component) соответствует программному модулю с хорошо определенным интерфейсом. Ниже обсуждаются различные стереотипы компонентов.

Спецификация и тело подпрограммы (Subprogram Specification and Body) представляют видимую спецификацию подпрограммы и тело ее реализации (рис. 2.61, а, б). Обычно подпрограмма состоит из стандартных программных компонентов (subroutines) и не содержит определений класса.

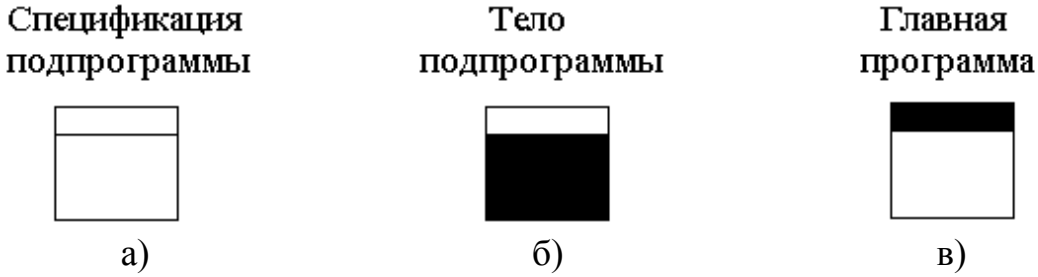


Рис. 2.61. Графическая нотация для изображения:
а) спецификации подпрограммы; б) тела подпрограммы; в) главной программы

Главная программа (Main Program) - это файл, содержащий корень программы (рис. 2.61, в). Например, в среде PowerBuilder такой файл содержит объект приложения.

Спецификация и тело пакета (Package Specification and Body) изображаются с помощью нотации показанной на рис. 2.62, а и б. Пакет в данном случае - это реализация класса. Спецификацией пакета является заголовочный файл со сведениями о прототипах функций для класса. На C++ это файл с расширением «h». Тело пакета содержит код операций класса. На C++ это файл с расширением «cpp».

Исполняемые компоненты - это исполняемые файлы, файлы DLL и задачи.

Файл динамической библиотеки (файл DLL) изображается с помощью нотации показанной на рис. 2.62, в.

Спецификация и тело задачи (Task Specification and Body) отображают пакеты, имеющие независимые потоки управления (рис. 2.62, г, д). Испол-

няемый файл обычно представляют как спецификацию задачи с расширением «exe».

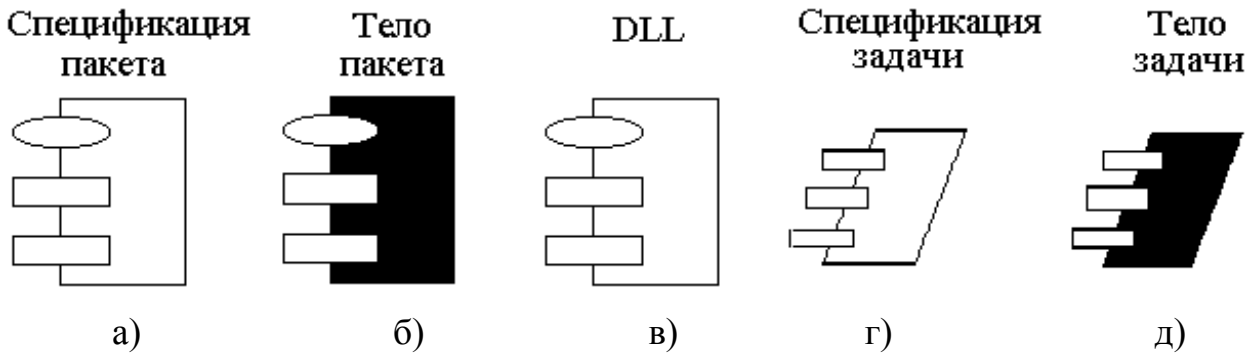


Рис.2. 62. Графическая нотация для изображения:
а) спецификации пакета; б) тела пакета; в) DLL; г) спецификации задачи; д) тела задачи

Единственный возможный тип связей между компонентами - это зависимость. Он показывает, что один из компонентов должен компилироваться перед началом компиляции другого. Зависимость между компонентами изображают пунктирной линией.

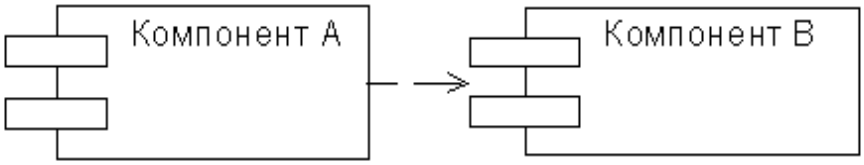


Рис. 2.63. Графическое представление связи между компонентами

На рис. 2.63 показан пример зависимости. В этом случае компонент А зависит от В. Это может означать, что в компоненте А существует некоторый класс, зависящий от какого-то класса компонента В. Знание о зависимостях важны при компиляции. Так как А зависит от В, А не может быть скомпилирован до В. Анализируя эту диаграмму, можно понять, что сначала компилируется В, а затем уже А. Следует избегать циклических зависимостей между компонентами. Если А зависит от В, а В от А, то ни один из них нельзя компилировать, пока не скомпилирован другой. Таким образом, оба компонента должны рассматриваться как один большой компонент. Все циклические зависимости необходимо устранить до начала генерации кода.

Зависимости связаны также с проблемами управления системой. Если А зависит от В, то любые изменения в В повлияют на А. С помощью диаграммы Компонентов персонал сопровождения системы может оценить последствия

вносимых изменений. Если компонент зависит от большего числа других компонентов, велика вероятность того, что его затронут изменения в системе.

Наконец, зависимости дают возможность понять, какие части системы можно использовать повторно, а какие нельзя. В нашем примере А трудно будет применить второй раз. Поскольку он зависит от В, то сделать это можно только совместно с В. С другой стороны, В легко использовать повторно, так как он ни от чего не зависит. Чем от меньшего числа компонентов зависит данный, тем легче его будет использовать повторно.

Перед началом генерации кода необходимо соотнести каждый из файлов с соответствующими компонентами.

Диаграммы Компонентов применяются теми участниками проекта, которые отвечает за компиляцию системы. Они нужна там, где начинается генерация кода. Диаграмма Компонентов дает представление о том, в каком порядке нужно компилировать компоненты, а также какие исполняемые компоненты будут созданы системой. Эта диаграмма показывает соответствие классов реализованным компонентам.

На рисунках 2.64 и 2.65 представлены примеры диаграмм Компонентов для клиента и сервера ATM.

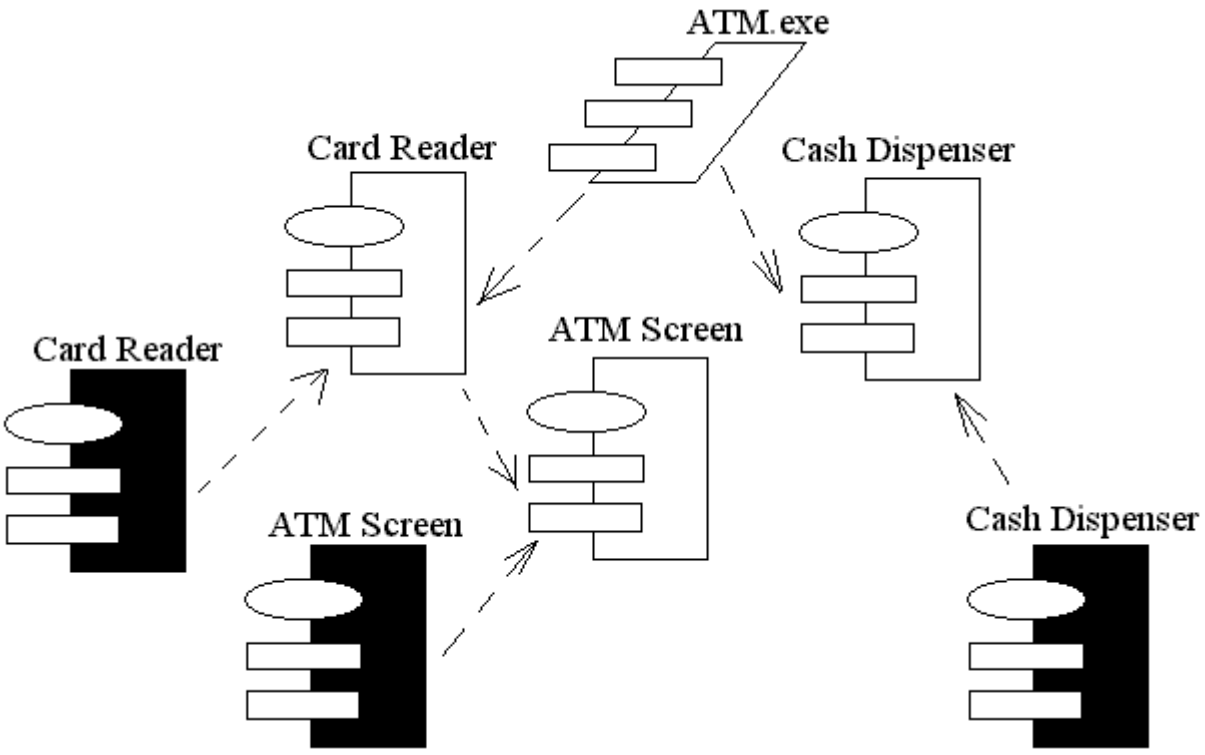


Рис. 2.64. Диаграмма Компонентов для клиента ATM

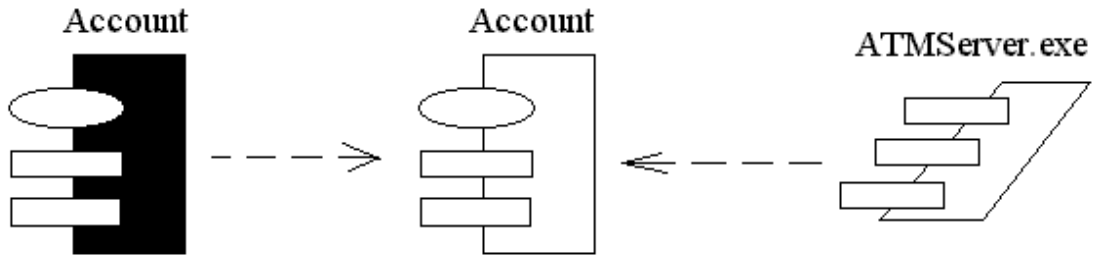


Рис. 2.65. Диаграмма Компонентов для сервера ATM

2.7.8. Диаграммы Размещения

На ***диаграмме Размещения*** (Deployment diagram) показывают физическое расположение различных компонентов системы в сети. Она содержит процессоры, устройства, процессы и связи между процессорами и устройствами. Для системы может быть создана только одна диаграмма Размещения. Диаграмма Размещения нужна всем участникам проекта. Эксплуатационный персонал, например, исходя из нее, планирует работу по установке системы.

Процессором (processor) называется любая машина, имеющая вычислительную мощность, т.е. способная производить обработку данных. В эту категорию попадают серверы, рабочие станции и другие устройства, содержащие физические процессоры. Процессор изображается с помощью нотации показанной на рис. 2.66.

Рис. 2.66. Графическое нотация для изображения процессора и устройства

Устройством (device) называется аппаратура, не обладающая вычислительной мощностью. Это, например, принтеры, терминалы ввода/вывода (dumb terminals), сканеры т.п. Устройство изображается с помощью нотации показанной на рис. 2.66.

Процессоры и устройства называются также узлами (nodes) сети.

Связью (connection) называется физическая связь между двумя процессорами, двумя устройствами или процессором и устройством. Чаще всего

связи отражают физическую сеть соединений между узлами сети. Кроме того, это может быть ссылка Интернета, связывающая два узла.

Процессом (process) называется поток обработки информации (execution), выполняющийся на процессоре. Процессом, например, считается исполняемый файл. Процессы отображаются непосредственно под процессором (процессорами), на котором выполняются. На рис. 2.67 приведена Диаграмма Размещения для системы ATM

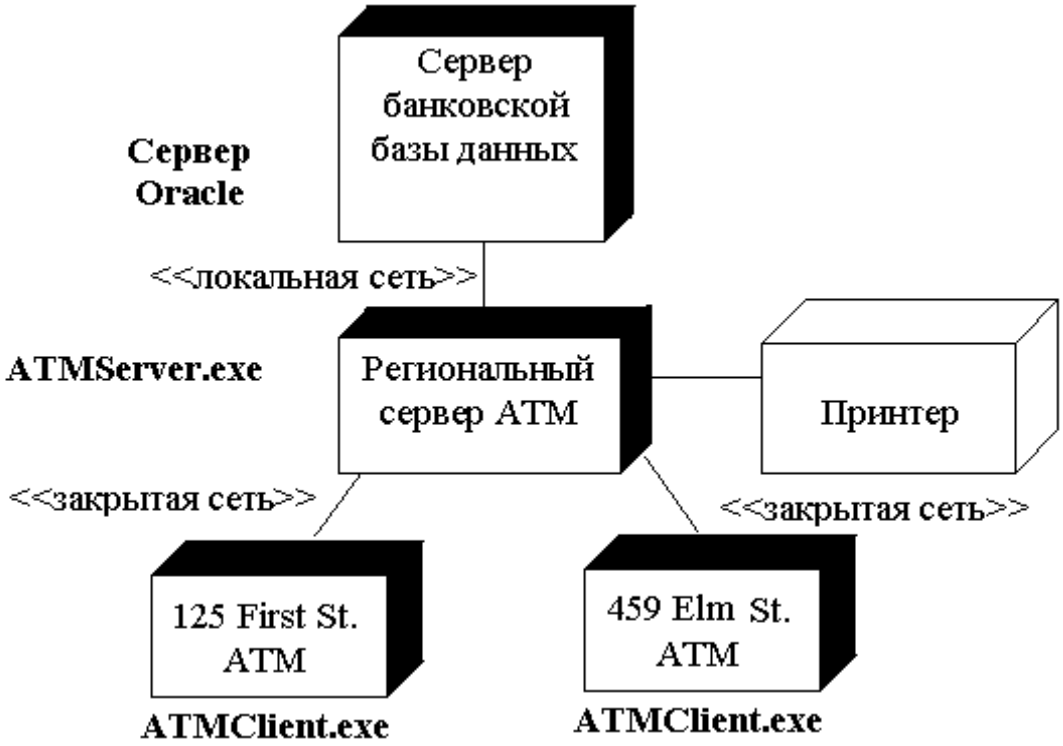


Рис. 2.67. Диаграмма Размещения для системы ATM