Лекция 6. УНИФИЦИРОВАННЫЙ ЯЗЫК МОДЕЛИРОВАНИЯ UML

Унифицированный язык моделирования UML[1] (Unified Modeling Language) представляет собой язык для определения, представления, проектирования и документирования программных систем, организационно-экономических систем, технических систем и других систем различной природы. UML содержит стандартный набор диаграмм и нотаций самых разнообразных видов.

UML — это преемник того поколения методов объектно-ориентированного анализа и проектирования, которые появились в конце 1980-х и начале 1990-х годов. Создание UML фактически началось в конце 1994 г., когда Гради Буч и Джеймс Рамбо начали работу по объединению их методов Booch и OMT (Object Modeling Technique) под эгидой компании Rational Software. К концу 1995 г. они создали первую спецификацию объединенного метода, названного ими Unified Method, версия 0.8. Тогда же в 1995 г. к ним присоединился создатель метода OOSE (Object-Oriented Software Engineering) Ивар Якобсон. Таким образом, UML является прямым объединением и унификацией методов Буча, Рамбо и Якобсона, однако дополняет их новыми возможностями. Главными в разработке UML были следующие цели:

- предоставить пользователям готовый к использованию выразительный язык визуального моделирования, позволяющий им разрабатывать осмысленные модели и обмениваться ими;
- предусмотреть механизмы расширяемости и специализации для расширения базовых концепций;
- обеспечить независимость от конкретных языков программирования и процессов разработки;
- обеспечить формальную основу для понимания этого языка

[1] *Фаулер М., Скотт К.* UML в кратком изложении. Применение стандартного языка объектного моделирования: Пер. с англ. — М.: Мир, 1999.

моделирования (язык должен быть одновременно точным и доступным для понимания, без лишнего формализма);

- стимулировать рост рынка объектно-ориентированных инструментальных средств;
- интегрировать лучший практический опыт.

UML находится в процессе стандартизации, проводимом OMG (Object Management Group) — организацией по стандартизации в области объектно-ориентированных методов и технологий, в настоящее время принят в качестве стандартного языка моделирования и получил широкую поддержку в индустрии ПО. UML принят на вооружение почти всеми крупнейшими компаниями — производителями ПО (Microsoft, IBM, Hewlett-Packard, Oracle, Sybase и др.). Кроме того, почти все мировые производители CASE-средств, помимо IBM Rational

Software, поддерживают UML в своих продуктах (Together (Borland), Paradigm Plus (Computer Associates), System Architect (Popkin Software), Microsoft Visual Modeler и др.). Полное описание UML можно найти на сайтах http://www.omg.org и http://www.rational.com.

Стандарт UML версии 1.1, принятый OMG в 1997 г., предлагает следующий набор диаграмм:

- Структурные (structural) модели:

диаграммы классов (class diagrams) - для моделирования статической структуры классов системы и связей между ними; диаграммы компонентов (component diagrams) — для моделирования иерархии компонентов (подсистем) системы;

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

- Модели поведения (behavioral):

диаграммы вариантов использования (use case diagrams) — для моделирования бизнес-процессов и функциональных требований к создаваемой системе; диаграммы взаимодействия (interaction diagrams):

диаграммы последовательности (sequence diagrams) и кооперативные диаграммы (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 деятельности (activity diagrams) — для моделирования поведения системы в рамках различных вариантов использования, или потоков управления.

2.5.1. ДИАГРАММЫ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

Идея описания функциональных требований в виде вариантов использования (use case) была сформулирована в 1986 г. Иваром Якобсоном. Эта идея была признана конструктивной и получила широкое одобрение. Впоследствии наиболее значительный вклад в решение проблемы определения сущности вариантов использования и способов их описания внес Алистер Коберн[2].

Вариант использования представляет собой последовательность действий (транзакций), выполняемых системой в ответ на событие, инициируемое некоторым внешним объектом (действующим лицом). Вариант использования описывает типичное взаимодействие между пользователем и системой и отражает представление о поведении системы с точки зрения пользователя. В простейшем случае вариант использования определяется в процессе обсуждения с пользователем тех функций, которые он хотел бы реализовать, или целей, которые он преследует по отношению к разрабатываемой системе.

Действующее лицо (actor) — это роль, которую пользователь играет по отношению к системе. Действующие лица представляют собой роли, а не конкретных людей или наименования работ.

---

[2] *Коберн А.* Современные методы описания функциональных требований к системам.: Пер. с англ. - М.: ЛОРИ, 2002

Несмотря на то что на диаграммах вариантов использования они изображаются в виде стилизованных человеческих фигурок, действующее лицо может также быть внешней системой, которой необходима некоторая информация от данной системы.

Действующие лица делятся на три основных типа — 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 применяются диаграммы вариантов использования. На рис. 2.47 показан пример такой диаграммы для банковской системы.

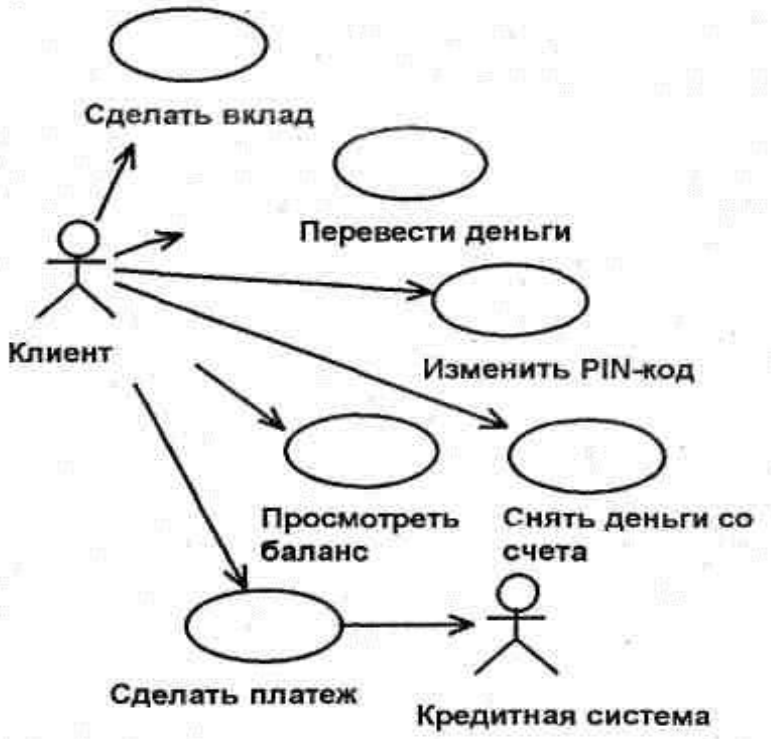

Рис. 2.47. Пример диаграммы вариантов использования

На данной диаграмме человеческие фигурки обозначают действующих лиц, овалы — варианты использования, а линии и стрелки — различные связи между действующими лицами и вариантами использования. На этой диаграмме показаны два действующих лица: клиент и кредитная система. Существует также шесть основных действий, выполняемых моделируемой

системой: перевести деньги, сделать вклад, снять деньги со счета, просмотреть баланс, изменить PIN-код и сделать платеж.

На диаграмме вариантов использования показано взаимодействие между вариантами использования и действующими лицами. Она отражает функциональные требования к системе с точки зрения пользователя. Таким образом, варианты использования — это функции, выполняемые системой, а действующие лица — это заинтересованные лица (stakeholders) 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. Направленная от варианта использования к действующему лицу стрелка показывает, что вариант использования предоставляет некоторую информацию, используемую действующим лицом. В данном случае вариант использования «Сделать платеж» предоставляет Кредитной системе информацию об оплате по кредитной карточке.

Действующие лица могут играть различные роли по отношению к варианту использования. Они могут пользоваться его результатами сами непосредственно в нем участвовать. Значимость различных ролей действующего лица зависит от того, каким образом используются его связи.

Цель построения диаграмм вариантов использования — документирование функциональных требований к системе в самом общем виде, поэтому они должны быть предельно простыми. При построении диаграмм вариантов использования нужно придерживаться следующих правил:

- Не моделируйте связи между действующими лицами. По определению действующие лица находятся вне сферы действия системы. Это означает, что связи между ними также не относятся к ее компетенции.
- Не соединяйте стрелкой два варианта использования непосредственно. Диаграммы данного типа описывают только сами варианты использования, а не порядок их выполнения. Для отображения порядка выполнения вариантов использования применяют диаграммы

деятельности.

- Каждый вариант использования должен быть инициирован действующим лицом. Это означает, что всегда должна быть стрелка, начинающаяся на действующем лице и заканчивающаяся на варианте использования.

Хорошим источником для идентификации вариантов использования служат внешние события. Следует начать с перечисления всех событий, происходящих во внешнем мире, на которые система должна каким-то образом реагировать. Какое-либо конкретное событие может повлечь за собой реакцию системы, не требующую вмешательства пользователей, или, наоборот, вызвать чисто пользовательскую реакцию. Идентификация событий, па которые необходимо реагировать, помогает идентифицировать варианты использования.

Диаграмма вариантов использования является самым общим представлением функциональных требований к системе. Однако моделирование вариантов использования не сводится только к рисованию диаграмм. Для последующего проектирования системы требуются более конкретные детали. Эти детали описываются в документе, называемом **«сценарий варианта использования»** или **«поток событий» (flow of events).** Целью потока событий является подробное документирование процесса взаимодействия действующего лица с системой, реализуемого в рамках варианта использования. В потоке событий должно быть описано все, что служит удовлетворению запросов действующих лиц.

Хотя поток событий и описывается подробно, он также не должен зависеть от реализации. Цель - описать, что будет делать система, а не как она будет делать это. Обычно описание потока событий включает следующие разделы:

- краткое описание;
- предусловия (pre-conditions);
- основной поток событий;
- альтернативные потоки событий;

- постусловия (post-conditions);
- расширения (extensions).

Последовательно рассмотрим эти составные части.

Краткое описание. Каждый вариант использования должен иметь краткое описание того, что в нем происходит. Например, вариант использования «Перевести деньги» может содержать следующее описание.

Вариант использования «Перевести деньги» позволяет клиенту или служащему банка переводить деньги с одного счета до востребования или сберегательного счета на другой.

Предусловия. Предусловия варианта использования - это такие условия, которые должны быть выполнены, прежде чем вариант использования начнет выполняться сам. Например, таким условием может быть выполнение другого варианта использования или наличие у пользователя прав доступа, требуемых для начала работы. Не у всех вариантов использования бывают предусловия. Ранее упоминалось, что диаграммы вариантов использования не должны отражать порядок их выполнения. Такую информацию можно описать с помощью предусловий. Например, предусловием одного варианта использования может быть то, что в это время должен выполняться другой.

Основной и альтернативный потоки событий. Конкретные детали вариантов использования описываются в основном в альтернативных потоках событий. Поток событий поэтапно описывает, что должно происходить во время выполнения заложенной в варианты использования функциональности. Поток событий уделяет внимание тому, *что* будет делать система, а не *как* она будет делать это, причем описывает все это с точки зрения пользователя. Основной поток событий описывает нормальный ход событий (при отсутствии ошибок), и при наличии нескольких возможных вариантов хода событий может разветвляться на подчиненные потоки (subflow). Альтернативные потоки описывают отклонения от нормального хода событий (ошибочные ситуации) и их обработку. Например, потоки событий варианта использования «Снять деньги со счета» могут выглядеть следующим образом:

Основной поток событий

1. Вариант использования начинается, когда клиент вставляет свою карточку в банкомат.
2. Банкомат выводит приветствие и предлагает клиенту ввести свой персональный PIN-код.
3. Клиент вводит PIN-код.
4. Банкомат подтверждает введенный код.
5. Банкомат выводит список доступных действий: сделать вклад, снять деньги со счета, перевести деньги
6. Клиент выбирает пункт «Снять деньги со счета».
7. Банкомат запрашивает, сколько денег надо снять.
8. Клиент вводит требуемую сумму.
9. Банкомат определяет, имеется ли на счету достаточно денег.
10. Банкомат вычитает требуемую сумму из счета клиента.
11. Банкомат выдает клиенту требуемую сумму наличными.
12. Банкомат возвращает клиенту его карточку.
13. Банкомат печатает чек для клиента.
14. Вариант использования завершается.

Альтернативный поток событий 1. Ввод неправильного PIN-кода.

4а1. Банкомат информирует клиента, что код введен неправильно.

4а2. Банкомат возвращает клиенту его карточку.

4а3. Вариант использования завершается.

Альтернативный поток событий 2. Недостаточно денег на счете.

9а1. Банкомат информирует клиента, что денег на его счете недостаточно.

9а2. Банкомат возвращает клиенту его карточку.

9а3. Вариант использования завершается.

Альтернативный поток событий 3. Ошибка в подтверждении запрашиваемой суммы.

9б1. Банкомат сообщает пользователю, что при подтверждении запрашиваемой суммы произошла ошибка, и дает ему номер телефона службы поддержки клиентов банка.

9б2. Банкомат заносит сведения об ошибке в журнал ошибок. Каждая запись содержит дату и время ошибки, имя клиента, номер его счета и код ошибки.

9б3. Банкомат возвращает клиенту его карточку.

9б4. Вариант использования завершается.

Как видно из приведенного примера, хорошо написанный поток событий должен легко читаться и состоять из предложений, написанных в единой грамматической форме. На обучение его чтению не должно уходить больше нескольких минут. При написании основного потока событий нужно придерживаться следующих правил:

- использовать простые предложения;
- явно указывать в каждом пункте, кто выполняет действие — действующее лицо или система;
- не показывать слишком незначительные действия;
- не показывать детальные действия пользователя в процессе работы с пользовательским интерфейсом;
- не рассматривать возможные ошибочные ситуации (использовать действия «подтвердить», а не «проверить»).

При выявлении альтернативных потоков событий нужно в первую очередь обратить внимание на ситуации, связанные с:

- некорректными действиями пользователя (например, ввод неверного пароля);
- бездействием действующего лица (например, истечением времени

ожидания пароля);

- внутренними ошибками в разрабатываемой системе, которые должны быть обнаружены и обработаны в обычном порядке (например, заблокирован автомат для выдачи наличных);
- критически важными недостатками в производительности системы (например, время реакции не укладывается в 5 секунд).

Постусловия. Постусловиями называются такие условия, которые всегда должны быть выполнены после завершения варианта использования. Например, в конце варианта использования можно пометить флажком какой-нибудь переключатель. Информация такого типа входит в состав постусловий. Как и для предусловий, с помощью постусловий можно вводить информацию о порядке выполнения вариантов использования системы. Если, например, после одного из вариантов использования должен всегда выполняться другой, это можно описать как постусловие. Такие условия имеются не у каждого варианта использования.

Расширения. Этот пункт присутствует, если в основном потоке событий имеют место относительно редко встречающиеся ситуации (частные случаи). Описание таких ситуаций выносится в данный пункт.

В диаграммах вариантов использования может присутствовать несколько типов связей. Это связи коммуникации (communication), включения (include), расширения (extend) и обобщения (generalization).

Связь коммуникации — это связь между вариантом использования и действующим лицом, она изображается с помощью однонаправленной ассоциации (линии со стрелкой). Направление стрелки позволяет понять, кто инициирует коммуникацию.

Связь включения применяется в тех ситуациях, когда имеется какой-либо фрагмент поведения системы (часть потока событий), который повторяется более чем в одном варианте использования. С помощью таких связей обычно моделируют многократно используемую функциональность. В примере с банковской системой варианты использования «Снять деньги со счета» и

«Сделать вклад» должны аутентифицировать клиента и его PIN-код перед тем, как допустить осуществление самой транзакции. Вместо того чтобы подробно описывать процесс аутентификации для каждого из вариантов использования, можно поместить эту функциональность в свой собственный вариант использования под названием «Аутентифицировать клиента».

Связь расширения применяется при наличии изменений в нормальном поведении системы (описанных в пункте «Расширения»), которые также выносятся в отдельный вариант использования.

Связи включения и расширения изображаются в виде зависимостей, как показано на рис. 2.48.

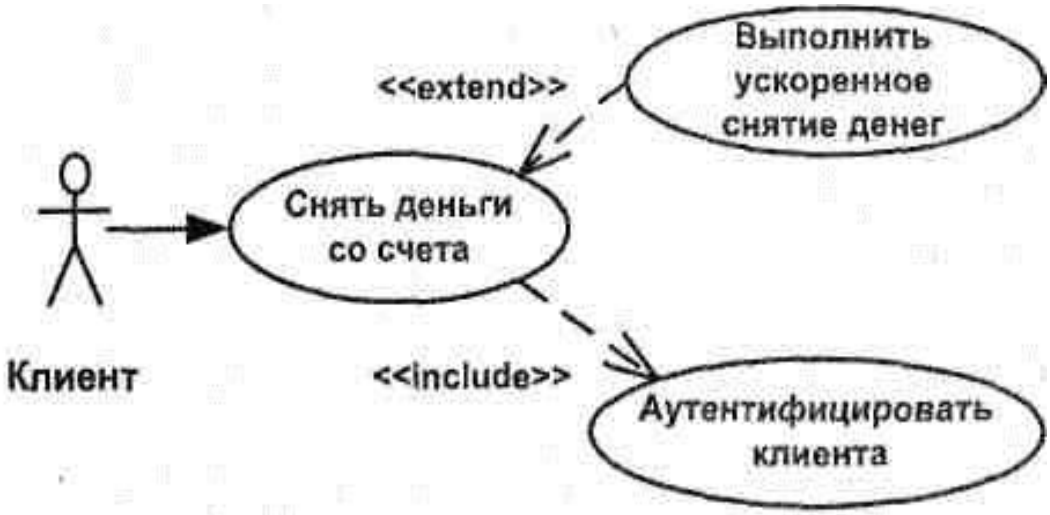

Рис. 2.48. Связи включения и расширения

С помощью связи обобщения показывают, что у нескольких действующих лиц имеются общие черты и различия. Например, у служащих организации имеются как общие свойства, так и разные способы оплаты труда (рис. 2.49).

Нет необходимости всегда создавать связи этого типа. В общем случае они нужны, если отличия в поведении действующего лица одного типа от поведения другого затрагивают функции системы. Если оба подтипа используют одни и те же варианты использования, показывать обобщение действующего лица не требуется.

Варианты использования являются необходимым средством на стадии формирования требований к ПО. Каждый вариант использования — это потен-

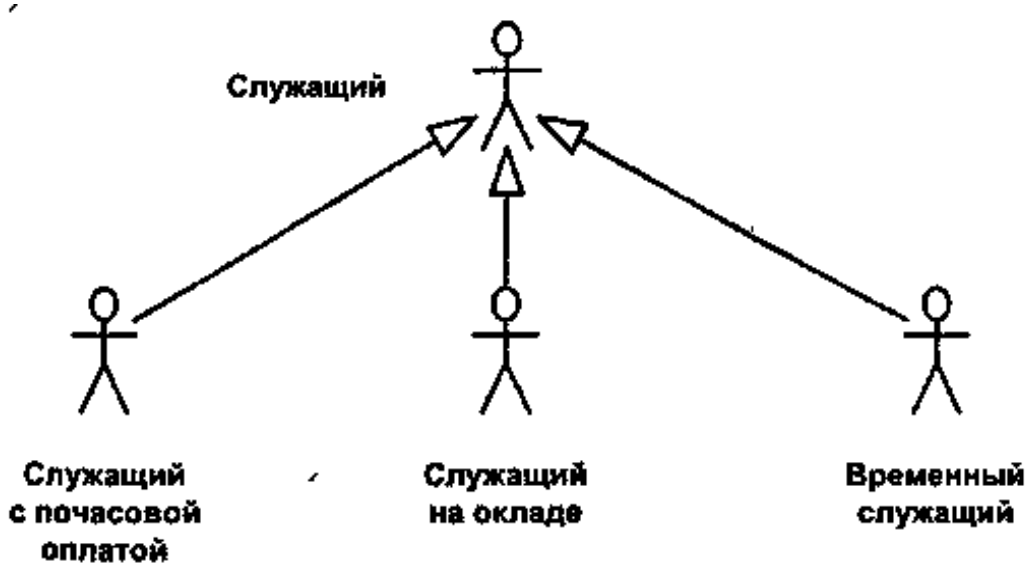

Рис. 2.49. Обобщение действующего лица

циальное требование к системе, и пока оно не выявлено, невозможно запланировать его реализацию.

Различные разработчики подходят к описанию вариантов использования с разной степенью детализации. Например, Ивар Якобсон утверждал, что для проекта с трудоемкостью 10 человеко-лет количество вариантов использования может составлять около 20 (не считая связей «включения» и «расширения»). Следует предпочитать небольшие и детализированные варианты использования, поскольку они облегчают составление и реализацию согласованного плана проекта.

Достоинства модели вариантов использования заключаются в том, что она:

- определяет пользователей и границы системы;
- определяет системный интерфейс;
- удобна для общения пользователей с разработчиками;
- используется для написания тестов;
- является основой для написания пользовательской документации;
- хорошо вписывается в любые методы проектирования (как объектно-ориентированные, так и структурные).

2.5.2.

ДИАГРАММЫ ВЗАИМОДЕЙСТВИЯ

Диаграммы взаимодействия описывают поведение взаимодействующих групп объектов (в рамках варианта использования или некоторой операции класса). Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного потока событий варианта использования. На такой диаграмме отображается ряд объектов и те сообщения, которыми они обмениваются между собой.

Сообщение (message) — средство, с помощью которого объект-отправитель запрашивает у объекта-получателя выполнение одной из его операций.

Информационное (informative) сообщение — сообщение, снабжающее объект-получатель некоторой информацией для обновления его состояния.

Сообщение-запрос (interrogative) — сообщение, запрашивающее выдачу некоторой информации об объекте-получателе.

Императивное (imperative) сообщение — сообщение, запрашивающее у объекта-получателя выполнение некоторых действий.

Существуют два вида диаграмм взаимодействия: диаграммы последовательности и кооперативные диаграммы.

Диаграммы последовательности отражают временную последовательность событий, происходящих в рамках варианта использования. Например, вариант использования «Снять деньги со счета» предусматривает несколько возможных потоков событий, таких как снятие денег, попытка снять деньги, не имея их достаточного количества на счете, попытка снять деньги по неправильному PIN-коду и некоторых других. Нормальный сценарий (основной поток событий) снятия некоторой суммы денег со счета показан на рис. 2.50.

Все действующие лица показаны в верхней части диаграммы; и приведенном примере изображено действующее лицо Клиент (Customer).

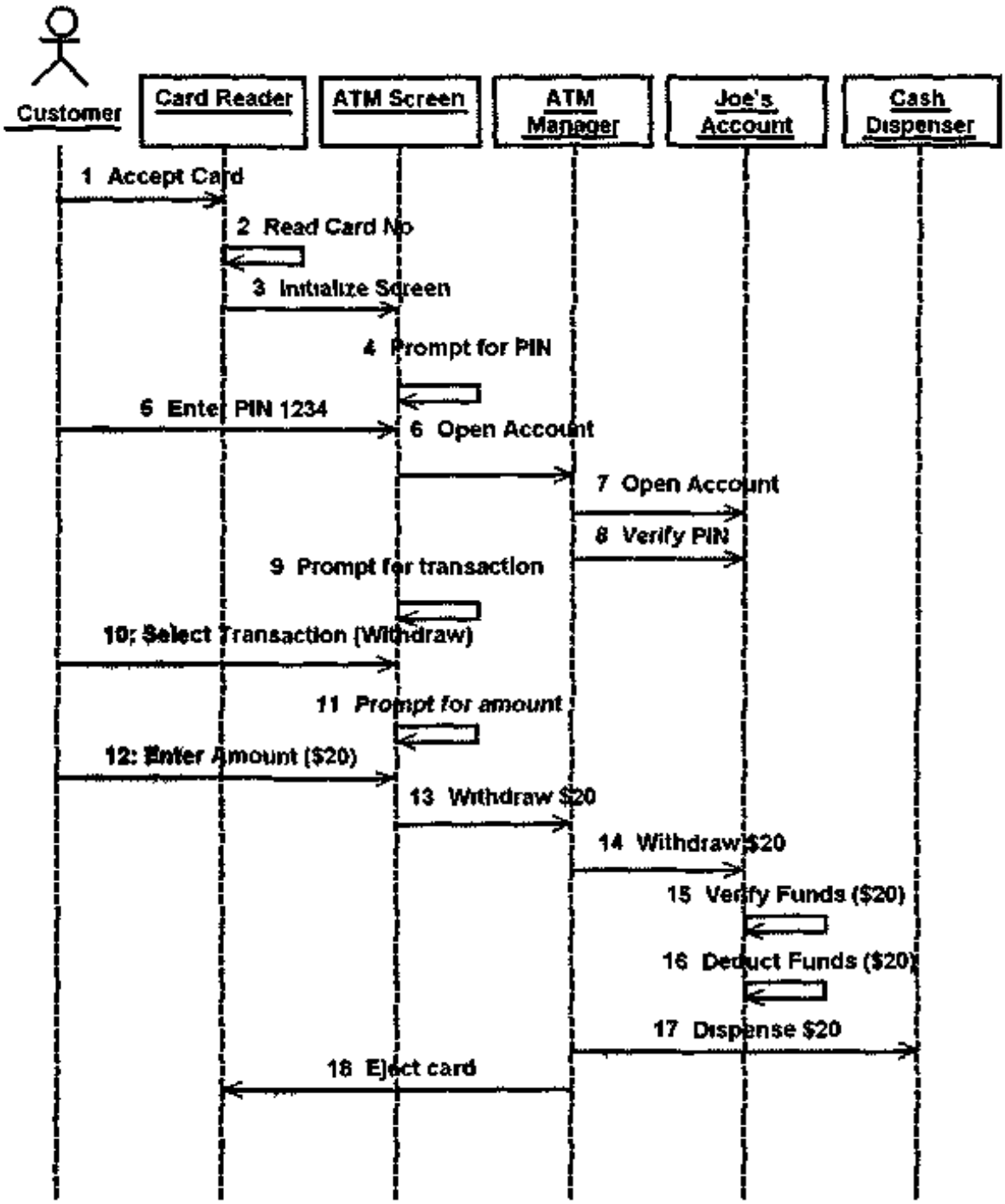

Рис. 2.50. Диаграмма последовательности

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.

На диаграмме последовательности объект изображается в виде прямоугольника на вершине пунктирной вертикальной линии. Эта вертикальная линия называется линией жизни (lifeline) объекта. Она представляет собой фрагмент жизненного цикла объекта в процессе взаимодействия.

Каждое сообщение представляется в виде стрелки между линиями жизни двух объектов. Сообщения появляются в том порядке, как они показаны на странице сверху вниз. Каждое сообщение помечается, как минимум, именем сообщения; при желании можно добавить также аргументы и некоторую управляющую информацию, и, кроме того, можно показать самоделегирование (self-delegation) — сообщение, которое объект посылает самому себе, при этом стрелка сообщения указывает на ту же самую линию жизни.

Один из способов первоначального обнаружения некоторых объектов - это изучение имен существительных в потоке событий. Поток событий для варианта использования «Снять деньги со счета» говорит о человеке, снимающем некоторую сумму денег со счета с помощью банкомата.

Не все объекты, показанные на диаграмме, явно присутствуют в потоке событий. Там, например, может не быть форм для заполнения, но их необходимо показать на диаграмме, чтобы позволить действующему лицу ввести новую информацию в систему или просмотреть ее. В потоке событий, скорее всего, также не будет и управляющих объектов (control objects). Эти объекты управляют последовательностью событий в варианте использования.

Вторым видом диаграммы взаимодействия является ***кооперативная диаграмма*** (рис. 2.51).

Подобно диаграммам последовательности кооперативные диаграммы отображают поток событий варианта использования. Диаграммы последовательности упорядочены по времени, а кооперативные диаграммы концентрируют внимание на связях между объектами. На рис. 2.51 приведена кооперативная диаграмма, описывающая, как клиент снимает деньги со счета.

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

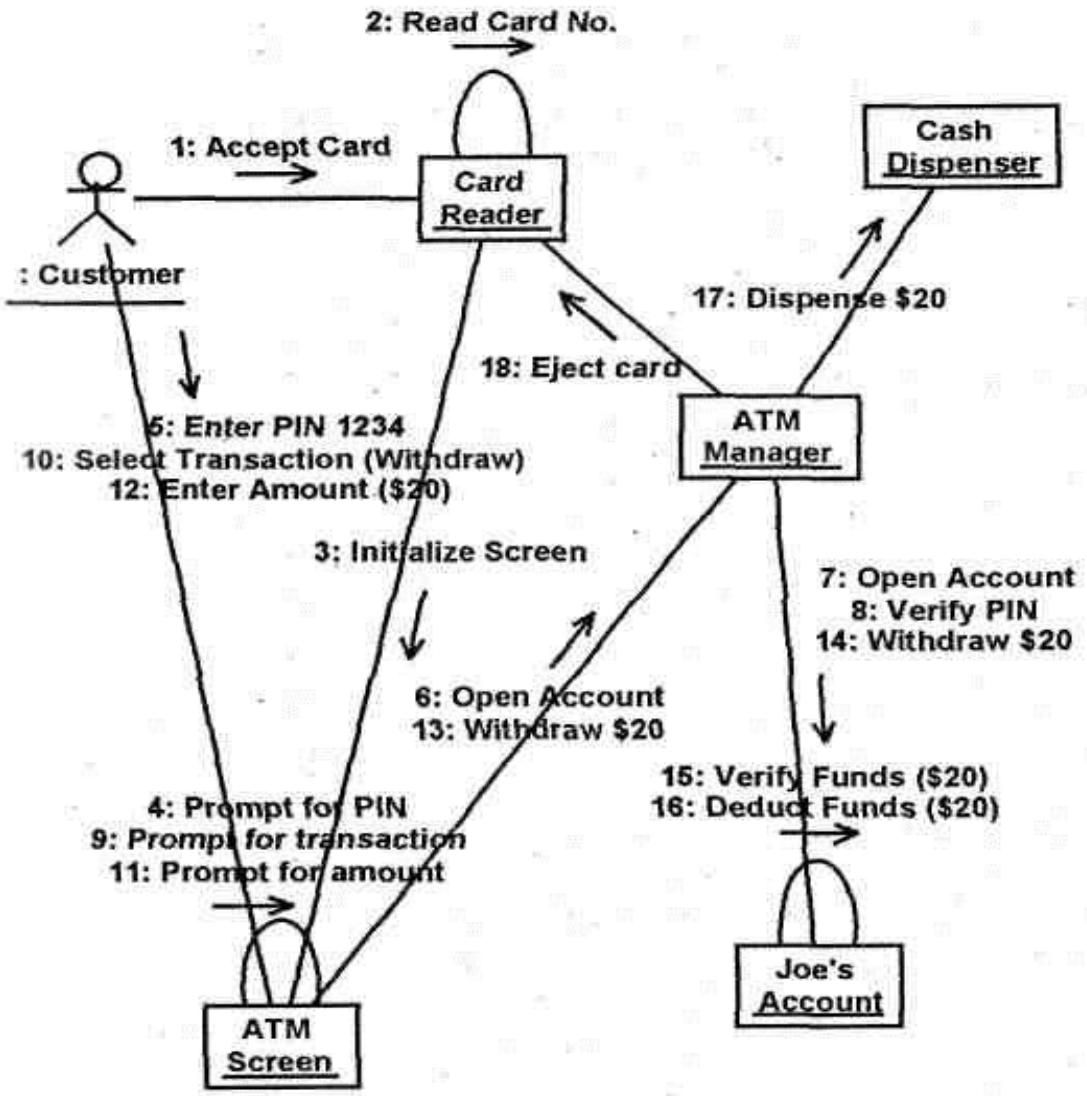

Рис. 2.51. Кооперативная диаграмма

По этой причине часто для какого-либо сценария создают диаграммы обоих типов. Хотя они служат одной и той же цели и содержат одну и ту же информацию, но представляют ее с различных точек зрения.

На кооперативной диаграмме так же, как и на диаграмме последовательности, стрелки обозначают сообщения, обмен которыми осуществляется в рамках

данного варианта использования. Их временная последовательность, однако, указывается путем нумерации сообщений.

2.5.3. ДИАГРАММЫ КЛАССОВ

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Диаграмма классов для варианта использования «Снять деньги со счета» показана на рис. 2.52.

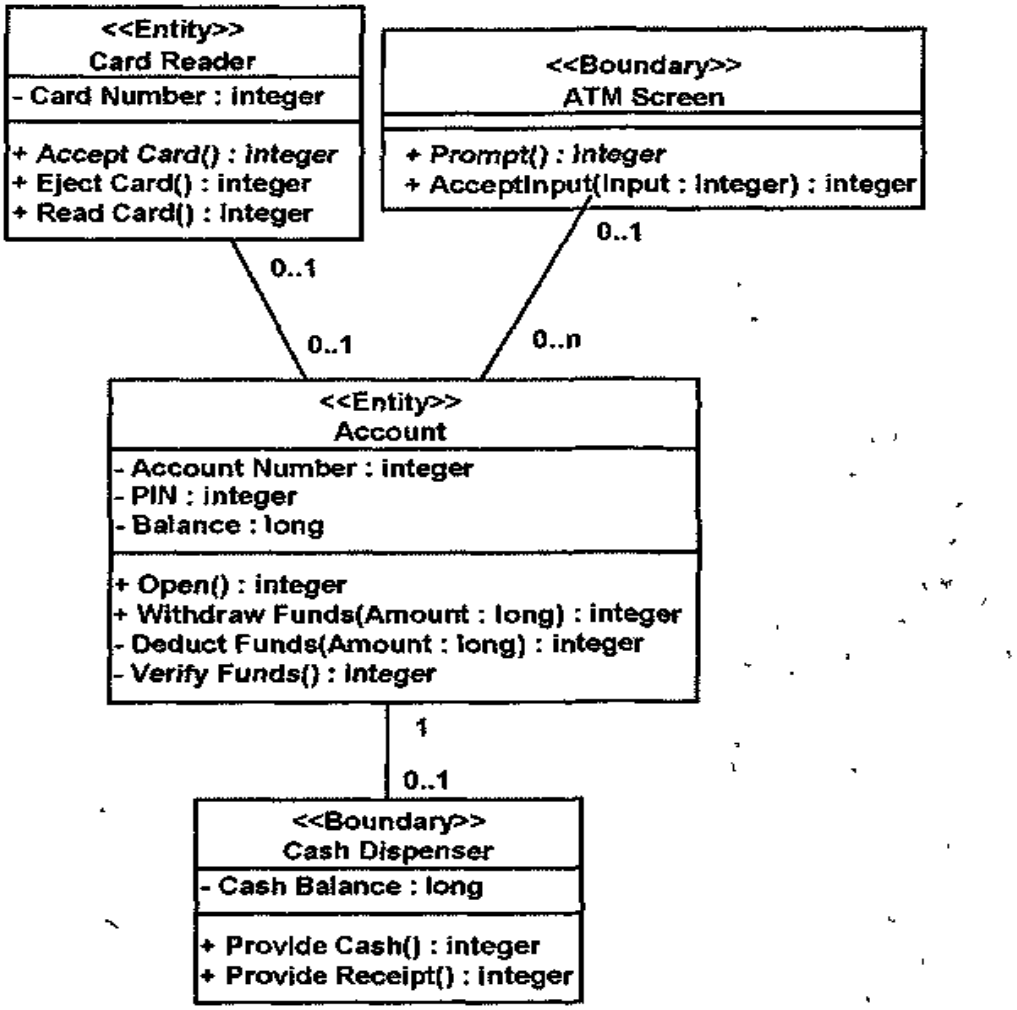

Рис. 2.52. Диаграмма классов для варианта использования «Снять деньги со счета»

На этой диаграмме присутствуют четыре класса: Card Reader (устройство для чтения карточек), Account (счет), ATM Screen (экран АТМ) и Cash Dispenser (кассовый аппарат). Связывающие классы линии отражают взаимодействие между классами. Так, 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. В изображении классов присутствуют также стереотипы, которые будут рассматриваться в подразд. 2.5.8.

Для группировки классов, обладающих некоторой общностью, применяются пакеты. ***Пакет*** — общий механизм для организации элементов модели в группы. Пакет может включать другие элементы. Каждый элемент модели может входить только в один пакет. Пакет является средством организации модели в процессе разработки, повышения ее управляемости и читаемости, а также единицей управления конфигурацией.

Существуют несколько подходов к группировке классов. Во-первых, можно группировать их по стереотипу (типу класса). Например, один пакет содержит классы-сущности предметной области, другой — классы пользовательского интерфейса и т.д. Этот подход может быть полезен с точки зрения размещения системы в среде реализации.

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

Если между любыми двумя классами, находящимися в разных пакетах, существует некоторая зависимость, то имеет место зависимость и между этими двумя пакетами. Таким образом, ***диаграмма пакетов*** (рис. 2.53) представляет собой диаграмму, содержащую пакеты классов и зависимости между ними. Строго говоря, пакеты и зависимости являются элементами

диаграммы классов, т.е. диаграмма пакетов — это форма диаграммы классов. Диаграммы пакетов можно считать основным средством управления общей структурой системы.

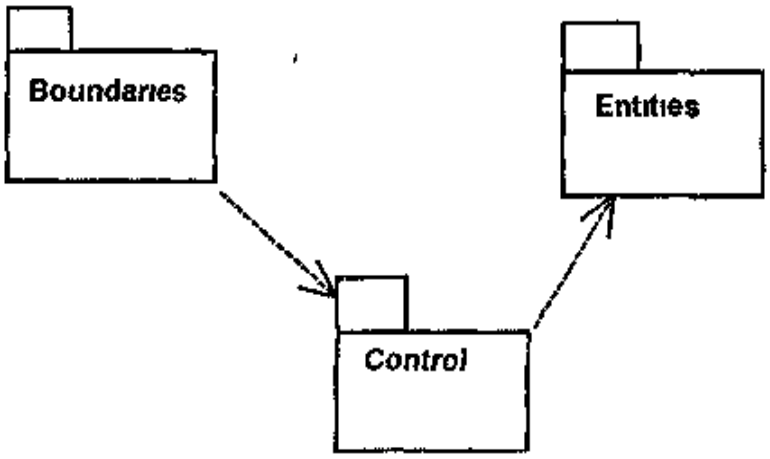

Рис. 2.53. Диаграмма пакетов

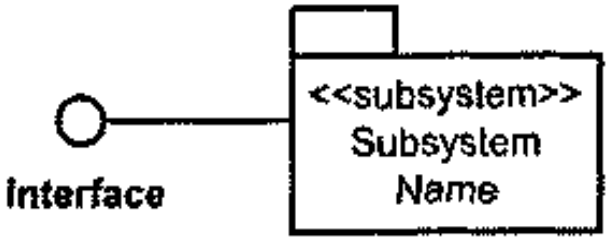

Рис. 2.54. Подсистема

Пакеты также используются для представления подсистем (модулей) системы (рис. 2.54). ***Подсистема*** — это комбинация пакета (поскольку она включает некоторое множество классов) и класса (поскольку она обладает поведением, т.е. реализует набор операций, которые определены в ее интерфейсах). Связь между подсистемой и интерфейсом называется ***связью реализации***. Подсистема используется для представления компонента в процессе проектирования.

2.5.4.

ДИАГРАММЫ СОСТОЯНИЙ

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. 2.55 приведен пример диаграммы состояний для банковского счета. Из данной диаграммы видно, в каких состояниях может существовать счет. Можно также видеть процесс перехода счета из одного состояния в другое. Например, если клиент требует закрыть открытый счет, он переходит в состояние «закрыт». Требование клиента называется событием (event), именно такие события и вызывают переход из одного состояния в другое.

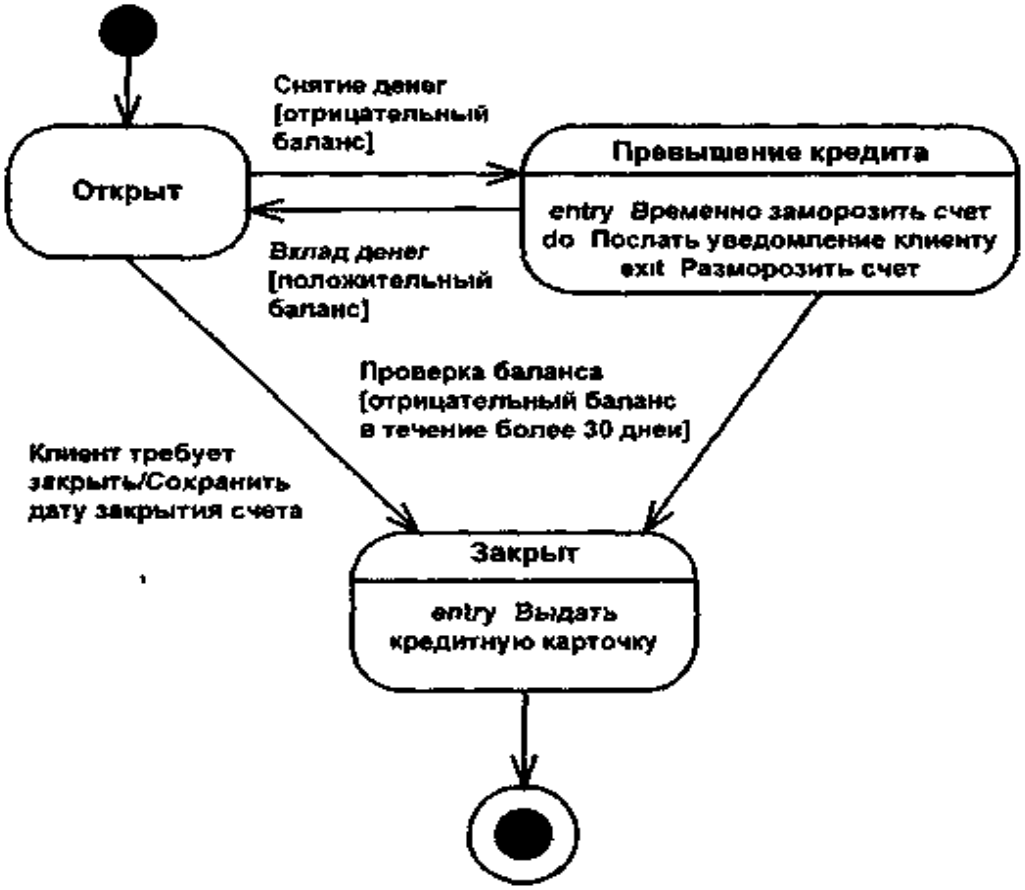

Рис. 2.55. Диаграмма состояний для класса Account

Если клиент снимает деньги с открытого счета, он может перейти в состояние «Превышение кредита». Это происходит, только если баланс по этому счету меньше нуля, что отражено условием [отрицательный баланс] на диаграмме. Заключенное в квадратных скобках ограничивающее условие (guard condition) определяет, когда может или не может произойти переход из одного состояния в другое.

На диаграмме имеются два специальных состояния — начальное (start) и конечное (stop). Начальное состояние выделено черной точкой, оно соответствует состоянию объекта, когда он только что был создан. Конечное состояние обозначается черной точкой в белом кружке, оно соответствует состоянию объекта непосредственно перед его уничтожением. На диаграмме состояний может быть только одно начальное состояние, а конечных состояний может быть столько, сколько нужно, или их может не быть вообще. Когда объект находится в каком-то конкретном состоянии, могут выполняться различные процессы. На рис. 2.55 при превышении кредита клиенту посылается соответствующее сообщение. Процессы, происходящие, когда объект находится в определенном состоянии, называются действиями (actions).

С состоянием можно связывать данные пяти типов: деятельность, входное действие, выходное действие, событие и история состояния. Рассмотрим каждый из них в контексте диаграммы состояний для класса Account банковской системы.

Деятельность (activity) — это поведение, реализуемое объектом, пока он находится в данном состоянии. Например, когда счет находится в состоянии «Закрыт», происходит возврат кредитной карточки пользователю. Деятельность — это прерываемое поведение. Оно может выполняться до своего завершения, пока объект находится в данном состоянии, или может быть прервано переходом объекта в другое состояние. Деятельность

изображают внутри самого состояния, ей должно предшествовать слово do (выполнять) и двоеточие.

Входное действие (entry action) — это поведение, которое выполняется, когда объект переходит в данное состояние. Когда счет в банке переходит в состояние «Превышение кредита», выполняется действие «Временно заморозить счет» независимо оттого, откуда объект перешел в это состояние. Таким образом, данное действие осуществляется не после того, как объект перешел в это состояние, а, скорее, как часть этого перехода. В отличие от деятельности входное действие рассматривается как непрерываемое.

Входное действие также показывают внутри состояния, ему предшествует слово entry (вход) и двоеточие.

Выходное действие (exit action) подобно входному, однако оно осуществляется как составная часть процесса выхода из данного состояния. Так, при выходе объекта Account из состояния «Превышение кредита» независимо оттого, куда он переходит, выполняется действие «Разморозить счет». Оно является частью процесса такого перехода. Как и входное, выходное действие является непрерываемым.

Выходное действие изображают внутри состояния, ему предшествует слово exit (выход) и двоеточие.

Переходом (transition) называется перемещение объекта из одного состояния в другое. На диаграмме все переходы изображают в виде стрелки, начинающейся на первоначальном состоянии и заканчивающейся на последующем.

Переходы могут быть рефлексивными. Объект может перейти и 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, основными из которых являются события, ограждающие условия и действия.

Событие (event) вызывает переход из одного состояния в другое. Событие «Клиент требует закрыть» вызывает переход счета из открытого в закрытое состояние. Событие размещают на диаграмме вдоль линии перехода.

На диаграмме для отображения события можно использовать как имя операции, так и обычную фразу, как в примере. Если нужно использовать операции, то событие «Клиент требует закрыть» можно было бы назвать RequestClosure().

Большинство переходов должны иметь события, так как именно они, прежде всего, заставляют переход осуществиться. Тем не менее, бывают и автоматические переходы, не имеющие событий. При этом объект сам перемещается из одного состояния в другое со скоростью, позволяющей осуществиться входным действиям, деятельности и выходным действиям.

Ограничивающие условия (guard conditions) определяют, когда переход может, а когда не может осуществиться. В примере событие «Вклад денег» переведет счет из состояния «Превышение кредита» в состояние «Открыт», но только если баланс будет больше нуля. В противном случае переход не осуществится. Ограничивающие условия изображают на диаграмме вдоль линии перехода после имени события, заключая их в квадратные скобки.

Ограничивающие условия задавать необязательно. Однако, если существует несколько автоматических переходов из состояния, необходимо определить для них взаимно исключающие ограждающие условия, чтобы понять, какой путь перехода будет автоматически выбран.

Действие может быть не только входным или выходным, но и частью перехода. Например, при переходе счета из открытого в закрытое состояние выполняется действие «Сохранить дату закрытия счета».

Действие изображают вдоль линии перехода после имени события, ему предшествует косая черта.

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

2.5.5.

ДИАГРАММЫ ДЕЯТЕЛЬНОСТИ

В отличие от большинства других средств UML диаграммы деятельности заимствуют идеи из нескольких различных методов, в частности из метода моделирования состояний SDL и сетей Петри. Эти диаграммы особенно полезны в описании поведения, включающего большое количество параллельных процессов. Диаграммы деятельности также полезны при параллельном программировании, поскольку можно графически изобразить все ветви и определить, когда их необходимо синхронизировать.

Диаграммы деятельности можно применять для описания потоков событий в вариантах использования. С помощью текстового описания можно достаточно подробно рассказать о потоке событий, но в сложных и запутанных потоках с множеством альтернативных ветвей будет трудно понять логику событий. Диаграммы деятельности предоставляют ту же информацию, что и текстовое описание потока событий, но в наглядной графической форме.

На рис. 2.56 приведена диаграмма деятельности для потока событий, связанного с системой бронирования авиабилетов. Рассмотрим ее нотацию.

Основным элементом диаграммы является ***деятельность (activity).*** Интерпретация этого термина зависит от той точки зрения, с которой строится диаграмма (это может быть некоторая задача, которую необходимо выполнить вручную или автоматизированным способом, или операция класса). Деятельность изображается в виде закругленного прямоугольника с текстовым описанием.

Любая диаграмма деятельности должна иметь начальную точку, определяющую

начало потока событий. Конечная точка необязательна. На диаграмме может быть несколько конечных точек, но только одна начальная.

На диаграмме могут присутствовать объекты и ***потоки объектов (object flow).*** Объект может использоваться или изменяться в одной из деятельностей. Показ объектов и их состояний (в дополнение к диаграммам состояний) помогает понять, когда и как происходит смена состояний объекта.

Объекты связаны с деятельностями через потоки объектов. Поток объектов отмечается пунктирной стрелкой от деятельности к изменяемому объекту или от объекта к деятельности, использующей объект.

На рис. 2.56 после ввода пользователем информации о кредитной карточке билет переходит в состояние «не подтвержден». Когда завершится процесс обработки кредитной карточки и будет подтвержден перевод денег, возникает деятельность «зарезервировать место», переводящая билет в состояние «приобретен»,

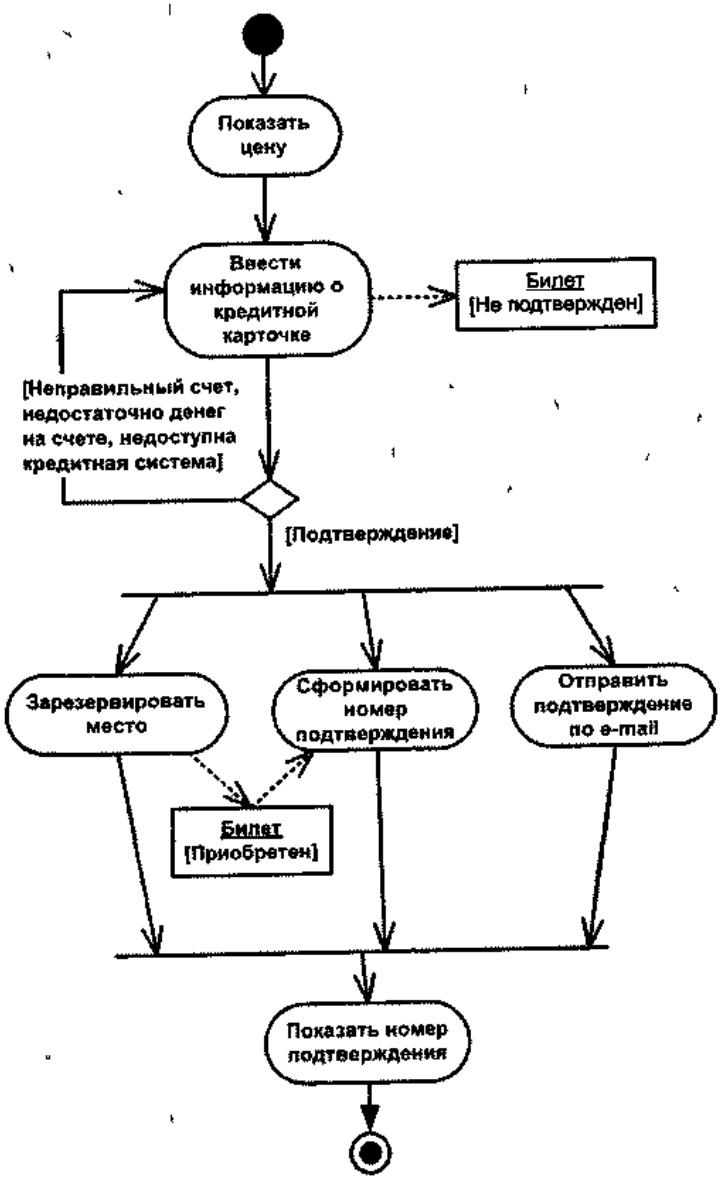

Рис. 2. 56. Диаграмма деятельности

и затем он используется в деятельности «формирование номера подтверждения». ***Переход*** (стрелка) показывает, как поток управления переходит от одной деятельности к другой. Если для перехода определено событие, то переход выполняется только после наступления такого события. Ограничивающие условия определяют, когда переход может, а когда не может осуществиться.

Если необходимо показать, что две или более ветвей потока выполняются параллельно, используются ***линейки синхронизации***. В данном примере параллельно выполняются резервирование места, формирование номера подтверждения и отправка почтового сообщения, а после завершения всех трех процессов пользователю выводится номер подтверждения.

Любая деятельность может быть подвергнута дальнейшей декомпозиции. Описание декомпозированной деятельности может быть представлено в виде другой диаграммы деятельности.

Подобно большинству других средств, моделирующих поведение, диаграммы деятельности отражают только вполне определенные его аспекты, поэтому их лучше всего использовать в сочетании с другими средствами.

Диаграммы деятельности предпочтительнее использовать в следующих ситуациях:

- анализ потоков событий в конкретном варианте использования. Здесь нас не интересует связь между действиями и объектами, а нужно только понять, какие действия должны иметь место и каковы зависимости в поведении системы. Связывание действий и объектов выполняется позднее с помощью диаграмм взаимодействия;
- анализ потоков событий в различных вариантах использования. Когда варианты использования взаимодействуют друг с другом, на диаграмме деятельности удобно представить и проанализировать все их потоки событий (в этом случае диаграмма с помощью вертикальных пунктирных линий разделяется на зоны — так называемые ***«плавательные дорожки» (swimlanes)***. В каждой зоне изображаются потоки событий одного из вариантов использования, а связи между разными потоками — в виде переходов или потоков объектов).

2.5.6.

ДИАГРАММЫ КОМПОНЕНТОВ

Диаграммы компонентов моделируют физический уровень системы. На них изображаются компоненты ПО и связи между 11 ими. На такой диаграмме обычно выделяют два типа компонентов: исполняемые компоненты и библиотеки кода.

Каждый класс модели (или подсистема) преобразуется в компонент исходного кода. Между отдельными компонентами изображают зависимости, соответствующие

зависимостям на этапе компиляции или выполнения программы.

На рис. 2.57 изображена одна из диаграмм компонентов для банковской системы.

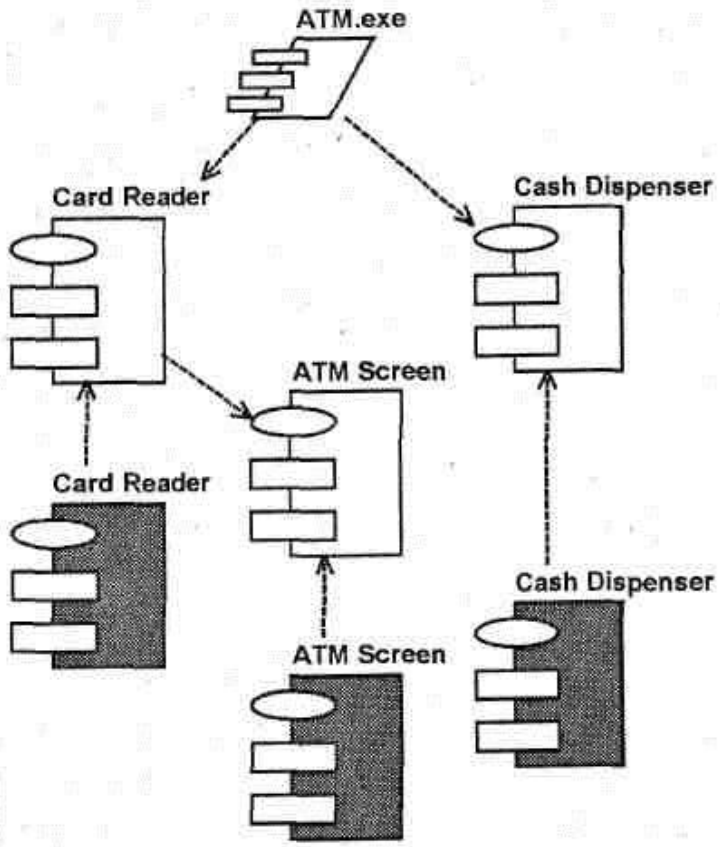

Рис. 2.57. Диаграмма компонентов для клиентской части системы

На этой диаграмме показаны компоненты клиентской части системы. В данном случае система разрабатывается на языке C++. У каждого класса имеется свой собственный заголовочный файл (файл с расширением .h) и файл тела класса (файл с расширением .cpp). Например, класс ATM Screen преобразуется в компоненты ATM Screen: тело и заголовок класса. Выделенный темным компонент называется спецификацией пакета (package specification) и соответствует файлу тела класса ATM Screen. Невыделенный компонент также называется спецификацией пакета, но соответствует заголовочному файлу класса. Компонент ATM.exe называется спецификацией задачи и моделирует поток управления (thread of processing) — исполняемую программу.

Компоненты соединены зависимостями. Например, класс Card Reader зависит от класса ATM Screen. Это означает, что, для того чтобы класс Card Reader мог быть

скомпилирован, класс ATM Screen должен уже существовать. После компиляции всех классов может быть создан исполняемый файл ATMClient.exe.

Банковская система содержит два потока управления и, таким образом, получаются два исполняемых файла. Один из них — 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. 2.58.

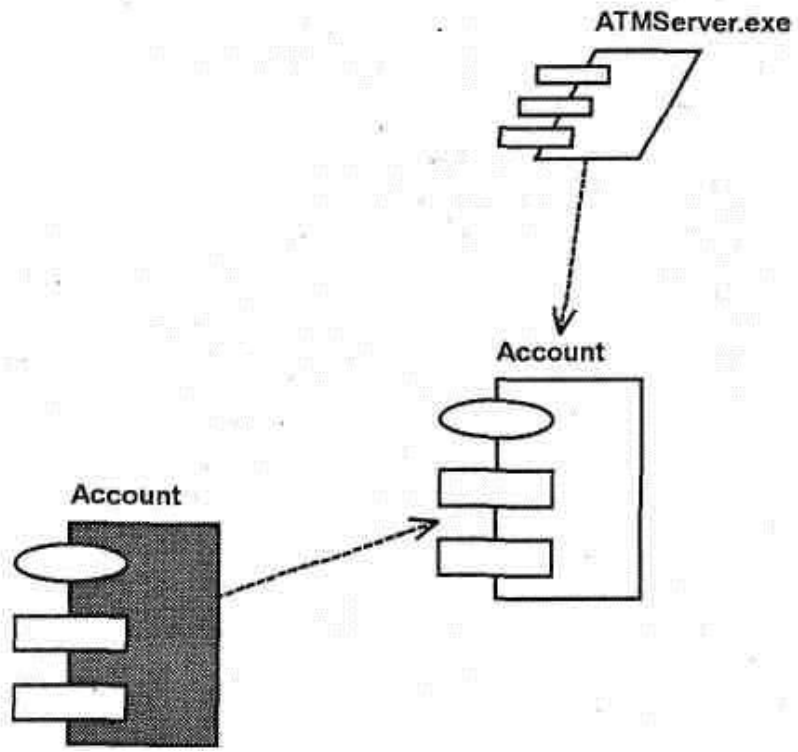

Рис. 2.58. Диаграмма компонентов для сервера

Как видно из примера, в модели системы может быть несколько диаграмм компонентов, в зависимости от числа подсистем или исполняемых файлов. Каждая подсистема является пакетом компонентов.

Диаграммы компонентов применяются теми участниками проекта, кто отвечает за компиляцию и сборку системы. Они нужны там, где начинается генерация кода.

2.5.7.

ДИАГРАММЫ РАЗМЕЩЕНИЯ

Диаграмма размещения отражает физические взаимосвязи между программными и аппаратными компонентами системы. Она является хорошим средством для того, чтобы показать размещение объектов и компонентов в распределенной системе.

Каждый узел на диаграмме размещения представляет собой некоторый тип вычислительного устройства — в большинстве случаев, часть аппаратуры. Эта аппаратура может быть простым устройством или датчиком, а может быть и мэйнфреймом.

Диаграмма размещения показывает физическое расположение сети и местонахождение в ней различных компонентов. Ее основные элементы:

- узел (node) — вычислительный ресурс — процессор или другое устройство (дисковая память, контроллеры различных устройств и т.д.). Для узла можно задать выполняющиеся на нем процессы;
- соединение (connection) — канал взаимодействия узлов (сеть).

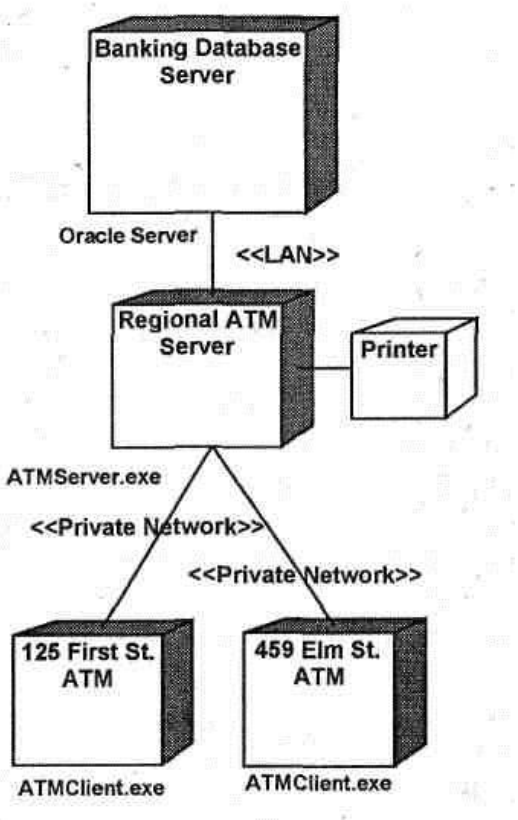

Рис. 2.59. Диаграмма размещения для банковской системы

Например, банковская система состоит из большого количества подсистем, выполняемых на отдельных физических устройствах, или узлах. Диаграмма размещения для такой системы показана на рис. 2.59.

Из данной 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. На нем будет работать ПО сервера. В свою очередь, посредством локальной сети региональный сервер будет сообщаться с сервером банковской базы данных, работающим под управлением Oracle. Наконец, 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.