Лекция 6. Проектирование ПО при объектном подходе

Модели разрабатываемого ПО при объектном подходе основаны на предметах и явлениях реального мира. В основе этих моделей также лежит описание требуемого поведения разрабатываемого ПО, т.е. его функциональности, но это поведение связывается с состояниями элементов (объектов) конкретной предметной области.

На этапе анализа ставятся две задачи:

• уточнить требуемое поведение разрабатываемого ПО;

• разработать концептуальную модель его предметной области с точки зрения поставленных задач.

Стандартный язык описания разработки программных продуктов с использованием объектного подхода UML

В основе объектного подхода к разработке ПО лежит объектная декомпозиция - представление разрабатываемого ПО в виде совокупности объектов, в процессе взаимодействия которых через передачу сообщений и происходит выполнение функций (рис. 6.1).

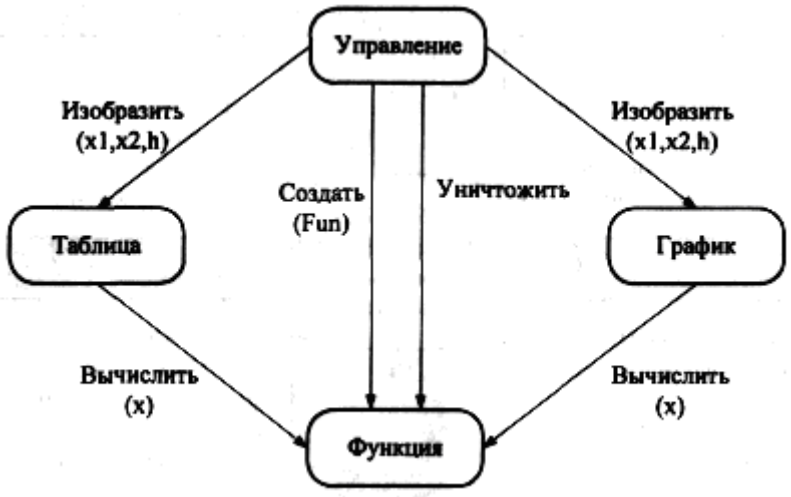

Рис. 6.1. Объектная декомпозиция программы построения таблиц и графиков

При объектном подходе, как и при структурном подходе, сразу можно выполнить декомпозицию только очень простого ПО. Поэтому на заре эпохи ООП были предложены различные методы анализа и проектирования ПО в рамках объектного подхода, использующие различные модели и нотации. Спорить о достоинствах и недостатках этих методов и моделей можно было бесконечно. Эта ситуация получила название «войны методов». Конец «войне методов» положило появление в 1995 г. первой версии языка UML (Unified Modeling Language - унифицированный язык моделирования), который фактически признан стандартным средством описания проектов, создаваемых с использованием объектно-ориентированного подхода. Его создатели - Гради Буч, Ивар Якобсон и Джеймс Рамбо.

Унифицированный язык моделирования UML представляет собой язык для определения, представления, проектирования и документирования программных систем, организационно-экономических, технических и других систем различной природы.

Спецификация разрабатываемого ПО при использовании UML объединяет несколько моделей: использования, логическую, реализации, процессов, развертывания (рис. 6.2).

Рис. 6.2. Полная спецификация разрабатываемого ПО при объектном подходе (UML)

Модель использования - описание функциональности ПО с точки зрения пользователя.

Логическая модель описывает ключевые абстракции ПО (классы, интерфейсы и т.п.), т.е. средства, обеспечивающие требуемую функциональность.

Модель реализации определяет реальную организацию программных модулей в среде разработки.

Модель процессов отображает организацию вычислений, оперирует понятиями «процессы» и «нити». Позволяет оценить производительность, масштабируемость и надежность ПО.

Модель развертывания показывает особенности размещения программных компонентов на конкретном оборудовании.

Каждая из моделей характеризует определенный аспект проектируемой системы, а все они вместе составляют относительно полную модель разрабатываемого ПО.

UML содержит стандартный набор диаграмм и нотаций самых разнообразных видов.

Стандарт UML (принятый OMG в 1997г.) - набор диаграмм для моделирования:

• диаграммы вариантов использования (use case diagrams) - для моделирования бизнес-процессов организации и требований к создаваемой системе;

• диаграммы классов (class diagrams) -для моделирования статической структуры классов системы и связей между ними;

• диаграммы поведения системы (behavior diagrams);

• диаграммы взаимодействия (interaction diagrams);

• диаграммы последовательности и кооперативные диаграммы (sequence, collaboration diagrams) - для моделирования процесса обмена сообщениями между объектами;

• диаграммы состояний (statechart diagrams) - для моделирования поведения объектов системы при переходе из одного состояния в другое;

• диаграммы деятельностей (activity diagrams) - для моделирования поведения системы в рамках различных вариантов использования, или моделирования деятельностей;

• диаграммы реализации (implementation diagrams);

• диаграммы компонентов (component diagrams) - для моделирования иерархии компонентов (подсистем) системы;

• диаграммы размещения (deployment diagrams) - для моделирования физической архитектуры системы.

Все диаграммы по возможности используют единую графическую нотацию, что облегчает их понимание.

Помимо указанных диаграмм, как и при структурном подходе, спецификация включает словарь терминов, а также различного рода описания и текстовые спецификации. Конкретный набор документации определяется разработчиком.

UML и методика Rational Unified Process поддерживаются пакетом Rational Rose фирмы Rational Software Corporation. Ряд диаграмм UML можно построить средствами MS Visual Modeler и других CASE-средств. В настоящее время практически все ведущие компьютерные компании используют UML при разработке ПО с использованием объектного подхода, UML фактически стал стандартом описания подобных разработок.

Диаграммы вариантов использования. Вариант использования представляет собой последовательность действий (транзакций), выполняемых системой в ответ на событие, инициируемое внешним объектом (действующим лицом). Вариант использования описывает типичное взаимодействие между пользователем и системой. В простейшем случае вариант использования определяется в процессе обсуждения с пользователем тех функций, которые он хотел бы реализовать.

Действующее лицо (actor) - это роль, которую пользователь играет по отношению к системе. Действующие лица представляют собой роли, а не конкретных людей или наименования работ. На диаграммах вариантов использования они изображаются в виде человеческих фигурок, действующее лицо может также быть внешней системой, которой необходима информация от данной системы. Действующих лиц показывают на диаграмме в том случае, когда им действительно необходимы некоторые варианты использования.

Три основных типа действующих лиц: пользователи системы, другие системы, взаимодействующие с данной, и время. Время становится действующим лицом, если от него зависит запуск каких-либо событий в системе.

Для наглядного представления вариантов использования в качестве основных элементов процесса разработки ПО применяются диаграммы вариантов использования. На рис. 6.3 показан пример такой диаграммы для банковской системы с банкоматами.

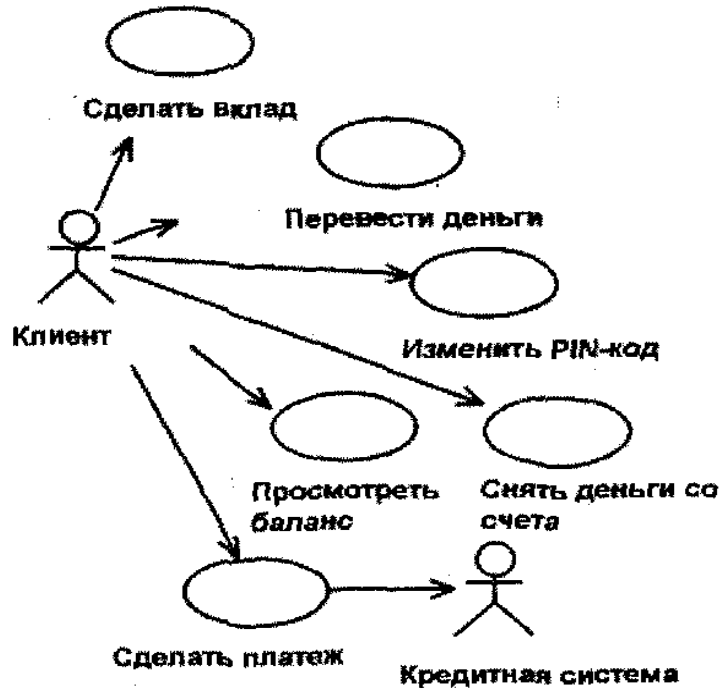

Рис. 6.3. Диаграмма вариантов использования для банковской системы с банкоматами

Человеческие фигурки – это действующие лица, овалы - варианты использования, а линии и стрелки - связи между действующими лицами и вариантами использования.

На диаграмме показаны два действующих лица: клиент и кредитная система. Шесть основных действий, выполняемых моделируемой системой: перевести деньги, сделать вклад, снять деньги со счета, просмотреть баланс, изменить PIN-код и сделать платеж.

На диаграмме показано взаимодействие между вариантами использования и действующими лицами. Она отражает требования к системе с точки зрения пользователя. Варианты использования - это функции, выполняемые системой, а действующие лица - это заинтересованные лица по отношению к создаваемой системе. Такие диаграммы показывают, какие действующие лица инициируют варианты использования. Из них также видно, когда действующее лицо получает информацию от варианта использования.

Данная диаграмма отражает взаимодействие между вариантами использования и действующими лицами банковской системы. В сущности, диаграмма вариантов использования иллюстрирует требования к системе. В примере клиент банка инициирует различные варианты использования: *«Снять деньги со счета», «Перевести деньги»,*

«Сделать вклад», «Просмотреть баланс» и «Изменить PIN-код». От варианта использования *«Сделать платеж»* стрелка указывает на *«Кредитную систему».* Действующими лицами могут быть внешние системы, потому *«Кредитная система»* показана как действующее лицо - она внешняя для банковской системы. Вариант использования *«Сделать платеж»* предоставляет *«Кредитной системе»* информацию об оплате по кредитной карточке.

Все варианты использования связаны с внешними требованиями к функциональности системы. Варианты использования следует анализировать вместе с действующими лицами системы, определяя при этом реальные задачи пользователей и рассматривая альтернативные способы решения этих задач. Действующие лица могут играть различные роли по отношению к варианту использования. Они могут пользоваться его результатами или могут сами непосредственно в нем участвовать. Значимость различных ролей действующего лица зависит от того, каким образом используются его связи.

Цель диаграмм вариантов использования - это документирование вариантов использования (все, входящее в сферу применения системы), действующих лиц (все вне этой сферы) и связей между ними.

Варианты использования начинают описывать, что должна будет делать система. Чтобы разработать систему, требуются более конкретные детали, которые описываются в документе «поток событий». Его цель - документирование процесса обработки данных, реализуемого в рамках варианта использования. Этот документ подробно описывает, что будут делать пользователи системы и что - сама система.

Поток событий не должен зависеть от реализации. Цель - описать то, что будет делать система, а не как она будет делать это. Поток событий включает: краткое описание; предусловия; основной поток событий; альтернативный поток событий; постусловия.

Связи между вариантами использования и действующими лицами

В языке UML для вариантов использования и действующих лиц поддерживается несколько типов связей: коммуникации, включения, расширения и обобщения.

Связь коммуникации - это связь между вариантом использования и действующим лицом. Связи коммуникации показывают с помощью однонаправленной ассоциации (линии со стрелкой). Направление позволяет понять, кто инициирует коммуникацию.

Связь включения применяется в ситуациях, когда имеется фрагмент поведения системы, который повторяется более чем в одном варианте использования. В примере банковской системы варианты использования *«Снять деньги со счета»* и *«Сделать вклад»* должны аутентифицировать клиента и его PIN-код перед осуществлением самой транзакции.

Чтобы не описывать процесс аутентификации для каждого из них, можно поместить эту функциональность в свой вариант использования «Аутентифицировать клиента».

Связь расширения применяется при описании изменений в нормальном поведении системы. Она позволяет варианту использования только при необходимости применять функциональные возможности другого.

На языке UML связи включения и расширения показывают в виде зависимостей с соответствующими стереотипами (рис. 6.4).

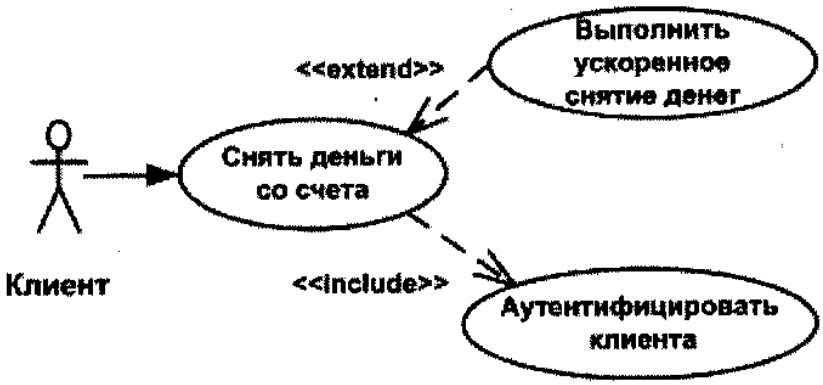

Рис. 6.4. Связи использования и расширения

С помощью *связи обобщения* показывают, что у нескольких действующих лиц имеются общие черты. Например, клиенты могут быть двух типов: корпоративные и индивидуальные.

Эту связь можно моделировать с помощью нотации (рис. 6.5). Связи этого типа нужны, если поведение действующего лица одного типа отличается от поведения другого.

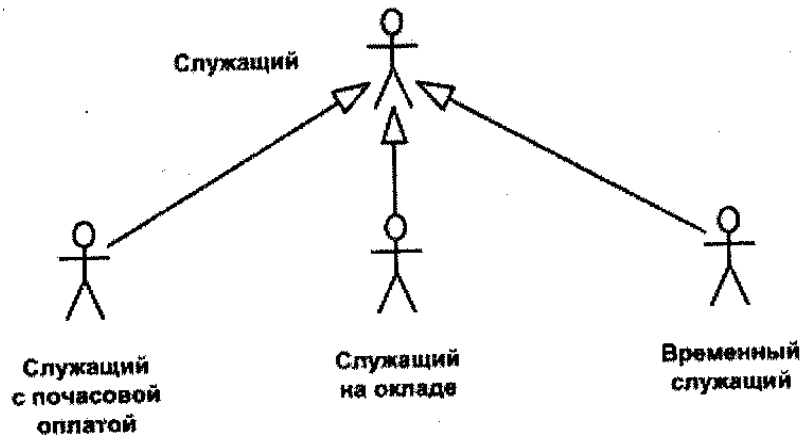

Рис. 6.5. Связь обобщения

Варианты использования необходимы на стадии формирования требований к ПО. Каждый вариант использования - это потенциальное требование к системе, и пока оно не выявлено, невозможно запланировать его реализацию.

Диаграммы взаимодействия описывают поведение взаимодействующих групп объектов. Диаграмма охватывает поведение объектов в рамках только одного варианта использования. На диаграмме отображаются ряд объектов и те сообщения, которыми они обмениваются между собой.

Два вида диаграмм взаимодействия: диаграммы последовательности (sequence diagrams) и кооперативные диаграммы (collaboration diagrams).

Диаграммы последовательности отражают поток событий, происходящих в рамках варианта использования. Например, вариант использования *«Снять деньги со счета»* предусматривает несколько возможных последовательностей: снятие денег, попытка снять деньги, не имея их достаточного количества на счете, попытка снять деньги по неправильному PIN-коду и др. На рис.6.6. - нормальный сценарий снятия некоторой суммы денег со счета.

Эта диаграмма последовательности отображает поток событий в рамках варианта использования *«Снять деньги со счета».* Все действующие лица показаны в верхней части диаграммы. В примере - действующее лицо Клиент (Customer). Объекты, требуемые системе для выполнения варианта использования *«Снять деньги со счета»,* также представлены в верхней части диаграммы. Стрелки соответствуют сообщениям, передаваемым между действующим лицом и объектом или между объектами для выполнения требуемых функций.

Объект изображается в виде прямоугольника на вершине пунктирной вертикальной линии - линии жизни (lifeline) объекта. Она представляет собой фрагмент ЖЦ объекта в процессе взаимодействия. Каждое Сообщение изображается в виде стрелки между линиями жизни двух объектов. Сообщения появляются в том порядке, как они показаны, сверху вниз.

Кооперативные диаграммы отображают поток событий через конкретный сценарий варианта использования. Диаграммы последовательности упорядочены по времени, а кооперативные диаграммы заостряют внимание на связях между объектами. На рис. 6.7 - кооперативная диаграмма, описывающая, как клиент снимает деньги со счета.

Здесь представлена вся та информация, которая была и на диаграмме последовательности, но кооперативная диаграмма по-другому описывает поток событий. Из нее легче понять связи между объектами, но труднее уяснить последовательность событий.

Поэтому часто для сценария создают диаграммы обоих типов. Хотя они служат одной цели и содержат одну же информацию, но представляют ее с разных точек зрения.

На кооперативной диаграмме также стрелки обозначают сообщения, обмен которыми осуществляется в рамках данного варианта использования. Но их временная последовательность указывается путем нумерации сообщений.

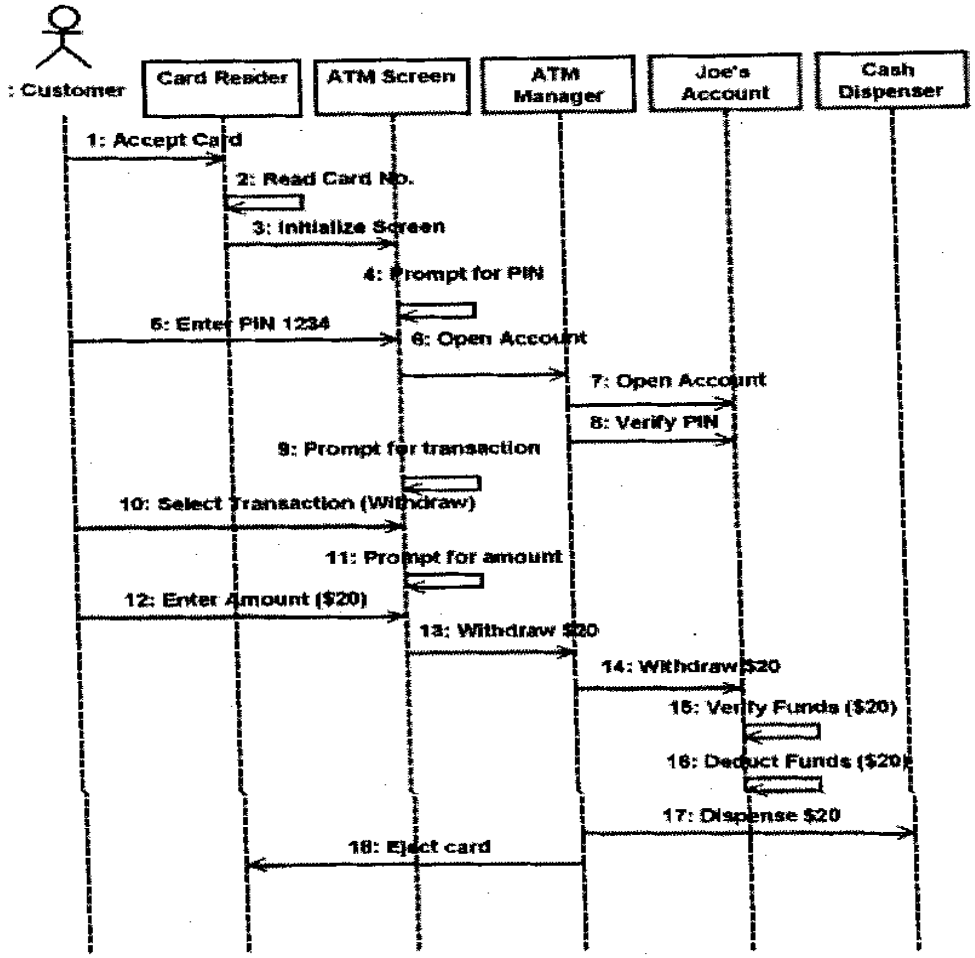

Рис. 6.6. Диаграмма последовательности

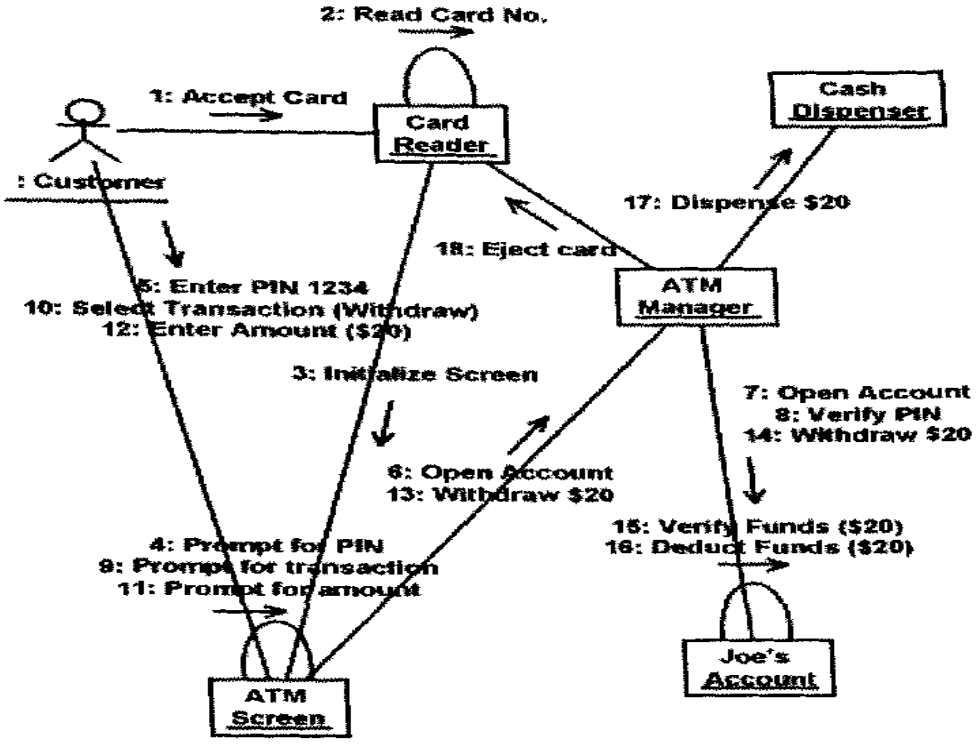

Рис. 6.7. Кооперативная диаграмма

Диаграмма классов определяет типы классов системы и статические связи, которые существуют между ними. На них изображаются атрибуты и операции классов, ограничения, которые накладываются на связи между классами. Диаграмма классов для варианта использования *«Снять деньги со счета»* показана на рис. 6.8.

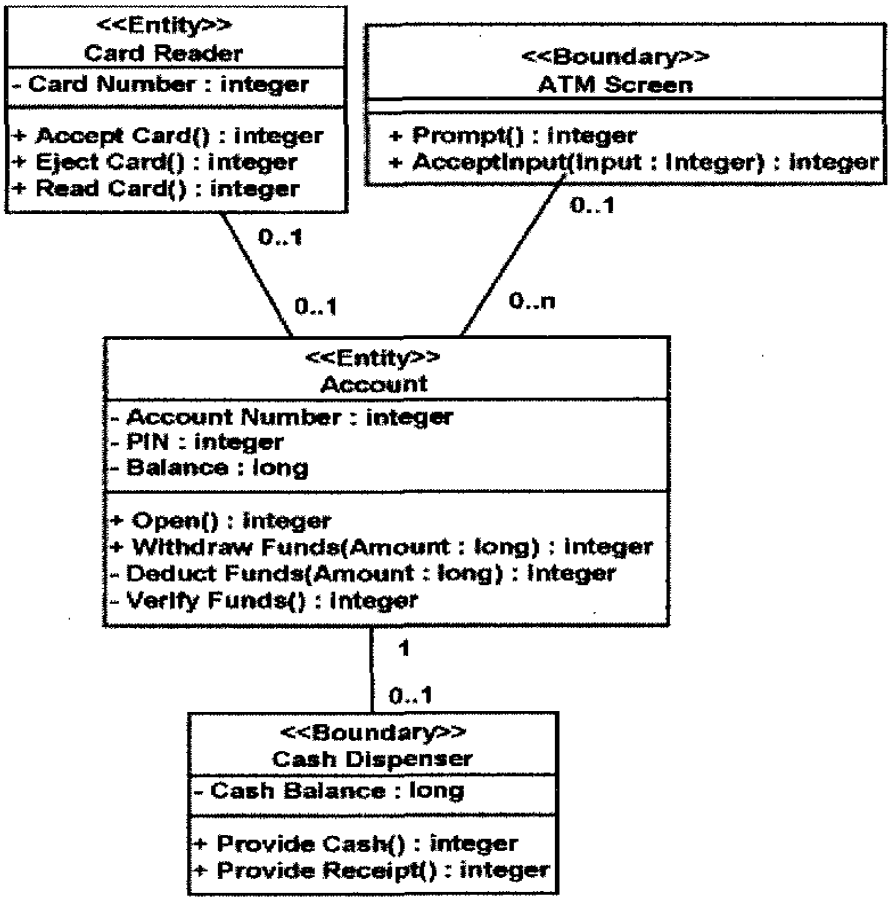

Рис. 6.8. Диаграмма классов для варианта использования «Снять деньги со счета»

На диаграмме показаны связи между классами, реализующими вариант использования *«Снять деньги со счета».* В процессе задействованы четыре класса: Card Reader (устройство для чтения карточек), Account (счет), ATM Screen (экран ATM) и Cash Dispenser (кассовый аппарат). Каждый класс на диаграмме - в виде прямоугольника, разделенного на три части. В первой содержится имя класса, во второй - его атрибуты. В последней - операции класса, отражающие его поведение (действия, выполняемые классом).

Связывающие классы линии отражают взаимодействие между классами. Класс Account связан с классом ATM Screen, потому что они непосредственно сообщаются и взаимодействуют друг с другом. Класс Card Reader не связан с классом Cash Dispenser, поскольку они не сообщаются друг с другом непосредственно.

Стереотипы классов. Стереотипы - это механизм, позволяющий разделять классы на категории. В языке UML основными стереотипами являются: Boundary (граница), Entity (сущность) и Control (управление). Можно создавать свои собственные стереотипы.

Граничные классы - это классы, которые расположены на границе системы и окружающей среды. Они включают все формы, отчеты, интерфейсы с аппаратурой (принтеры, сканеры) и с другими системами.

Классы-сущности отражают основные понятия (абстракции) предметной области и содержат хранимую информацию. Для каждого класса-сущности создают таблицу в БД.

Управляющие классы отвечают за координацию действий других классов, но сам не несет в себе функциональности - остальные классы не посылают ему большого количества сообщений. Вместо этого он сам посылает множество сообщений. Управляющий класс делегирует ответственность другим классам, его часто называют классом-менеджером. У каждого варианта использования имеется один управляющий класс, в системе могут быть и другие управляющие классы, общие для нескольких вариантов использования.

Механизм пакетов. Пакеты применяют, чтобы сгруппировать классы, обладающие некоторой общностью. Существует несколько подходов к группировке. Во-первых, можно группировать их по стереотипу - получается один пакет с классами-сущностями, один с граничными классами, один с управляющими классами и т.д. Этот подход полезен с т.зр. размещения готовой системы, т.к. все находящиеся на клиентских машинах компоненты с граничными Классами уже оказываются в одном пакете.

Другой подход - в объединении классов по их функциональности. Например, в пакете Security (безопасность) содержатся все классы, отвечающие за безопасность приложения. Другие пакеты могут называться Employee Maintenance (Работа с сотрудниками), Reporting (Подготовка отчетов) и Error Handling (Обработка ошибок). Преимущество этого подхода - в возможности повторного использования.

Механизм пакетов применим к любым элементам модели, а не только к классам.

Между двумя пакетами существует зависимость в случае, если между любыми двумя классами в пакетах существует любая зависимость. Диаграмма пакетов (рис. 6.9) представляет собой диаграмму, содержащую пакеты Классов и зависимости между ними.

Пакеты не отвечают на вопрос, как уменьшить количество зависимостей в системе, они помогают выделить зависимости, а затем поработать над снижением их количества.

Пакеты используют для больших проектов. Диаграммы пакетов - основное средство управления общей структурой системы.

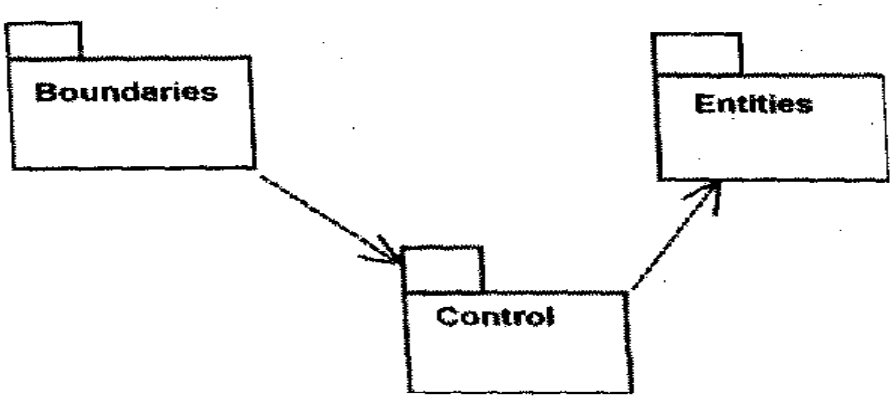

Рис.6.9. Диаграмма пакетов

Диаграммы состояний определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий. Существует много форм диаграмм состояний, незначительно отличающихся друг от друга семантикой.

На рис. 6.10 - пример диаграммы состояний для банковского счета (account). Можно наблюдать процесс перехода счета из одного состояния в другое. Если клиент требует закрыть счет, он переходит в состояние «закрыт». Требование клиента называется событием (event), такие события и вызывают переход из одного состояния в другое.

Если клиент снимает деньги со счета, он может перейти в состояние «Превышение кредита». Это происходит только в том случае, если баланс по счету меньше нуля, что отражено условием [отрицательный баланс] на диаграмме. Условие определяет, когда может произойти переход из одного состояния в другое.

На диаграмме два специальных состояния - начальное (start) и конечное (stop). Может быть только одно начальное состояние и столько угодно конечных состояний. Когда объект находится в конкретном состоянии, могут выполняться различные процессы. В примере при превышении кредита клиенту посылается сообщение. Процессы, происходящие в момент, когда объект находится в определенном состоянии, называются *действиями* (actions).

С состоянием можно связывать данные: деятельность, входное действие, выходное действие и событие. У перехода существует несколько спецификаций. Они включают события, аргументы, ограждающие условия, действия и посылаемые события.

Событие (event) - это то, что вызывает переход из одного состояния в другое. В примере событие «Клиент требует закрыть» вызывает переход счета из открытого в закрытое состояние. Событие размещают на диаграмме вдоль линии перехода.

У событий могут быть аргументы. Событие «Сделать вклад», вызывающее переход счета из состояния «Превышен счет» в состояние «Открыто», может иметь аргумент Amount (Количество), описывающий сумму депозита.

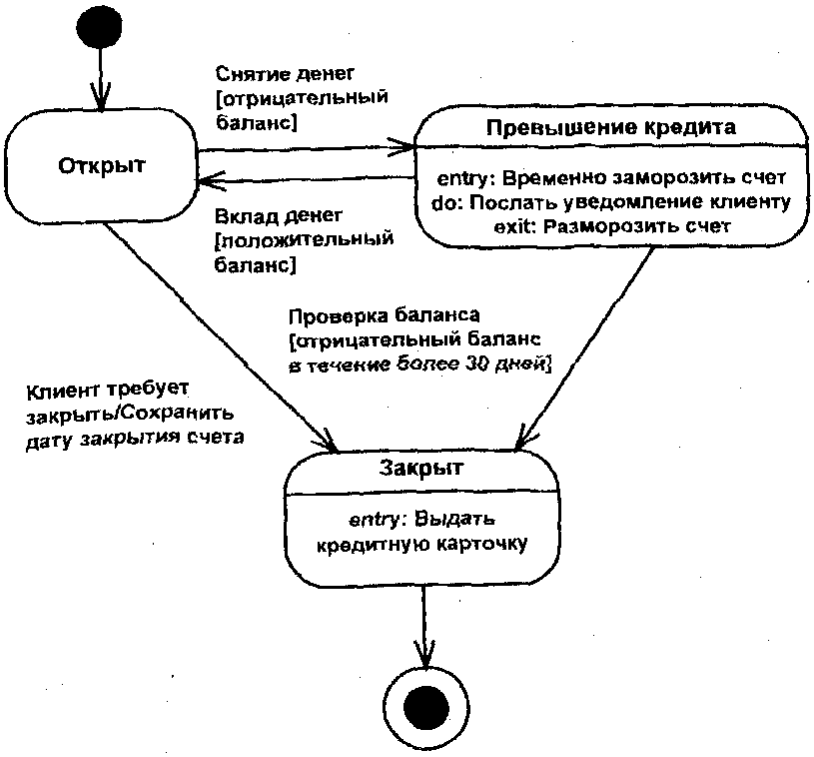

Рис. 6.10. Диаграмма состояний для класса Account

Диаграммы состояний не надо создавать для каждого класса, они применяются в сложных случаях. Если объект класса может существовать в нескольких состояниях и в каждом ведет себя по-разному, для него может потребоваться диаграмма состояний.

Диаграммы деятельностей полезны в описании поведения, включающего большое количество параллельных процессов. Достоинство - поддержка параллелизма. Они применяются для моделирования потоков работ и параллельного программирования. Недостаток - связи между действиями и объектами просматриваются не слишком четко.

Диаграммы деятельностей используют в ситуациях:

• анализ варианта использования. На этой стадии нас не интересует связь между действиями и объектами, а нужно только понять, какие действия должны иметь место и каковы зависимости в поведении системы;

• анализ потоков работ (workflow) в различных вариантах использования.

Диаграммы компонентов показывают, как выглядит модель на физическом уровне. На них изображены компоненты ПО и связи между ними. На такой диаграмме выделяют два типа компонентов: исполняемые компоненты и библиотеки кода.

На рис. 6.11 изображена одна из диаграмм компонентов для банковской системы.

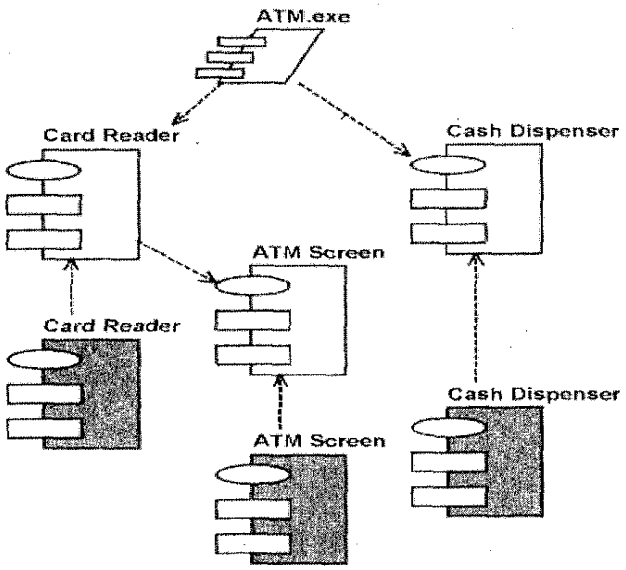

Рис. 6.11. Диаграмма компонентов для клиентской части системы

Пример банковской системы содержит два потока обработки, получаются два исполняемых файла. Один - это клиентская часть системы, она содержит компоненты Cash Dispenser, Card Reader и ATM Screen. Второй файл - это сервер, включающий в себя компонент Account. Диаграмма компонентов для сервера показана на рис.6.12.

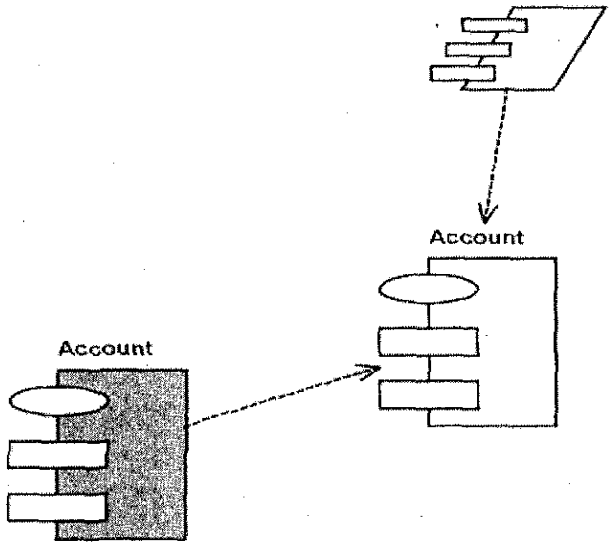

Рис. 6.12. Диаграмма компонентов для сервера

У системы может быть несколько диаграмм компонентов в зависимости от числа подсистем или исполняемых файлов. В общем случае пакеты - это совокупности компонентов. Пример банковкой системы содержит два пакета: клиентская часть и сервер. Диаграммы компонентов применяются теми участниками проекта, кто отвечает за компиляцию системы.

Диаграмма размещения отражает физические взаимосвязи между программными и аппаратными компонентами системы. Она показывает физическое расположение сети и местонахождение в ней различных компонентов.

Каждый узел на диаграмме размещения представляет собой некоторый тип вычислительного устройства - в большинстве случаев часть аппаратуры.

В данном примере банковская система состоит из большого количества подсистем, выполняемых на отдельных физических устройствах, или узлах (node). Диаграмма размещения для банковской системы показана на рис. 6.13.

Из диаграммы можно узнать о физическом размещении системы. Клиентские программы будут работать в нескольких местах на различных сайтах. Через закрытые сети будет осуществляться их сообщение с региональным сервером системы, с работающим ПО. Региональный сервер посредством локальной сети будет сообщаться с сервером банковской БД. С региональным сервером соединен принтер.

Диаграмма размещения используется менеджером проекта, пользователями, архитектором системы и эксплуатационным персоналом, чтобы понять физическое размещение системы и расположение ее отдельных подсистем.

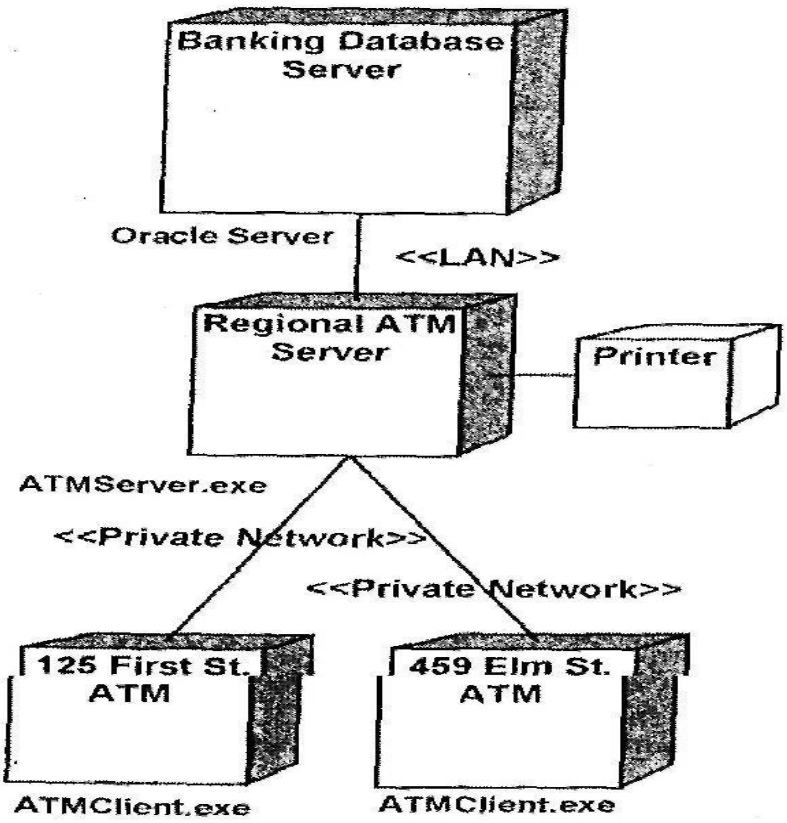

Рис. 6.13. Диаграмма размещения