# Оглавление

## УНИФИЦИРОВАННЫЙ ЯЗЫК МОДЕЛИРОВАНИЯ UML

Унифицированный язык моделирования UML (Unified Modeling Language) представляет собой язык для специфицирования, визуализации, 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 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, http://www.rational.com, http://www.uml.org.

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.

На рисунке ниже представлено применение диаграмм UML для моделирования системы программного обеспечения.


С 2011 года действует версия UML 2.4.1, которая принята в качестве международного стандарта ISO/IEC 19505-1, 19505-2. Последняя версия UML 2.5.1 принята в декабре 2017 года.

## 1. ДИАГРАММЫ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

Отдельные варианты использования могут применяться как для спецификации требований к проектируемой системе, так и для документирования процесса поведения имеющейся системы. Варианты использования неявно специфицируют требования, определяющие особенности взаимодействия пользователей с системой, что обеспечивает возможность корректной работы с предоставляемыми данной системой сервисами.

Требование (requirement) - желательное свойство, характеристика или условие, которым должна удовлетворять система в процессе своей эксплуатации.

Применительно к программным системам предложена следующая классификация требований, которая получила название модели FURPS+, что соответствует первым буквам соответствующих категорий требований на английском языке:

- Функциональные требования (Functionality) содержат основные свойства/функции системы..
- Требования удобства использования (Usability)
  - эстетика и логичность пользовательского интерфейса,
  - защита от человеческого фактора,
  - эксплуатационная документация, ее состав (руководства пользователей, администраторов и др.), отраслевые и гос. стандарты оформления,
  - квалификация пользователей и их обучение,
  - справочная информация в системе.
- Требования надежности (Reliability):
  - сбои:
    - допустимая частота/периодичность сбоев,
    - среднее время сбоев и их серьезность,
    - возможность восстановления системы после сбоев, в т.ч. возможность предварительного резервного копирования данных,
  - предсказуемость поведения,

    - время готовности системы к работе, режим работы или время доступности системы (например, «Система должна быть доступна 24 часа в сутки 7 дней в неделю»),
    - точность вычислений.
- Требования производительности (Performance):
    - скорость работы, время отклика системы,
    - результативность/эффективность,
    - пропускная способность, включая общее и допустимое количество одновременно работающих пользователей, количество пользовательских запросов, число обращений системы к БД и объем запрашиваемых/передаваемых данных в единицу времени,
    - время, необходимое на восстановление — скорость восстановления (необходимо отличать эту характеристику P/производительности от характеристик R/надежности «возможность восстановления» и «время доступности»),
    - время, необходимое для запуска и завершения работы — скорость запуска и завершения,
    - потребление ресурсов.
- Требования возможности сопровождения (Supportability):
    - тестирования,
    - расширения — наращивания дополнительного функционала системы,
    - масштабирования — тиражирования, например, в филиалах/подразделениях организации,
    - адаптации/приспособления к использованию в заданной среде, в т.ч. путем предварительной настройки,
    - конфигурирования — оперативной, регулярной настройки, переопределения параметров,
    - совместимости,
    - сопровождения, поддержки работоспособности: исправление ошибок, обновление данных, частота архивации и резервного копирования,
    - сервисного обслуживания и ремонта, их удобство,
    - установки,
    - локализации (например, «Продукт будет поддерживать несколько естественных языков»),
    - портативность,
    - соответствие международным стандартам.

При этом символом "+" обозначены дополнительные условия, к которым относятся:

- Проектные ограничения
    - ограничения на технологии (например, «Хранение необходимо реализовать с помощью реляционной БД»),
    - процесс («RUP»),
    - средства разработки («диаграммы должны создаваться в MS Visio, документация — в MS Word»),
    - прочие.
- Ограничения реализации, разработки, построение, написания программного кода:
    - стандарты разработки,
    - стандарты качества ПО, в т.ч. кода,
    - языки программирования (например, «Вся бизнес-логика должна быть реализована на языке Visual Basic»),
    - средства разработки («В качестве СУБД должна быть использована Oracle 10g»),
    - ресурсные ограничения,

    - лицензионные ограничения,
    - ограничения на техническое (аппаратное) обеспечение,
    - прочие.
- Требования к интерфейсам — ограничения (например, на форматы, протоколы) накладываемые необходимостью взаимодействия с другими системами:
    - форматы данных,
    - протоколы взаимодействия,
    - внешние системы,
    - требования к графическому интерфейсу пользователя
    - прочие.
- Физические требования: физические ограничения, накладываемые на технические (аппаратные) средства и окружение системы:
    - форма,
    - размер,
    - вес,
    - температурный режим,
    - влажность,
    - ограничения на вибрацию,
    - прочие.
- Юридические требования:
    - международные соглашения: единицы измерения, языки,
    - авторское право,
    - соглашения о лицензировании,
    - законодательство,
    - отраслевые стандарты.

Центральное место среди указанных требований занимают функциональные, которые специфицируют особенности реализации отдельных бизнес-процессов моделируемой системы. В контексте моделей языка UML именно функциональные требования должны служить исходной информацией для построения диаграмм вариантов использования. Однако графических средств языка UML на практике оказывается недостаточно для спецификации функциональных требований.

Следует отметить, что одним из требований языка UML является самодостаточность диаграмм для представления информации о моделях проектируемых систем. Однако большинство разработчиков и экспертов согласны с тем, что изобразительных средств языка UML явно не хватает для того, чтобы учесть на диаграммах вариантов использования особенности функционального поведения сложной системы. С этой целью рекомендуется дополнять этот тип диаграмм текстовыми сценариями, которые уточняют или детализируют последовательность действий, совершаемых системой при выполнении ее вариантов использования.

Сценарий (scenario) - определенная последовательность действий, которая описывает действия экторов и поведение моделируемой системы в форме обычного текста.

В контексте языка UML сценарий используется для дополнительной иллюстрации взаимодействия актеров и вариантов использования. Предлагаются различные способы представления или написания подобных сценариев. Один из таких шаблонов рассматривается ниже и может быть рекомендован читателям для применения на начальных этапах концептуального моделирования.

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[1].

***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.

Прецеденты обозначаются в виде эллипса, внутри которого указано его название. Прецеденты и действующие лица (экторы) соединяются с помощью линий. Часто на одном из концов линии изображают стрелку, причем направлена она к тому, у кого запрашивают сервис, другими словами, чьими услугами пользуются. Это простое объяснение иллюстрирует понимание прецедентов как сервисов, пропагандируемое компанией IBM.


Прецеденты могут включать другие прецеденты, расширяться ими, наследоваться и т. д.

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

Графически эктор изображается либо «человечком», либо символом класса с соответствующим стереотипом, как показано на рисунке. Обе формы представления имеют один и тот же смысл и могут использоваться в диаграммах. «Стереотипированная» форма чаще применяется для представления системных экторов или в случаях, когда эктор имеет свойства и их нужно отобразить.


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.

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

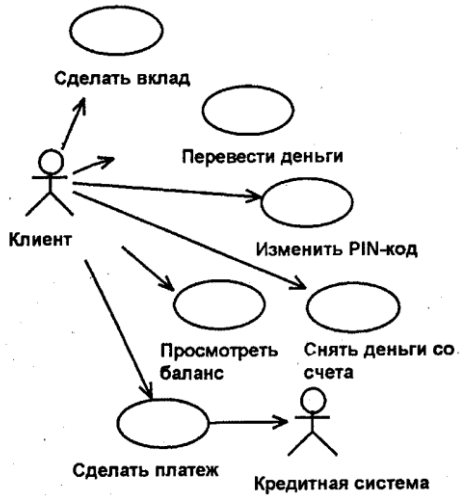



Рис. 2.47. Пример диаграммы вариантов использования

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, на 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 и 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.

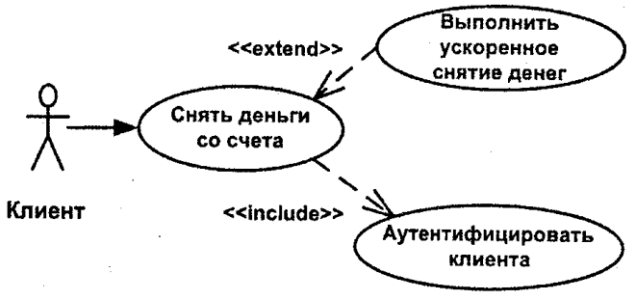



Рис. 2.48. Связи включения и расширения

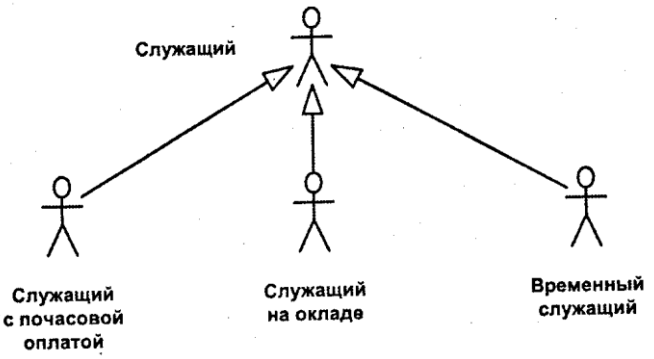



Рис. 2.49. Обобщение действующего лица

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 потенциальное 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. 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.

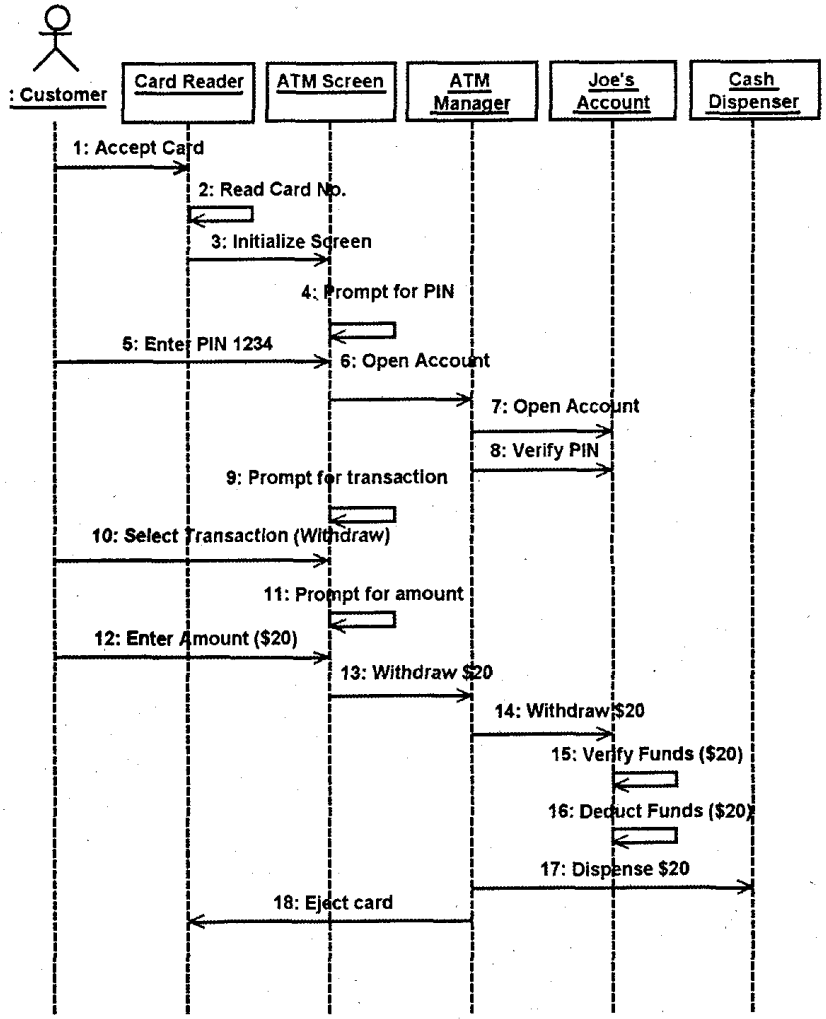



Рис. 2.50. Диаграмма последовательности

Все действующие лица показаны в верхней части диаграммы; в приведенном примере изображено 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.

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.

Подобно прочим диаграммам, диаграммы взаимодействия могут содержать примечания и ограничения.

**Объекты и их изображение на диаграмме последовательности**

На *диаграмме последовательности* изображаются *объекты*, которые непосредственно участвуют во взаимодействии. Никакие статические связи с другими *объектам* не визуализируются. Д*иаграммы последовательности* отображают динамику взаимодействия *объектов* во времени. Д*иаграмма последовательности* имеет два измерения. По оси X - слева направо изображаются *объекты*, участвующие во взаимодействии. Ось Y –направленная сверху вниз *диаграммы последовательности* - вертикальная временная ось.

Каждый *объект* графически изображается в форме прямоугольника и располагается в верхней части своей *линии жизни* (рис. 1). Внутри прямоугольника записываются собственное имя *объекта* со строчной буквы и *имя класса*, разделенные двоеточием. При этом вся *запись* подчеркивается, что является признаком *объекта*, который, как указывалось ранее, представляет собой экземпляр класса.

Для *объектов диаграммы последовательности* остаются справедливыми правила именования, рассмотренные ранее. Если на *диаграмме последовательности* отсутствует собственное имя *объекта*, то при этом должно быть указано *имя класса*. Такой *объект* считается **анонимным**. Может отсутствовать и *имя класса*, но при этом должно быть указано собственное имя *объекта*. Такой *объект* считается **сиротой**. Роль классов в именах *объектов* на *диаграммах последовательности*, как правило, не указывается.

Крайним слева на диаграмме изображается *объект* - инициатор моделируемого процесса взаимодействия (*объект* a на рис. 1). Правее - другой *объект*, который непосредственно взаимодействует с первым. Таким образом, порядок расположения *объектов* на *диаграмме последовательности* определяется исключительно соображениями удобства визуализации их взаимодействия друг с другом.

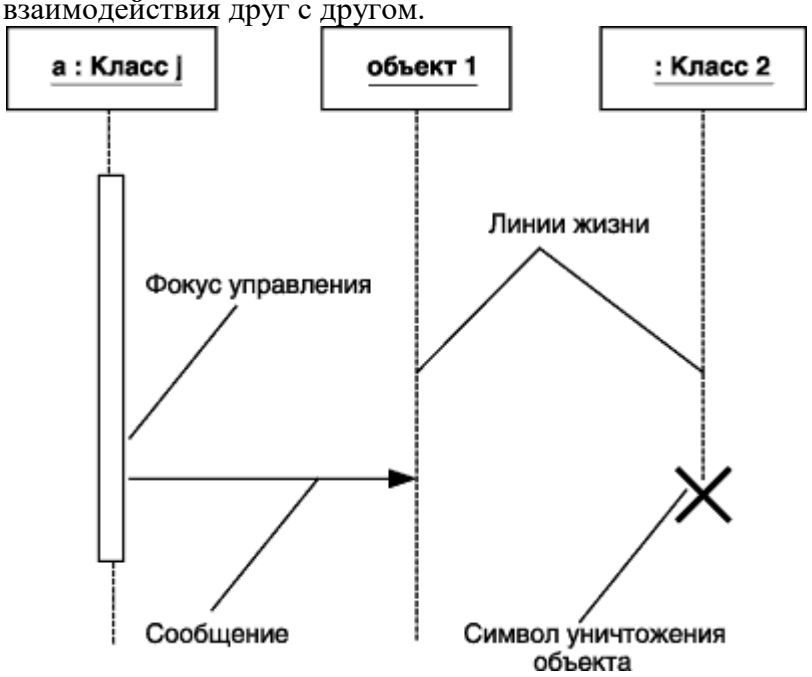


**Рис. 1.** Графические элементы диаграммы последовательности

Начальному моменту времени соответствует самая верхняя часть диаграммы. При этом процесс взаимодействия *объектов* реализуется посредством *сообщений*, которые посылаются одними *объектами* другим. *Сообщения* изображаются в виде горизонтальных стрелок с именем *сообщения* и образуют определенный порядок относительно времени своей инициализации. Другими словами, *сообщения*, расположенные на *диаграмме последовательности* выше, передаются раньше тех, которые расположены ниже. При этом масштаб на оси времени не указывается, поскольку *диаграмма последовательности* моделирует лишь временную упорядоченность взаимодействий типа "раньше-позже".

***Линия жизни объекта (object lifeline)*** - вертикальная линия на диаграмме последовательности, которая представляет существование объекта в течение определенного периода времени.

*Линия жизни объекта* 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 и его *линия жизни* должна продолжаться по всей рабочей области *диаграммы последовательности* от самой верхней ее части до самой нижней ( *объект* 1 и *объект* а Класса 1 на рис. 1).

Отдельные *объекты*, закончив выполнение своих операций, могут быть уничтожены, чтобы освободить занимаемые ими ресурсы. Для таких *объектов линия жизни* обрывается в момент его уничтожения. Для обозначения момента уничтожения *объекта* в языке *UML* применяется *специальный символ* в форме латинской буквы "X". На рис. 2 этот символ используется для уничтожения анонимного *объекта*, образованного от Класса 3. Ниже этого символа пунктирная линия не изображается, поскольку соответствующего *объекта* в системе уже нет, и этот *объект* должен быть исключен из всех последующих взаимодействий.

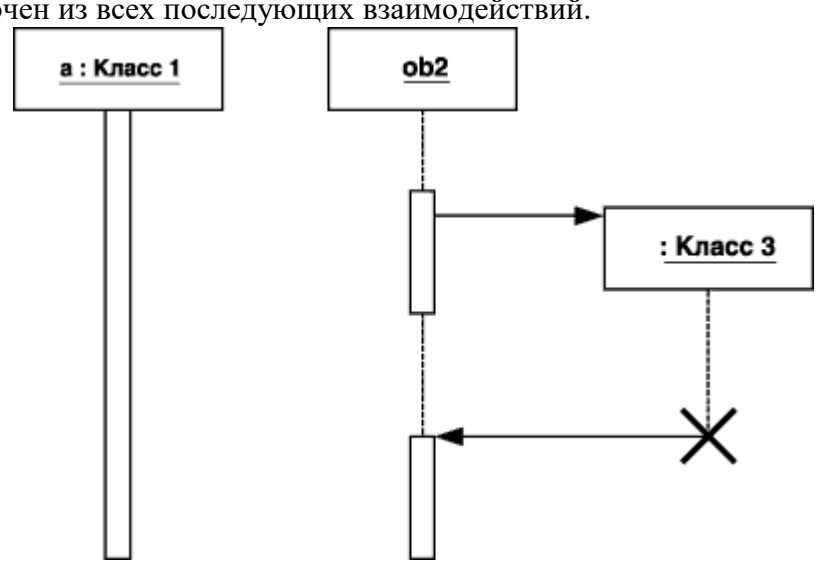



**Рис. 2.** Графическое изображение линий жизни и фокусов управления объектов

Вовсе не обязательно создавать все *объекты* в начальный момент времени. Отдельные *объекты* в системе могут создаваться по мере необходимости, существенно экономя ресурсы системы и повышая ее *производительность*. В этом случае *прямоугольник* такого *объекта* изображается не в верхней части *диаграммы последовательности*, а в той, которая соответствует моменту создания *объекта* (*анонимный объект*, образованный от Класса 3 на рис. 2). При этом *прямоугольник объекта* вертикально располагается в том месте диаграммы, которое по оси времени совпадает с моментом его возникновения в системе. *Объект* создается со своей *линией жизни* а, возможно, и с *фокусом управления*.

В процессе функционирования объектно-ориентированных систем одни *объекты* могут находиться в активном состоянии, непосредственно выполняя определенные действия, или в состоянии *пассивного ожидания сообщений* от других *объектов*. ***Фокус управления*** - символ,

применяемый для того, чтобы явно выделить подобную активность *объектов* на *диаграммах последовательности*.

***Фокус управления (focus of control)*** - специальный символ на диаграмме последовательности, указывающий период времени, в течение которого объект выполняет некоторое действие, находясь в активном состоянии.

*Фокус управления* изображается в форме вытянутого узкого прямоугольника (*объект* а на рис. 1), верхняя сторона которого обозначает начало получения *фокуса управления объекта* (начало активности), а ее нижняя сторона - окончание фокуса управления (окончание активности). Этот *прямоугольник* располагается ниже обозначения соответствующего *объекта* и может заменять его *линию жизни* (*объект* а на рис. 2), если на всем ее протяжении он активен.

Периоды активности *объекта* могут чередоваться с периодами его пассивности или ожидания. В этом случае у такого *объекта фокусы управления* изменяют свое изображение на *линию жизни* и наоборот ( *объект* сирота ob2 на рис. 2). Важно понимать, что получить *фокус управления* может только *объект*, у которого в этот момент имеется *линия жизни*. Если же *объект* был уничтожен, то вновь возникнуть в системе он уже не может. Вместо него может быть создан лишь экземпляр этого же класса, который, строго говоря, будет другим *объектом*.

В отдельных случаях инициатором взаимодействия в системе может быть *актер* или внешний *пользователь*. При этом *актер* изображается на *диаграмме последовательности* самым первым объектом слева со своим *фокусом управления* (рис. 3). Наиболее часто *актер* и его *фокус управления* будут существовать в системе постоянно, отмечая характерную для пользователя *активность* в инициировании взаимодействий с системой. *Актер* может иметь собственное имя либо оставаться анонимным.

В отдельных случаях *объект* может посылать *сообщения* самому себе, инициируя так называемые рефлексивные *сообщения*. Для этой цели служит специальное изображение (*сообщение* у *объекта* а на рис. 3). Такие *сообщения* изображаются в форме *сообщения*, начало и конец которого соприкасаются с *линией жизни* или *фокусом управления* одного и того же *объекта*. Подобные ситуации возникают, например, при обработке нажатий на клавиши клавиатуры при вводе текста в редактируемый документ, при наборе цифр номера телефона абонента.

Если в результате рефлексивного *сообщения* создается новый *подпроцесс* или нить управления, то говорят о рекурсивном или вложенном *фокусе управления*. На *диаграмме последовательности рекурсия* обозначается небольшим прямоугольником, присоединенным к правой стороне *фокуса управления* того *объекта*, для которого изображается данное рекурсивное взаимодействие (*анонимный объект* Класса 2 на рис. 3).


**Рис. 3.** Графическое изображение актера, рефлексивного сообщения и рекурсии на диаграмме последовательности

**Сообщения на диаграмме последовательности**

Все *сообщения* на *диаграмме последовательности* упорядочены по времени своей передачи в

моделируемой системе, хотя номера у них могут не указываться.

На *диаграммах последовательности* могут присутствовать три разновидности *сообщений*, каждое из которых имеет свое графическое изображение (рис. 4).

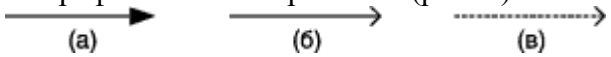



**Рис. 4.** Графическое изображение различных видов сообщений между объектами на диаграмме последовательности

Первая разновидность *сообщения* (рис. 4, а) наиболее распространена и используется для вызова процедур, выполнения операций или обозначения отдельных вложенных потоков управления. Начало этой стрелки, как правило, соприкасается с *фокусом управления* того объекта-клиента, который инициирует это *сообщение*. Конец стрелки соприкасается с *линией жизни* того *объекта*, который принимает это *сообщение* и выполняет в ответ определенные действия. При этом принимающий *объект* может получить *фокус управления*, становясь в этом случае активным. Передающий *объект* может потерять *фокус управления* или остаться активным.

Вторая разновидность *сообщения* (рис. 4, б) используется для обозначения простого асинхронного сообщения, которое передается в произвольный момент времени. Передача такого *сообщения* обычно не сопровождается получением *фокуса управления* объектом-получателем.

Третья разновидность *сообщения* (рис. 4, в) используется для возврата из вызова процедуры. Примером может служить простое *сообщение* о завершении вычислений без предоставления результата расчетов объекту-клиенту. В процедурных потоках управления эта стрелка может быть опущена, поскольку ее наличие неявно предполагается в конце активизации *объекта*. В то же время считается, что каждый вызов процедуры имеет свою пару - возврат вызова. Для непроцедурных потоков управления, включая параллельные и асинхронные *сообщения*, стрелка возврата должна указываться явным образом.

Обычно *сообщения* изображаются горизонтальными стрелками, соединяющими *линии жизни* или *фокусы управления* двух *объектов* на *диаграмме последовательности*. При этом неявно предполагается, что время передачи *сообщения* достаточно мало по сравнению с процессами выполнения действий *объектами*. Считается также, что за время передачи *сообщения* с соответствующими *объектами* не может произойти никаких событий. Другими словами, состояния *объектов* не изменяются. Если же это предположение не может быть признано справедливым, то стрелка *сообщения* изображается под наклоном, так чтобы конец стрелки располагался ниже ее начала.

Каждое *сообщение* на *диаграмме последовательности* ассоциируется с определенной операцией, которая должна быть выполнена принявшим его *объектом*. При этом операция может иметь аргументы или параметры, значения которых влияют на получение различных результатов. Соответствующие *параметры операции* будет иметь и вызывающее это действие *сообщение*. Более того, значения параметров отдельных *сообщений* могут содержать условные выражения, образуя *ветвление* или *альтернативные* пути основного потока управления.

**Ветвление потока управления**

Одна из особенностей *диаграммы последовательности* - возможность визуализировать простое *ветвление* процесса. Для изображения *ветвления* используются две или более стрелки, выходящие из одной точки *фокуса управления объекта* (*объект* ob1 на рис. 5). При этом рядом с каждой из них должно быть явно указано соответствующее условие ветви в форме булевского выражения.

Количество ветвей может быть произвольным, однако наличие *ветвлений* может существенно усложнить интерпретацию *диаграммы последовательности*. Предложение-условие должно быть явно указано для каждой ветви и записывается в форме обычного текста, псевдокода или выражения языка программирования. Это *выражение* всегда должно возвращать некоторое *булевское выражение*. *Запись* этих условий должна исключать одновременную передачу альтернативных *сообщений* по двум и более ветвям. В противном случае на *диаграмме последовательности* может возникнуть *конфликт ветвления*.

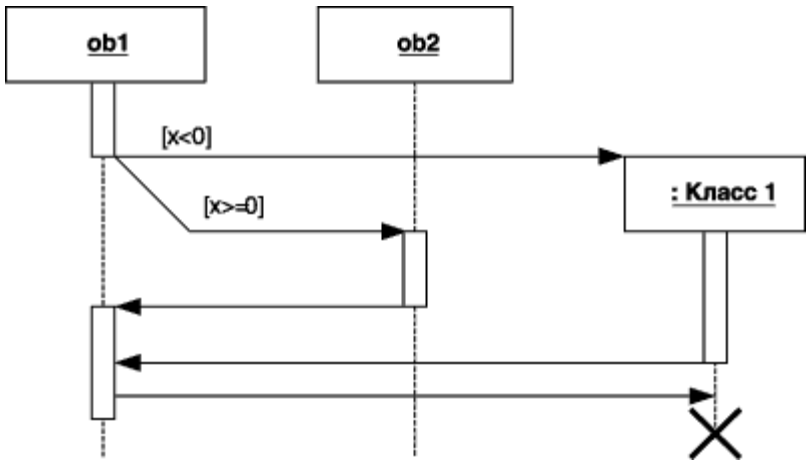



**Рис. 5.** Графическое изображение бинарного ветвления потока управления на диаграмме последовательности

С помощью *ветвления* можно изобразить и более сложную логику взаимодействия *объектов* между собой ( *объект* ob1 на рис. 6). Если условий более двух, то для каждого из них необходимо предусмотреть ситуацию единственного выполнения. Описанный ниже пример относится к моделированию взаимодействия программной системы обслуживания клиентов в банке. В этом примере *диаграммы последовательности объект* ob1 вызывает выполнение действий у одного из трех других *объектов*.

*Условием ветвления* может служить сумма снимаемых клиентом средств со своего *текущего счета*. Если эта сумма превышает 1500$, то могут потребоваться дополнительные действия, связанные с созданием и последующим разрушением *объекта* Класса 1. Если же сумма превышает 100$, но не превышает 1500$, то вызывается операция или процедура объекта ob3. И, наконец, если сумма не превышает 100$, то вызывается операция или процедура *объекта* ob2. При этом *объекты* ob1, ob2 и ob3 постоянно существуют в системе. Последний *объект* создается от Класса 1 только в том случае, если справедливо первое из альтернативных условий. В противном случае он может быть никогда не создан.

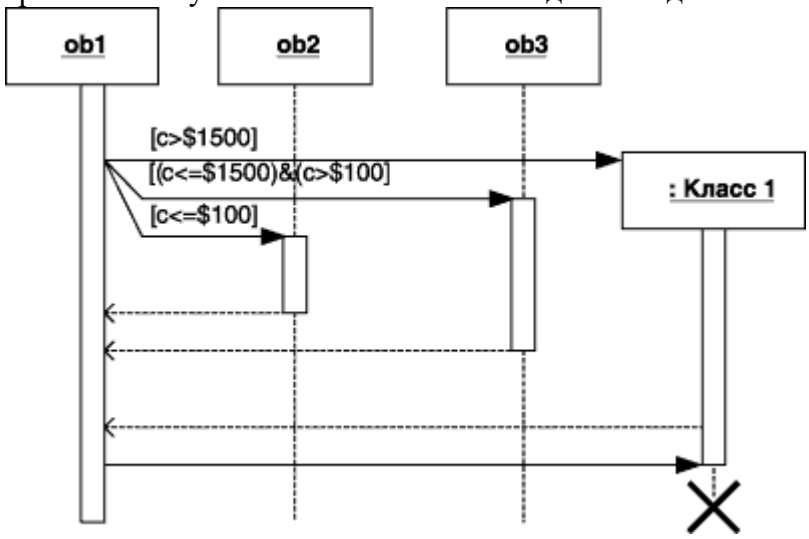



**Рис. 6.** Графическое изображение тернарного ветвления потока управления на диаграмме последовательности

*Объект* ob1 имеет постоянный *фокус управления*, а все остальные *объекты* - получают *фокус управления* только для выполнения ими соответствующих операций.

На *диаграммах* последовательности при записи *сообщений* также могут использоваться стереотипы. Их *семантика* и *синтаксис* остаются без изменения, как они определены в нотации языка *UML*. Ниже представлена *диаграмма последовательности* для описанного выше случая *ветвления*, дополненная стереотипными значениями отдельных *сообщений* (рис. 7). Очевидно,

эта *диаграмма последовательности* является более выразительной и простой для своей содержательной интерпретации.

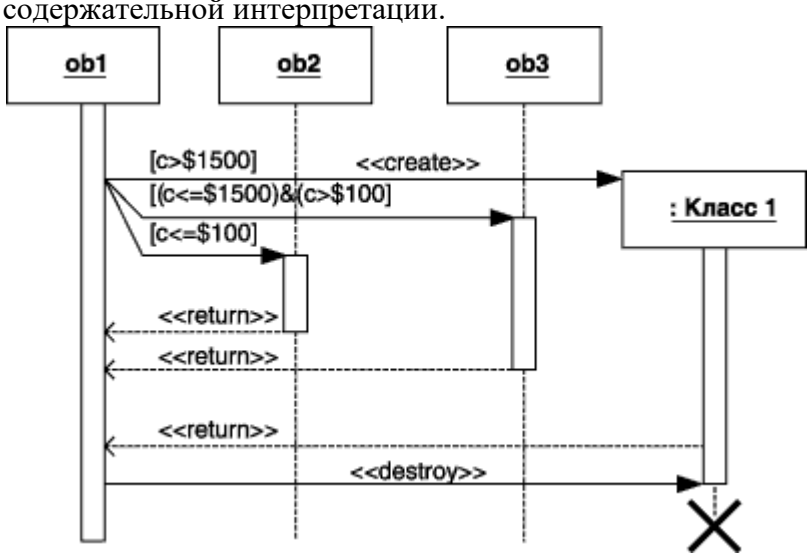



**Рис. 7.** Диаграмма последовательности со стереотипными значениями сообщений

Как уже отмечалось ранее, *сообщения* могут иметь собственное имя, в качестве которого выступает имя *операции*, вызов которой инициируют эти *сообщения* у принимающего *объекта*. В этом случае рядом со стрелкой записывается имя *операции* с круглыми скобками, в которых могут указываться параметры или аргументы соответствующей *операции*. Если параметры отсутствуют, то скобки после имени *операции* все равно должны быть изображены.

**Рекомендации по построению диаграмм последовательности**

Построение *диаграммы последовательности* целесообразно начинать с выделения из всей совокупности классов только тех, *объекты* которых участвуют в моделируемом взаимодействии. После этого все *объекты* наносятся на диаграмму, с соблюдением порядка инициализации *сообщений*. Здесь необходимо установить, какие *объекты* будут существовать постоянно, а какие временно - только на период выполнения ими требуемых действий.

Когда *объекты* визуализированы, можно приступать к спецификации *сообщений*. При этом необходимо учитывать те *операции*, которые имеют классы соответствующих *объектов* в модели системы. При необходимости уточнения этих операций следует использовать их стереотипы. Для уничтожения *объектов*, которые создаются на *время выполнения* своих действий, нужно предусмотреть явное *сообщение*. Наиболее простые случаи *ветвления* процесса взаимодействия можно изобразить на одной диаграмме с использованием соответствующих графических примитивов. В более сложных случаях для моделирования каждой ветви управления может потребоваться отдельная *диаграмма последовательности*. Следует помнить, что каждый альтернативный *поток* управления затрудняет понимание построенной модели.

Общим правилом является *визуализация* особенностей реализации каждого варианта использования на отдельной *диаграмме последовательности*. В этой ситуации отдельные диаграммы должны рассматриваться совместно как одна модель взаимодействия. Необходимость синхронизации сложных потоков управления, как правило, требуют введение в модель дополнительных ограничений. При этом общая *запись* таких ограничений должна следовать семантике *языка объектных ограничений OCL*.

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

Диаграмма кооперации акцентирует внимание на организации объектов, принимающие участие во взаимодействии. Для создания диаграммы кооперации нужно расположить участвующие во взаимодействии объекты в виде вершин графа. Затем связи, соединяющие эти объекты, изображаются в виде дуг этого графа. Наконец, связи дополняются сообщениями, которые объекты принимают и посылают. Это дает пользователю ясное визуальное представление о потоке управления в контексте структурной организации кооперирующихся

объектов.

У диаграмм кооперации есть два свойства, которые отличают их от диаграмм Последовательностей.

Первое - это путь. Для описания связи одного объекта с другим мы устанавливаем между ними отношение ассоциации с помощью прямой линии.

Второе свойство - это порядковый номер сообщения. Для обозначения временной последовательности перед сообщением можно поставить номер (нумерация начинается с единицы), который должен постепенно возрастать для каждого нового сообщения (2, 3 и т.д.). Для обозначения вложенности используется десятичная нотация Дьюи (1 - первое сообщение; 1.1- первое сообщение, вложенное в сообщение 1; 1.2 - второе сообщение, вложенное в сообщение 1 и т.д.). Уровень вложенности не ограничен. Для каждой связи можно показать несколько сообщений (вероятно, посылаемых разными отправителями), и каждое из них должно иметь уникальный порядковый номер.

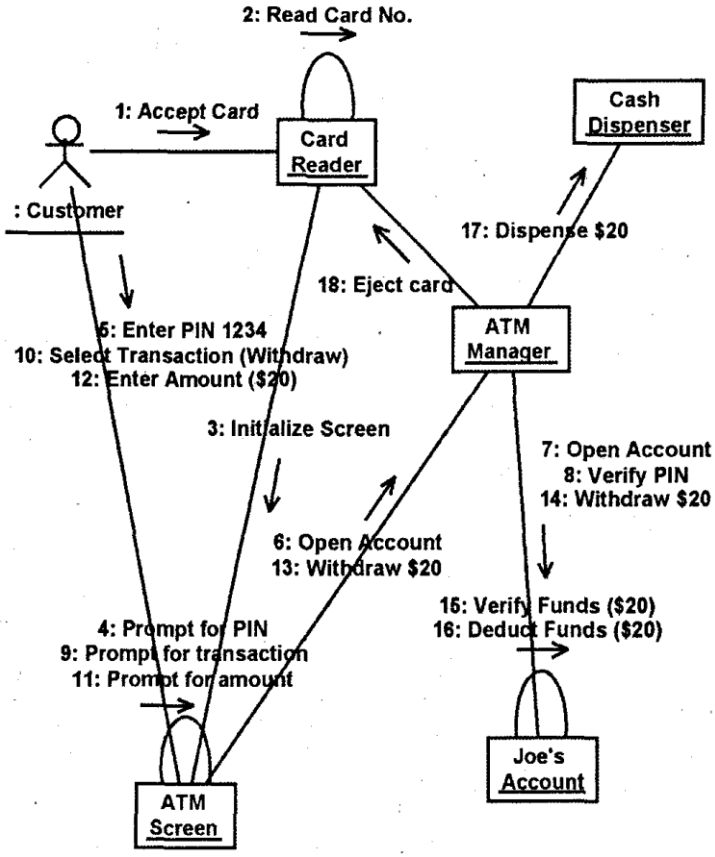



Рис. 2.51. Кооперативная диаграмма

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.

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.

**Советы**

Создавая диаграммы взаимодействий в UML, помните, что и диаграммы последовательностей, и диаграммы кооперации являются проекциями динамических аспектов системы на одну и ту же модель. Ни одна диаграмма взаимодействия, взятая в отдельности, не может охватить все динамические аспекты. Для моделирования динамики системы в целом, равно как и ее подсистем, операций, классов, прецедентов и кооперации, лучше использовать сразу несколько диаграмм взаимодействий.

Хорошо структурированная диаграмма взаимодействия обладает следующими свойствами:

- акцентирует внимание только на одном аспекте динамики системы;
- содержит только такие прецеденты и актеры, которые важны для понимания этого аспекта;
- содержит только такие детали, которые соответствуют данному уровню абстракции, и только те дополнения, которые необходимы для понимания системы;
- не настолько лаконична, чтобы ввести читателя в заблуждение относительно важных аспектов семантики.

При изображении диаграммы взаимодействий следует пользоваться нижеприведенными рекомендациями:

- дайте ей имя, соответствующее ее назначению;
- используйте диаграмму последовательностей, если хотите подчеркнуть временную упорядоченность сообщений, и диаграмму кооперации - если хотите подчеркнуть структурную организацию участвующих во взаимодействии объектов;
- расположите элементы так, чтобы свести к минимуму число пересечений;
- пространственно организуйте элементы так, чтобы семантически близкие сущности на диаграмме располагались рядом;
- используйте примечания и цвет, чтобы привлечь внимание читателя к важным особенностям диаграммы;
- не злоупотребляйте ветвлениями. Сложные ветвления лучше показывать на диаграммах деятельности.

## 3. ДИАГРАММЫ КЛАССОВ

Диаграммы классов при моделировании объектно-ориентированных систем встречаются чаще других. На таких диаграммах показывается множество классов, интерфейсов, кооперации и отношений между ними.

Диаграммы классов используются для моделирования статического вида системы с точки зрения проектирования. Сюда по большей части относится моделирование словаря системы, кооперации и схем. Кроме того, диаграммы классов составляют основу еще двух диаграмм - компонентов и развертывания.

Диаграммы классов важны не только для визуализации, специфицирования и документирования структурных моделей, но также для прямого и обратного проектирования исполняемых систем.

Диаграммой классов (Class diagram) называют диаграмму, на которой показано Множество классов, интерфейсов, кооперации и отношений между ними. Ее изображают в виде множества вершин и дуг.

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.

На этой диаграмме присутствуют четыре класса: Card Reader (устройство для чтения карточек), Account (счет), ATM Screen (экран АТМ) и Cash Dispenser (кассовый аппарат).

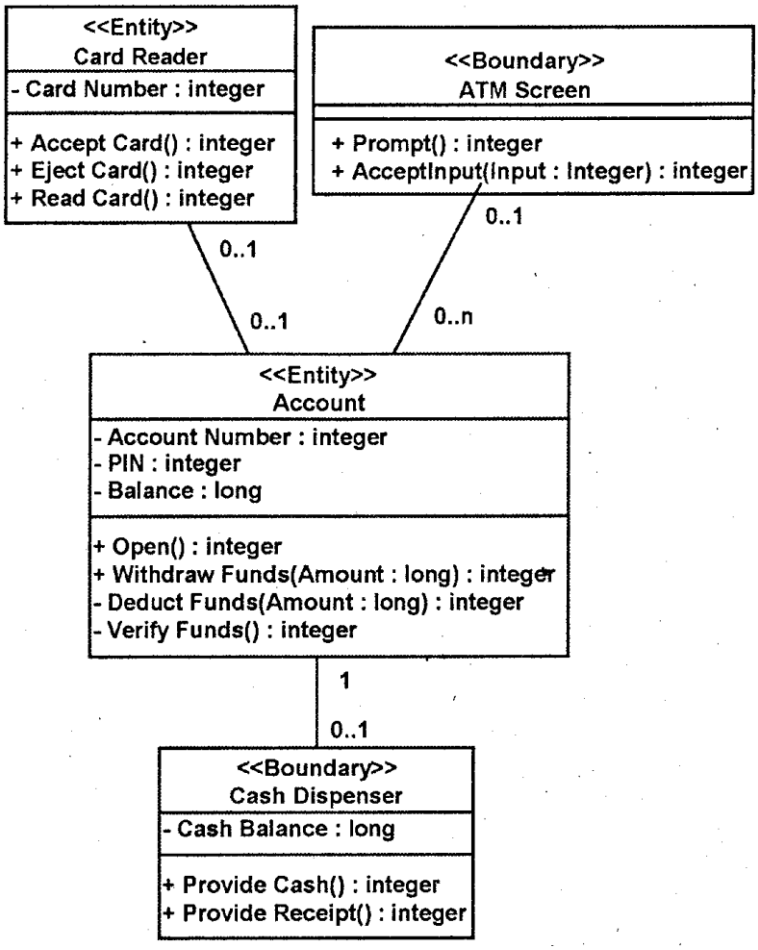



Рис. 2.52. Диаграмма классов для варианта использования «Снять деньги со счета»

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 позже.

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.

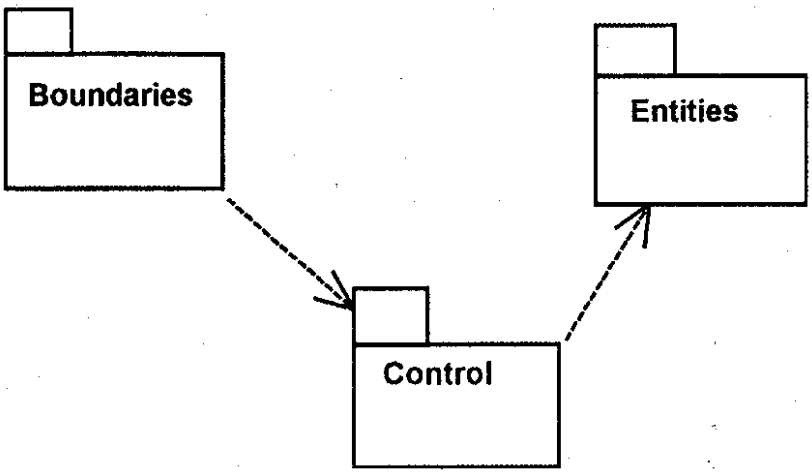



Рис. 2.53. Диаграмма пакетов

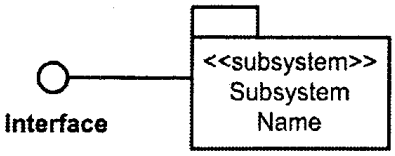



Рис. 2.54. Подсистема

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.

***Типичные примеры применения***

Диаграммы классов применяют для моделирования статического вида системы с точки зрения проектирования. В этом представлении удобнее всего описывать функциональные требования к системе - услуги, которые она предоставляет конечному пользователю.

Обычно диаграммы классов используются в следующих целях:

- для моделирования словаря системы. Моделирование словаря системы предполагает принятие решения о том, какие абстракции являются частью системы, а какие - нет. С помощью диаграмм классов вы можете определить эти абстракции и их обязанности;
- для моделирования простых кооперaций. Кооперация - это сообщество классов, интерфейсов и других элементов, работающих совместно для обеспечения некоторого кооперативного поведения, более значимого, чем сумма составляющих его элементов. Например, моделируя семантику транзакций в распределенной системе, вы не сможете понять происходящие процессы, глядя на один-единственный класс, поскольку соответствующая семантика обеспечивается несколькими совместно работающими классами. С помощью диаграмм классов удается визуализировать и специфицировать эти классы и отношения между ними;
- для моделирования логической схемы базы данных. Логическую схему можно представлять себе как чертеж концептуального проекта базы данных. Во многих сферах деятельности требуется хранить устойчивую (persistent) информацию в реляционной или объектно-ориентированной базе данных. Моделировать схемы также можно с помощью диаграмм классов.

## 4. ДИАГРАММЫ СОСТОЯНИЙ (statechart diagram)

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.

*Объекты характеризуются поведением и состоянием*, в котором находятся. Например, человек может быть новорожденным, младенцем, ребенком, подростком или взрослым. Другими словами, объекты что-то делают и что-то "знают". **Диаграммы состояний** *применяются для того, чтобы объяснить, каким образом работают сложные объекты*. Несмотря на то, что смысл понятия "состояние" интуитивно понятен, все же приведем его определение:

**Состояние (state)** - ситуация в жизненном цикле объекта, во время которой он удовлетворяет некоторому условию, выполняет определенную деятельность или ожидает какого-то события. Состояние объекта определяется значениями некоторых его атрибутов и присутствием или отсутствием связей с другими объектами.

**Диаграмма состояний**— это, по существу, диаграмма состояний из теории автоматов со стандартизированными условными обозначениями, которая может определять множество систем от компьютерных программ до бизнес-процессов.

Диаграмма состояний показывает, как объект переходит из одного состояния в другое. Очевидно, что диаграммы состояний служат для моделирования динамических аспектов системы (как и диаграммы последовательностей, прецедентов и, как мы увидим далее, диаграммы деятельности). Диаграмма состояний полезна при моделировании жизненного цикла объекта (как и ее частная разновидность - диаграмма деятельности, о которой мы будем говорить далее).

От других диаграмм диаграмма состояний отличается тем, что описывает процесс изменения состояний только одного экземпляра определенного класса - одного объекта, причем объекта *реактивного*, то есть объекта, поведение которого характеризуется его реакцией на внешние события. Понятие жизненного цикла применимо как раз к реактивным объектам, настоящее состояние (и поведение) которых обусловлено их прошлым состоянием. Но диаграммы состояний важны не только для описания динамики отдельного объекта. Они могут использоваться для *конструирования исполняемых систем* путем прямого и обратного проектирования.

Но поговорим об обозначениях на диаграммах состояний. Используются следующие условные обозначения:

- Круг, обозначающий начальное состояние, в котором находится объект сразу после его создания.
- Окружность с маленьким кругом внутри, обозначающая конечное состояние (если есть), которое объект не может покинуть, если перешел в него.
- Скруглённые прямоугольники, обозначающие состояния, через которые проходит объект в течение своего жизненного цикла. Верхушка прямоугольника содержит название состояния. В середине может быть горизонтальная линия, под которой записываются активности, происходящие в данном состоянии.
- Стрелка, обозначающая переход между состояниями, которые вызваны выполнением методов описываемого диаграммой объекта. Название события (если есть), вызывающего переход, отмечается рядом со стрелкой. Ограничивающее условие может быть добавлено перед «/» и заключено в квадратные скобки (***название_события*[ограничивающее_условие]**), что значит, что это выражение должно быть истинным, чтобы переход имел место. Если при переходе производится какое-то

действие, то оно добавляется после «/» (***название_события*[ограничивающее_условие]/действие**).

- Толстая горизонтальная линия с либо множеством входящих линий и одной выходящей, либо одной входящей линией и множеством выходящих. Это обозначает объединение и разветвление соответственно.

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.

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).

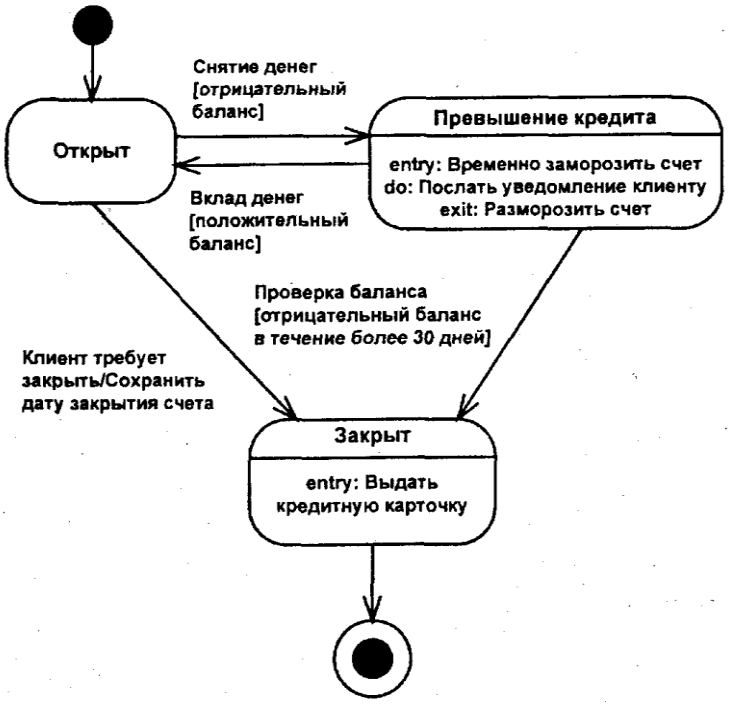



Рис. 2.55. Диаграмма состояний для объекта класса Account

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 деятельностью (***activity***).

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 «Превышение кредита», посылается

уведомление клиенту. ***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 от того, 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 класса 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 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, 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.

## 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 (objectflow)***. 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», и затем он используется в деятельности «формирование номера подтверждения».

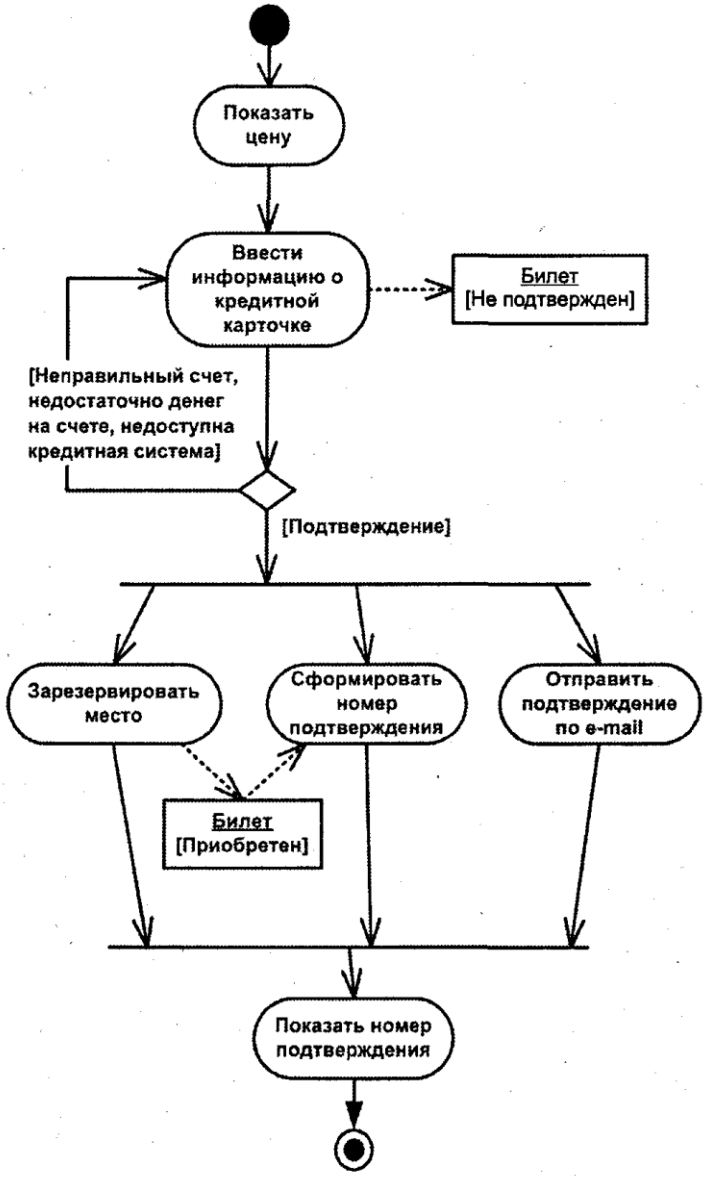



Рис. .56. Диаграмма деятельности

***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).

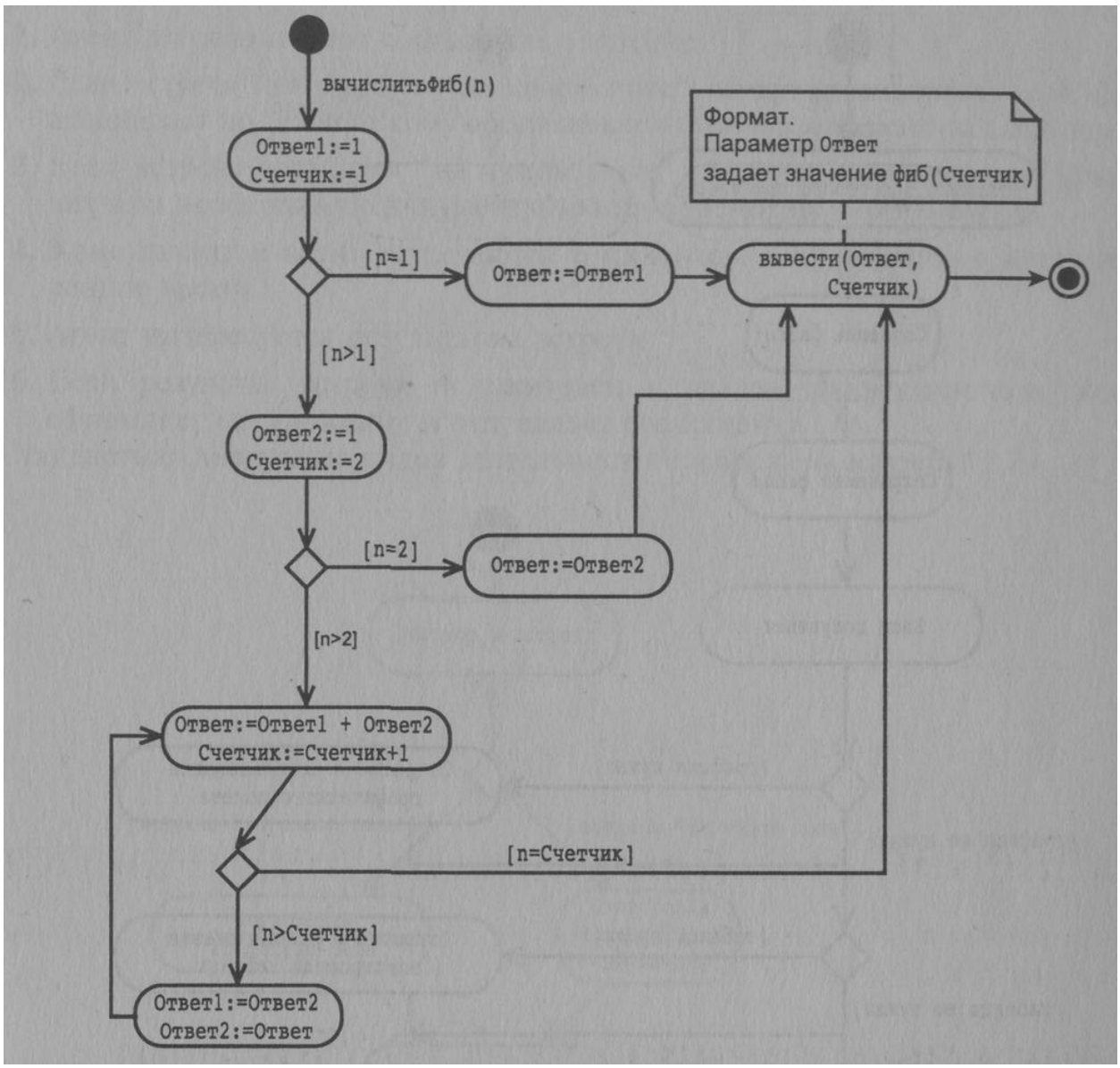
В качестве примера применения диаграмм видов деятельности построим диаграмму для одной операции. Такой операцией будет вычисление числа Фибоначчи по заданному номеру.

**Пример. Операция вычисления чисел Фибоначчи**

Ряд чисел 1, 1, 2, 3, 5, 8, 13, ... называется числами Фибоначчи, по имени средневекового математика, жившего около 800 лет назад. Каждое число называется "фибом". Таким образом, первый "фиб" или "фиб(1)" — это 1, "фиб(2)" — 1, "фиб(3)" — 2 и т.д. Ряд строится на основе того, что каждый последующий "фиб" является суммой двух предыдущих, т.е. фиб(8)=21.

Допустим, что один из наших классов является вычислительным устройством, а одной из его операций является вычисление чисел Фибоначчи и вывод их на печать. Назовем эту операцию вычислитьФиб(n). Построим диаграмму видов деятельности для моделирования этой операции.

Для этого понадобится несколько переменных и счетчик, показывающий, дошел ли процесс вычислений до n-ого "фиба". Необходима переменная для хранения текущего результата вычислений и еще две переменные для хранения двух "фибов", которые будут суммироваться. На рис. изображена диаграмма видов деятельности для такой операции.


вычислитьФиб(n)
Ответ1:=1
Счетчик:=1
Формат.
Параметр Ответ
задает значение фиб(Счетчик)
[n=1]
Ответ:=Ответ1
вывести(Ответ,
Счетчик)
[n>1]
Ответ2:=1
Счетчик:=2
[n=2]
Ответ:=Ответ2
[n>2]
Ответ:=Ответ1 + Ответ2
Счетчик:=Счетчик+1
[n=Счетчик]
[n>Счетчик]
Ответ1:=Ответ2
Ответ2:=Ответ

На диаграмме видов деятельности можно отразить роли участников процесса. Для этого диаграмму разбивают вертикальными пунктирными линиями на сегменты, напоминающие плавательные дорожки. В верхней части каждой дорожки указывается название роли. На дорожке отображаются виды деятельности для каждой роли. Между дорожками могут быть переходы. На рис. изображена версия диаграммы видов деятельности для процесса подготовки встречи с новым клиентом с "плавательными дорожками".

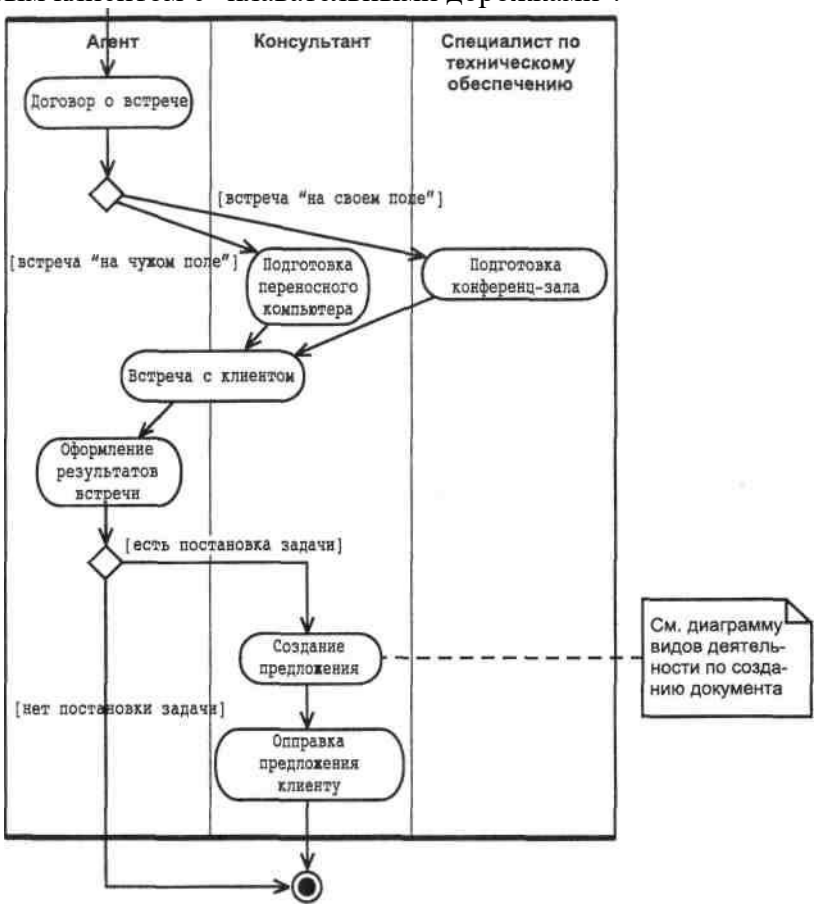



Рис. Указание ролей на диаграмме видов деятельности

На обеих диаграммах для процесса подготовки встречи с новым клиентом создание предложения описывается как вид деятельности. В каждом случае для этого вида деятельности можно сделать ссылку на диаграмму видов деятельности по созданию документа.

## 6. ДИАГРАММЫ КОМПОНЕНТОВ

Диаграммы компонентов моделируют физический уровень системы. На них изображаются компоненты ПО и связи между ними. 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.

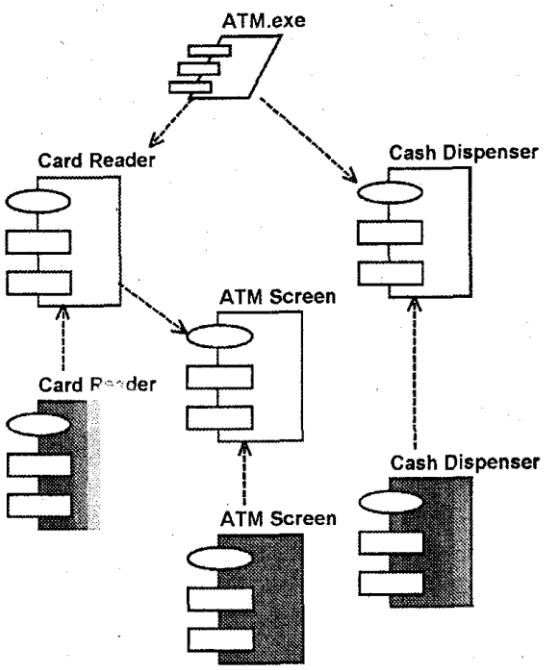



Рис. 2.57. Диаграмма компонентов для клиентской части системы

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.

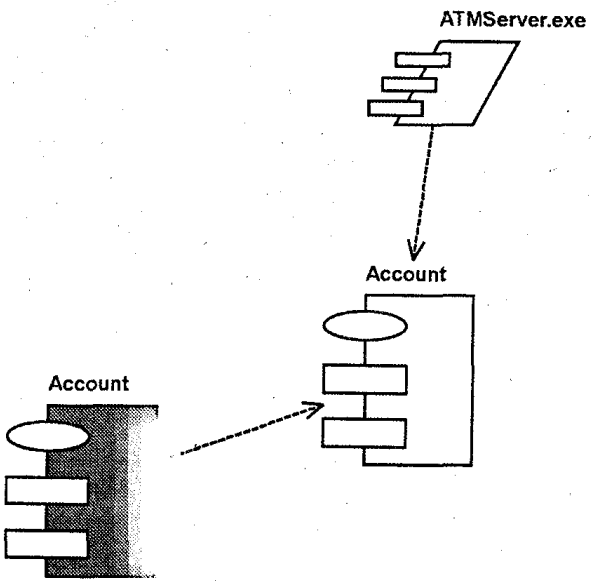



Рис. 2.58. Диаграмма компонентов для сервера

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.

## 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,

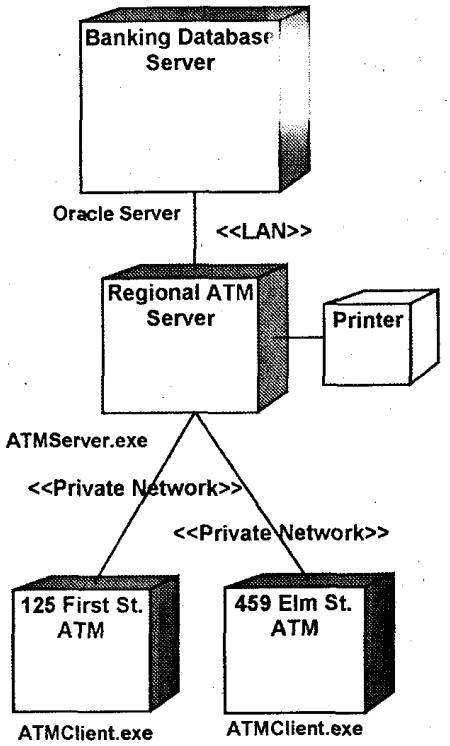



Рис. 2.59. Диаграмма размещения для банковской системы

контроллеры различных устройств и т.д.). Для узла можно задать выполняющиеся на нем процессы;

- соединение (connection) - канал взаимодействия узлов (сеть).

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.

## 8. МЕХАНИЗМЫ РАСШИРЕНИЯ UML

Механизмы расширения UML предназначены для того, чтобы разработчики могли адаптировать язык моделирования к своим конкретным нуждам, не меняя при этом его метамодель. Наличие механизмов расширения принципиально отличает UML от таких средств моделирования, как IDEF0, IDEF1X, IDEF3, DFD и ERM. Перечисленные языки моделирования можно определить как сильно типизированные (по аналогии с языками программирования), поскольку они не допускают произвольной интерпретации семантики элементов моделей. UML, допуская такую интерпретацию (в основном за счет стереотипов), является слабо типизированным языком. К его механизмам расширения относятся:

• стереотипы;

• тегированные (именованные) значения;

• ограничения.

***Стереотип*** — это новый тип элемента модели, который определяется на основе уже существующего элемента. Стереотипы расширяют нотацию модели, могут применяться к любым

элементам модели и представляются в виде текстовой метки (см. рис. 2.52) или пиктограммы (иконки).

Стереотипы классов — это механизм, позволяющий разделять классы на категории. Например, основными стереотипами, используемыми в процессе анализа системы, являются: Boundary (граница), Entity (сущность) и Control (управление).

***Граничными классами (boundary classes)*** называются такие классы, которые расположены на границе системы и всей окружающей среды. Они включают все формы, отчеты, интерфейсы с аппаратурой (такой, как принтеры или сканеры) и интерфейсы с другими системами.

***Классы-сущности (entity classes)*** отражают основные понятия (абстракции) предметной области и, как правило, содержат хранимую информацию. Обычно для каждого класса-сущности создают таблицу в базе данных.

***Управляющие классы (control classes)*** отвечают за координацию действий других классов. Обычно у каждого варианта использования имеется один управляющий класс, контролирующий потоки событий этого варианта использования. Управляющий класс отвечает за координацию, но сам не несет в себе никакой функциональности - остальные классы не посылают ему большого количества сообщений. Вместо этого он сам посылает множество сообщений. Управляющий класс просто делегирует ответственность другим классам, по этой причине его часто называют классом-менеджером.

В системе могут быть и другие управляющие классы, общие для нескольких вариантов использования. Например, класс SecurityManager (менеджер безопасности) может отвечать за контроль событий, связанных с безопасностью. Класс TransactionManager (менеджер транзакций) занимается координацией сообщений, относящихся к транзакциям с базой данных. Могут быть и другие менеджеры для работы с другими элементами функционирования системы, такими, как разделение ресурсов, распределенная обработка данных или обработка ошибок.

Помимо упомянутых стереотипов, разработчики ПО могут создавать собственные наборы стереотипов, формируя тем самым специализированные подмножества UML (например, для описания бизнес-процессов, Web-приложений, баз данных и т.д.). Такие подмножества (наборы стереотипов) в стандарте языка UML носят название **профилей языка**.

***Именованное значение*** — это пара строк «тег = значение», или «имя — содержимое», в которых хранится дополнительная информация о каком-либо элементе системы, например, время создания, статус разработки или тестирования, время окончания работы над ним и т.п.

***Ограничение*** — это семантическое ограничение, имеющее вид текстового выражения на естественном или формальном языке (OCL - Object Constraint Language), которое невозможно выразить с помощью нотации UML.

Одних графических средств, таких как диаграмма классов или диаграмма состояний, недостаточно для разработки точных и непротиворечивых моделей сложных систем. Существует необходимость задания дополнительных ограничений, которым должны удовлетворять различные компоненты или объекты модели. Традиционно подобные ограничения описывались с помощью естественного языка. Однако, как показала практика системного моделирования, такие сформулированные на естественном языке ограничения страдают неоднозначностью и нечеткостью.

Для преодоления этих недостатков и придания рассуждениям строгого характера были разработаны различные формальные языки. Однако традиционные формальные языки требуют для своего конструктивного использования знания основ математической логики и теории формального вывода. С одной стороны, это делает формальные языки естественным средством для построения математических моделей, а, с другой стороны, существенно ограничивает круг потенциальных пользователей, поскольку разработка формальных моделей требует специальной квалификации.

Другая особенность традиционных формальных языков заключается в присущей им бедной семантике. Базовые элементы формальных языков имеют слишком абстрактное содержание, что затрудняет их непосредственную интерпретацию в понятиях моделей конкретных технических систем и бизнес-процессов. Поскольку процесс объектно-ориентированного анализа и

проектирования направлен на построение конструктивных моделей сложных систем, которые должны быть реализованы в форме программных систем и аппаратных комплексов, это является серьезным недостатком. Необходимо применение такого формального языка, базовые элементы которого адекватно отражают семантику основных конструкций объектной модели.

Именно для этих целей и был разработан язык OCL. По своей сути он является формальным языком, более простым для изучения, чем традиционные формальные языки. В то же время язык OCL специально ориентирован на описание бизнес-процессов и бизнес-логики, поскольку был разработан в одном из отделений корпорации IBM.

OCL представляет собой формальный язык для описания ограничений, которые могут быть использованы при определении различных компонентов языка UML. Хотя язык OCL является частью языка UML, в общем случае он может иметь гораздо более широкую область приложений, чем конструкции языка UML. Средства языка OCL позволяют специфицировать не только объектные ограничения, но и другие выражения логико-лингвистического характера, такие, как предусловия, постусловия и ограничивающие условия. Этот язык ничего не изменяет в графических моделях, а только дополняет их. Это означает, что выражения языка OCL никогда не могут изменить состояние системы, хотя эти выражения и могут быть использованы для спецификации самого процесса изменения этого состояния. Выражения языка OCL также не могут изменять отдельные значения атрибутов и операций для объектов и их связей. Всякий раз, когда оценивается одно из таких выражений, на выходе получается лишь некоторое значение.

Язык OCL не является языком программирования в обычном смысле, поскольку не позволяет записать логику выполнения программы или последовательность управляющих действий. Средства этого языка не предназначены для описания процессов вычисления выражений, а только лишь фиксируют необходимость выполнения тех или иных условий применительно к отдельным компонентам моделей. Главное назначение языка OCL - гарантировать справедливость ограничений для отдельных объектов модели. В этом смысле OCL можно считать языком моделирования, который вовсе не обязан допускать строгую реализацию в инструментальных средствах.

Язык OCL может быть использован для решения следующих задач:

- описание инвариантов классов и типов в модели классов;
- описание пред- и постусловий в операциях и методах;
- описание ограничивающих условий элементов модели;
- навигация по структуре модели;
- спецификация ограничений на операции.

Описание языка OCL отличается от описаний традиционных формальных языков менее формальным характером. Базовым элементом языка OCL является выражение, которое строится по определенным правилам. При этом допускается расширение конструкций языка за счет включения в его состав дополнительных типов.

В модели выражение используется для записи некоторых условий, которым должны удовлетворять все экземпляры соответствующего классификатора. В этом случае говорят, что выражение служит для представления некоторых инвариантных свойств соответствующих элементов модели.

## 9. КОЛИЧЕСТВЕННЫЙ АНАЛИЗ ДИАГРАММ UML

Методика количественной оценки и сравнения диаграмм UML основана на присвоении элементам диаграмм оценок, зависящих от их информационной ценности, а также от вносимой ими в диаграмму дополнительной сложности. Ценность отдельных элементов меняется в зависимости от типа диаграммы, на которой они находятся.

Количественную оценку диаграммы можно провести по следующей формуле:

$$S = ( \sum S_{Obj} + \sum S_{Lnk} ) / ( 1 + Obj + \sqrt{} ( T_{Obj} + T_{Lnk} )) ,$$

где S — оценка диаграммы, $S_{Obj}$ — оценки для элементов диаграммы, $S_{Lnk}$ - оценки для связей на диаграмме, Obj — число объектов на диаграмме, $T_{0bj}$ — число типов объектов на диаграмме, $T_{Lnk}$ -число типов связей на диаграмме.

Если диаграмма содержит большое число связей одного типа (например, модель предметной области), то число и тип связей можно не учитывать, и формула расчета приводится к виду:

$$S = (\sum S_{Obj} ) / ( 1 + Obj + \sqrt{} T_{Obj}) .$$

Если на диаграмме классов показаны атрибуты и операции классов, можно учесть их при расчете, при этом оценка прибавляется к оценке соответствующего класса:

$$S_{cls} = (\sqrt{} Op + \sqrt{} Atr ) / 0{,}3*( Op + Atr ),$$

где $S_{cls}$ – оценка операций и атрибутов для класса, $Op$ – число операций у класса, $Atr$ – число атрибутов у класса. При этом учитываются только атрибуты и операции, отображенные на диаграмме.

Далее приводятся оценки для различных типов элементов и связей.

**Основные элементы языка UML**

| Тип элемента | Оценка для элемента |
|---|---|
| Класс | 5 |
| Интерфейс | 4 |
| Вариант использования | 2 |
| Компонент | 4 |
| Узел (node) | 3 |
| Процессор | 2 |
| Взаимодействие | 6 |
| Пакет | 4 |
| Состояние | 4 |
| Примечание | 2 |

**Основные типы связей языка UML**

| Тип связи | Оценка для связи |
|---|---|
| Зависимость | 2 |
| Ассоциация | 1 |
| Агрегация | 2 |
| Композиция | 3 |
| Обобщение | 3 |
| Реализация | 2 |

Остальные типы связей должны рассматриваться как ассоциации.

Недостатком диаграммы является как слишком низкая оценка (при этом диаграмма недостаточно информативна), так и слишком высокая оценка (при этом диаграмма обычно слишком сложна для понимания). Далее приведены диапазоны оптимальных оценок для основных типов диаграмм.

Диапазоны оценок для диаграмм UML

| Тип диаграммы | Диапазон оценок |
|---|---|
| Классов – с атрибутами и операциями | 5–5,5 |
| Классов – без атрибутов и операций | 3–3,5 |
| Компонентов | 3,5–4 |
| Вариантов использования | 2,5–3 |
| Размещения | 2–2,5 |
| Последовательности | 3–3,5 |
| Кооперативная | 3,5–4 |
| Пакетов | 3,5–4 |
| Состояний | 2,5–3 |