2.7. Визуальное моделирование

2.7.1. Визуальное моделирование. Метод Буча, OMT и UML

Собираясь сделать пристройку к дому, Вы, вероятно, не начнёте с того, что купите кучу досок и будете сколачивать их вместе различными способами, пока не получите что-то приблизительно подходящее. Для работы понадобятся проекты и схемы, так что Вы скорее всего начнёте с планирования и структурирования пристройки. Тогда результат будет долговечнее, и дом не разрушится от небольшого дождика.

В мире программного обеспечения то же самое делается с помощью моделирования. Модели представляют собой проекты систем. Проект дома позволяет планировать его до начала непосредственной работы, ***модель*** даёт возможность спланировать систему до того, как вы приступите к её созданию. Это гарантирует, что проект удастся, требования будут учтены и система сможет выдержать «ураган» последующих изменений.

Визуальным моделированием называется процесс графического представления модели с помощью некоторого стандартного набора графических элементов. Наличие стандарта необходимо для реализации одного из преимуществ визуального моделирования - коммуникации. Общение между пользователями, разработчиками, аналитиками, тестировщиками, менеджерами и всеми остальными участниками проекта является основной целью визуального моделирования. Общение можно обеспечить с помощью невизуальной (текстовой) информации, но люди легче понимают сложную информацию, если она представлена визуально, нежели описана в тексте. Создавая визуальную модель системы, мы можем показать её работу на различных уровнях – можно моделировать взаимодействие между пользователем и системой, взаимодействие объектов внутри системы, а также взаимодействие между различными системами.

Созданные модели представляются всем заинтересованным сторонам, которые могут извлечь из них ценную информацию. Например, глядя на модель, пользователи визуализируют своё взаимодействие с системой. Аналитики увидят взаимодействие между объектами модели. Разработчики поймут, какие объекты нужно создать и что эти объекты должны делать. Тестировщики визуализируют взаимодействие между объектами, что позволит им построить тесты. Менеджеры увидят как всю систему в целом, так и взаимодействие её частей. Наконец, руководители информационной службы, глядя на высокоуровневые модели, поймут, как взаимодействуют друг с другом системы в их организации. Таким образом, визуальные модели предоставляют

мощный инструмент, позволяющий показать разрабатываемую систему всем заинтересованным сторонам.

Составление диаграмм - это еще не анализ и не проектирование. Диаграммы лишь помогают описать поведение системы (для анализа) или показать детали архитектуры (для проектирования). Будущая система сначала формируется в сознании разработчика, и только когда система будет понятна в общих чертах, она будет записана. Однако хорошо продуманная и выразительная система обозначений очень важна для разработки. Во-первых, общепринятая система позволяет разработчику описать сценарий или сформулировать архитектуру и доходчиво изложить свои решения коллегам. Символ транзистора понятен всем электронщикам мира. Аналогично, над проектом жилого дома, разработанным архитекторами в Санкт-Петербурге, строителям из Сан-Франциско скорее всего не придется долго ломать голову, решая, как надо расположить двери, окна или электрическую проводку. Во-вторых, как подметил Уайтхед в своей основополагающей работе по математике, «освобождая мозг от лишней работы, хорошая система обозначений позволяет ему сосредоточиться на задачах более высокого порядка» [3]. В-третьих, четкая система обозначений позволяет автоматизировать большую часть утомительной проверки на полноту и правильность. Кроме этого CASE-средства, используя принципы визуального моделирования, облегчают процесс разработки ПО. Они позволяют автоматизировать процесс создания и поддержания моделей в целостном состоянии, а также генерировать по разработанным моделям «скелетный» код системы.

Важный вопрос визуального моделирования – выбор графической нотации для описания различных аспектов системы. Нотация должна быть понятна всем заинтересованным сторонам, иначе модель будет бесполезна. Среди нотаций, которые используются объектно-ориентированной методологией, поддержку получили метод Буча, технология объектного моделирования (OMT, Object Modeling Technology) и унифицированный язык моделирования (UML, Unified Modeling Language). Однако большинством производителей и такими комитетами по стандартам, как ANSI и Object Management Group (OMG), был принят стандарт UML.

Метод Буча назван по имени его изобретателя Гради Буча (Grady Booch), работающего в корпорации Rational Software руководителем по науке (Chief Scientist). Он написал несколько книг, в которых обсуждаются необходимость и преимущества визуального моделирования, и разработал нотацию графических символов для описания различных аспектов модели. Объекты в его модели представляются в виде облаков, иллюстрируя то, что они могут быть почти чем угодно. Нотация Буча предусматривает также несколько видов стрелок для отображения разных типов отношений между объектами.

Нотация OMT была разработана Джеймсом Рамбо (Dr. James Rumbaugh), написавшим несколько книг о системном анализе и проектировании. В книге «System Analysis and Design» он рассматривает значимость моделирования систем с помощью компонентов (объектов) реального мира. Предложенная им нотация OMT получила широкое признание, ее поддерживают такие стандартные промышленные инструменты моделирования программного обеспечения, как Rational Rose и Select OMT. OMT использует более простую графику моделирования систем по сравнению с методом Буча.

Унифицированный язык моделирования (UML) является результатом совместных усилий Гради Буча, Джеймса Рамбо, Ивара Якобсона (Ivar Jacobson), Ребекки Вирс-Брок (Rebecca Wirfs-Brock), Питера Йордона (Peter Yourdon) и многих других. Якобсон первым описал процесс выявления и фиксации требований к системе в виде совокупностей транзакций, называемых вариантами использования (use case). Якобсон также разработал метод проектирования систем под названием «Объектно-ориентированное проектирование программного обеспечения» (Object Oriented Software Engineering, OOSE), концентрирующий внимание на анализе. Буч, Рамбо и Якобсон, о которых обычно говорят как о «трёх друзьях» (three amigos), работают в корпорации Rational Software. Их деятельность связана в основном со стандартизацией и усовершенствованием языка UML. Символы UML сильно напоминают нотации Буча и OMT, но содержат также элементы из других нотаций.

Процесс консолидации методов, вошедших в состав UML, начался в 1993 г. Каждый из трёх авторов этого языка – Буч, Рамбо и Якобсон – начал внедрять в свои разработки идеи остальных двух авторов. Такая унификация методологий продолжалась до 1995 г., когда появилась версия 0.8 Унифицированного Метода (Unified Method). В 1996 г. Унифицированный Метод был ратифицирован и передан в группу Object Technology Group, после чего его начали адаптировать у себя многие основные производители программного обеспечения. Наконец, 14 ноября 1997 г. группа OMG объявила язык UML 1.1 промышленным стандартом.

UML позволяет создавать несколько типов визуальных диаграмм, которые иллюстрируют различные аспекты системы:

1) Диаграммы Вариантов Использования;
2) Диаграммы Последовательности;
3) Кооперативные диаграммы;
4) Диаграммы Классов;
5) Диаграммы Состояний;
6) Диаграммы Компонентов;
7) Диаграммы Размещения.

Далее диаграммы UML рассматриваются подробнее.

2.7.2. Диаграммы Вариантов Использования

Диаграмма Вариантов Использования позволяет наглядно представлять требования к системе. Этот тип диаграмм описывает общую функциональность системы. Все участники проекта, изучая диаграммы Вариантов Использования, могут понять, что система должна делать. Диаграмма Вариантов Использования содержит варианты использования системы, действующих лиц и связи между ними. ***Вариант использования*** (use case) - это функции, выполняемые системой. ***Действующее лицо*** (actor) - это все, что взаимодействует системой (люди или системы, получающие или передающие информацию в данную систему). ***Связи*** позволяют показать способы взаимодействия между элементами системы. На рис. 2.33 приведен пример диаграммы Вариантов Использования автоматического банкомата (ATM). На ней показаны три действующих лица: клиент, банковский служащий и кредитная система. Существуют также шесть основных действий, выполняемых моделируемой системой: перевести деньги, положить деньги на счет, снять деньги со счета, показать баланс, изменить идентификационный номер и произвести оплату.

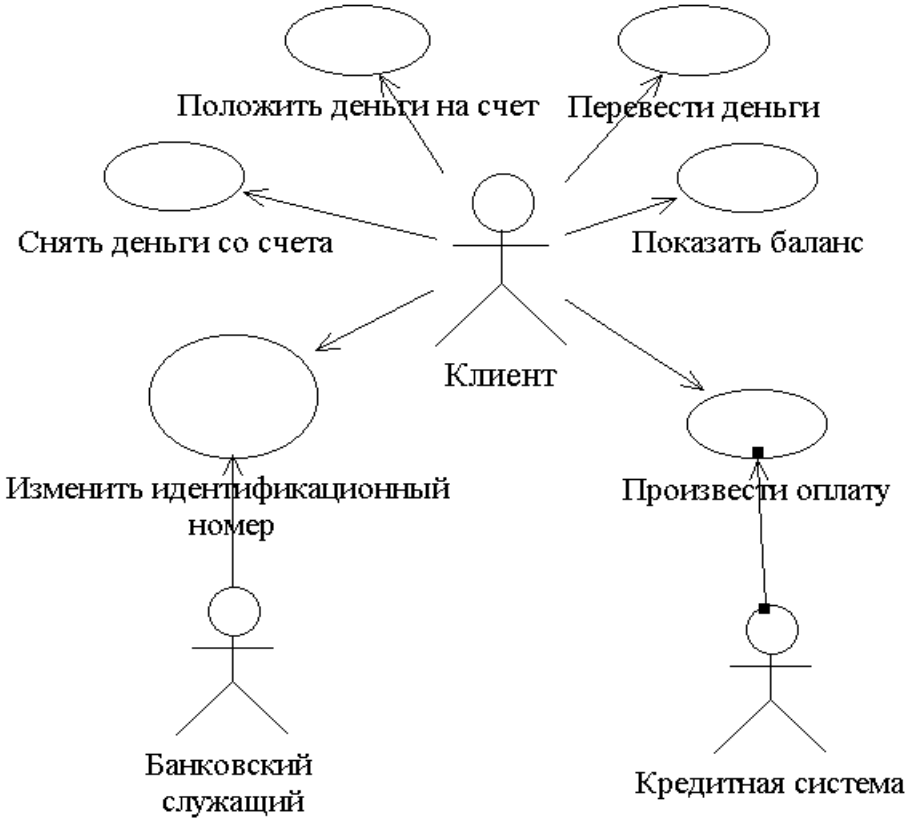


Рис. 2.33. Пример диаграммы Вариантов Использования

Из вариантов использования можно понять, какие функциональные возможности будут заложены в систему. Рассматривая действующих лиц можно выяснить, кто будет взаимодействовать с ней. Изучая все множество вариантов использования и действующих лиц, можно определить сферу применения системы, что она должна будет делать, а также что она не позволяет делать, и внести коррективы. Например, взглянув на приведенную в примере диаграмму, пользователь может сказать: «Я хочу дополнительно иметь возможность получать отчет о десяти последних транзакциях для моего счета».

Обычно для системы создается несколько диаграмм Вариантов Использования. На диаграмме высокого уровня (называемой «Главной» (Main)) указываются только группы (пакеты) вариантов использования. На других диаграммах эти группы конкретизируются, на них описываются варианты использования и действующие лица в рамках конкретной группы. Иногда может потребоваться нанести на одну диаграмму все варианты использования и всех действующих лиц системы. Количество и состав создаваемых диаграмм Вариантов Использования полностью зависит от разработчика. Важно только, чтобы они содержали достаточно информации, чтобы быть полезными, но не слишком много, чтобы не привести в замешательство.

Подробнее рассмотрим основные элементы диаграммы Вариантов Использования.

Действующее лицо (actor) - это то, что взаимодействует с создаваемой системой. Если варианты использования описывают все, что происходит внутри области действия системы, действующие лица определяют все, что находится вне этой области. На языке UML действующие лица представляются в виде фигур (рис. 2.33). Действующие лица делятся на три типа: пользователи системы, другие системы, взаимодействующие с данной, и время.

Первый тип действующих лиц - это ***физические личности***. Они наиболее типичны и имеются практически в каждой системе. В системе ATM, например, к этому типу относятся клиенты и обслуживающий персонал. Называя действующих лиц, используют их ролевые имена. Конкретный человек может играть множество ролей. Например, один человек утром может отвечать за поддержку системы ATM, т.е. относится к обслуживающему персоналу, а днем он может снять деньги со счета, чтобы пойти пообедать, при этом он уже является клиентом.

Второй тип действующих лиц – это ***другие системы***. Допустим, что у банка имеется кредитная система, используемая для работы с информацией о кредитных счетах клиентов. Если система ATM должна иметь возможность взаимодействовать с кредитной системой, в таком случае последняя становится действующим лицом. Нужно иметь в виду, что, делая систему действующим лицом, мы предполагаем, что она никогда не будет изменяться.

Действующие лица находятся вне сферы действия того, что мы разрабатываем, и, следовательно, не подлежат контролю с нашей стороны. Если мы собираемся изменять или разрабатывать и кредитную систему, то она попадает в наш проект и, таким образом, не может быть показана как действующее лицо.

Третий наиболее распространенный тип действующих лиц - ***время***. Время становится действующим лицом, если от него зависит запуск каких-либо событий в системе. Например, система АТМ может каждую полночь выполнять какие-либо служебные процедуры по настройке к согласованию своей работы. Так как время не подлежит нашему контролю, оно является действующим лицом.

Абстрактным называется действующее лицо, не имеющее экземпляров. Например, в системе может быть несколько действующих лиц: служащий с почасовой оплатой, служащий с окладом и служащий, нанятый на определенное время. Все они являются разновидностями одного действующего лица – служащего (рис. 2.34). Однако никто в компании не является просто служащим - каждый относится к одному из трех вышеназванных типов. Действующее лицо «Служащий» существует только для того, чтобы показать общность между этими тремя типами. У него нет экземпляров, так что это абстрактное действующее лицо.

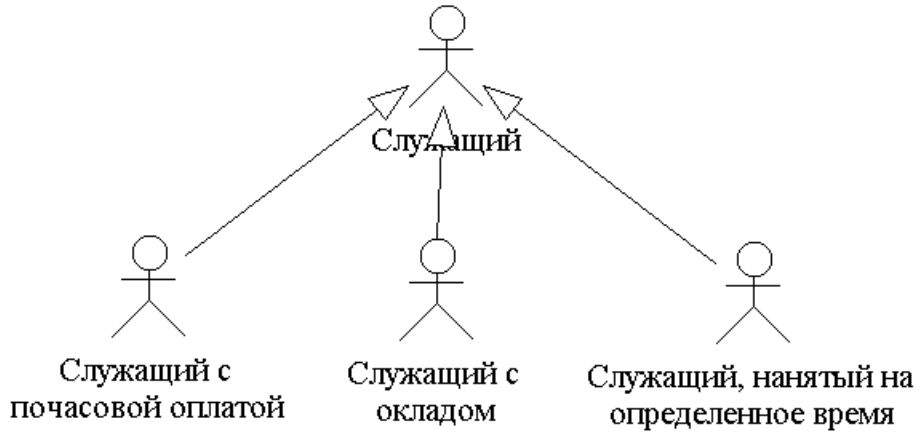
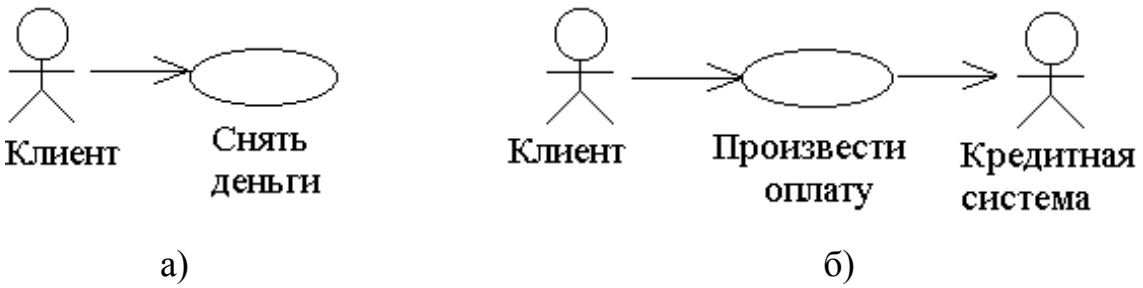


Рис. 2.34. Абстрактное действующее лицо

В языке UML для вариантов использования и действующих лиц поддерживается несколько типов ***связей***. Это связи: коммуникации (communication), использования (uses), расширения (extends) и обобщения действующего лица (actor generalization). Связи коммуникации описывают связи между действующими лицами и вариантами использования. Связи использования и расширения отражают связи между вариантами использова-

ния, а связи обобщения действующего лица - между действующими лицами.

Связь коммуникации (communicates relationship) - это связь между вариантом использования и действующим лицом. Связь коммуникации изображают в виде стрелки (рис. 2.35, а). Направление стрелки показывает, кто инициирует коммуникацию. В приведенном примере действующее лицо Клиент инициирует коммуникацию с системой для запуска функции «Снять деньги». Вариант использования также может инициировать коммуникацию с действующим лицом (рис. 2.35, б). В данном примере вариант использования инициирует коммуникацию с действующим лицом Кредитная система. Когда выполняется вариант использования «Произвести оплату», система АТМ инициирует коммуникацию с Кредитной системой, чтобы завершить транзакцию. Информация при этом движется в обоих направлениях: от АТМ к Кредитной системе и обратно; стрелка показывает, кто инициирует коммуникацию.



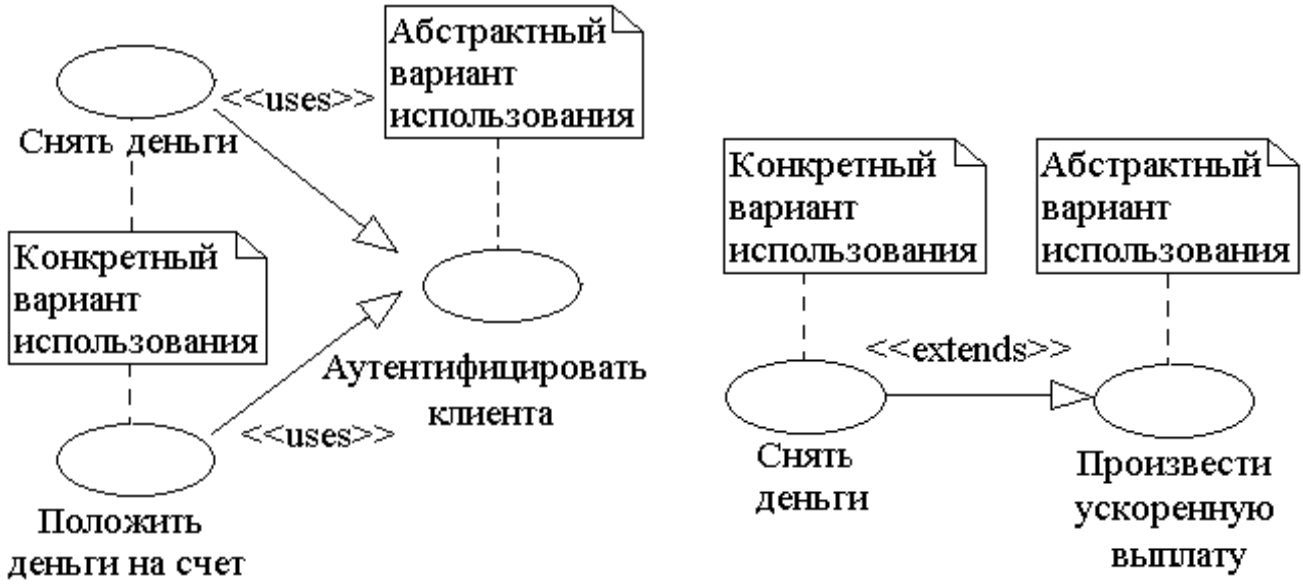
а) б)

Рис. 2.35. Графическая нотация для изображения:
а) однонаправленная коммуникация; б) вариант использования инициирует коммуникацию

Каждый вариант использования должен быть инициирован действующим лицом; исключения составляют лишь варианты использования в связях использования и расширения.

Связь использования (uses relationship) позволяет одному варианту использования задействовать функциональность другого. С помощью таких связей обычно моделируют многократно применяемую функциональность, встречающуюся в двух или более вариантах использования. В примере АТМ варианты использования «Снять деньги» и «Положить деньги на счет» должны опознать (аутентифицировать) клиента и его идентификационный номер перед тем, как разрешить выполнение самой транзакции. Вместо того чтобы подробно описывать процесс аутентификации для каждого из них, можно поместить эту функциональность в свой собственный вариант использования под названием «Аутентифицировать клиента». Когда какому-нибудь варианту использования потребуется выполнить эти действия, он сможет воспользоваться функциональностью созданного варианта использования «Аутентифи-

цировать клиента». Связь использования изображается с помощью стрелок и слова «uses» (использование), как показано на рис. 2.36, а.



а) б)

Рис. 2.36. Графическая нотация для изображения:
а) связи использования; б) связи расширения

Связь использования предполагает, что один вариант использования всегда применяет функциональные возможности другого. Например, независимо от того, как именно осуществляется вариант использования «Снять деньги», вариант использования «Аутентифицировать клиента» будет запущен в любом случае. ***Связи расширения*** (extends relationship) в отличие от использования позволяют варианту использования применять функциональные возможности, предоставляемые другим вариантом использования, только при необходимости. В связях расширения общая функциональность также выделяется в отдельный вариант использования.

Связи расширения изображают в виде стрелки со словом «extends» (расширение) (рис. 2.36, б). В этом примере вариант использования «Снять деньги» иногда применяет функциональные возможности, предоставляемые вариантом использования «Произвести ускоренную выплату». Это произойдет, например, когда клиент выберет пункт «Быстро снять $40». Предоставляющий дополнительные возможности вариант использования «Произвести ускоренную выплату» является абстрактным. Вариант использования «Снять деньги» - конкретный.

С помощью ***связи обобщения*** действующего лица (actor generalization

relationship) показывают, общие черты нескольких действующих лиц. Это отношение моделируется с помощью нотации, представленной на рис. 2.37, а. На диаграмме показано, что имеются два типа клиентов: индивидуальные и корпоративные. Это конкретные действующие лица. Они будут инстанцированы (наполнены) непосредственно. Действующее лицо Клиент - абстрактное. Оно никогда не инстанцируется непосредственно. Оно нужно лишь для того, чтобы показать наличие двух типов клиентов. При необходимости можно усиливать ветвление. Допустим, имеются два типа корпоративных клиентов: правительственные агентства и частные компании (рис. 2.37, б).

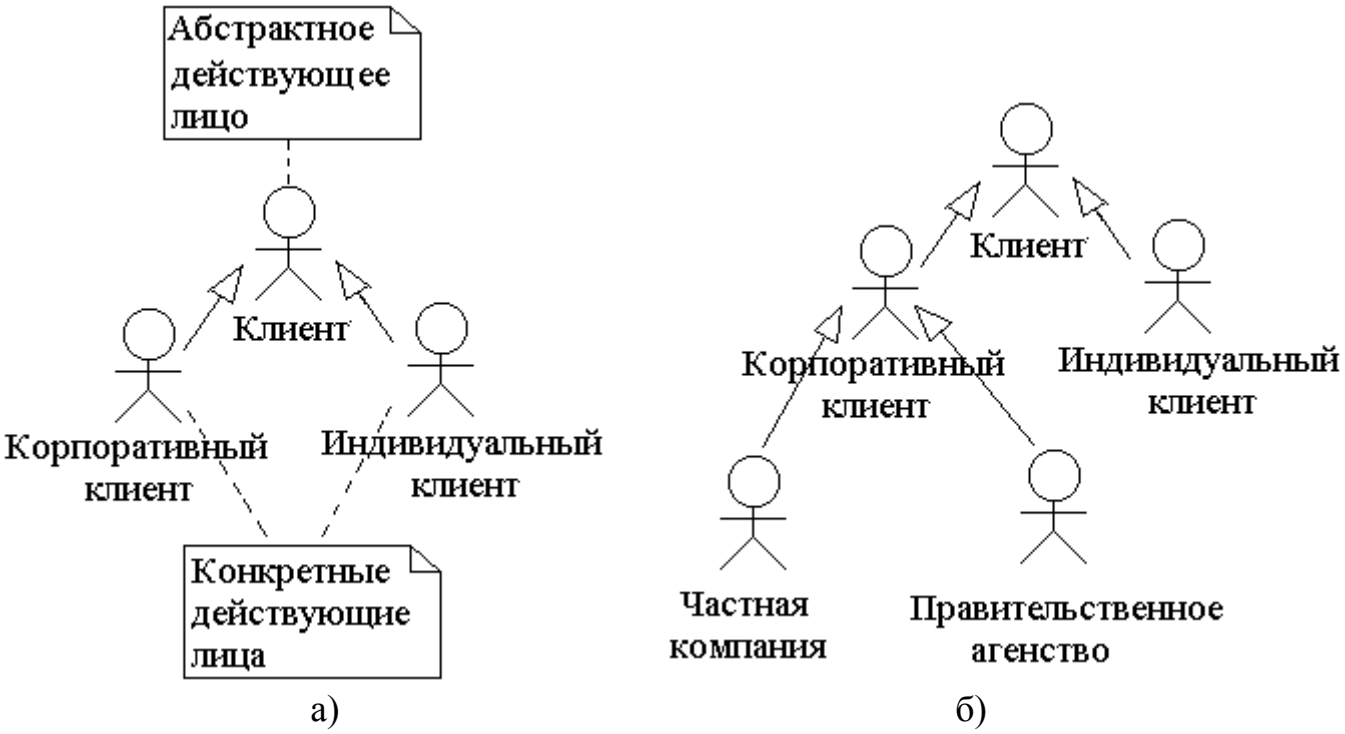


Рис. 2.37. Графическая нотация для изображения:
а) связи обобщения действующего лица; б) усиление ветвления связи обобщения

Связи обобщения создаются не всегда. В общем случае они нужны, если поведение действующего лица одного типа отличается от поведения другого настолько, что это влияет на систему. Если, например, корпоративные клиенты инициируют некоторые варианты использования, которые не могут быть запущены индивидуальными клиентами, то связи обобщения имеет смысл показать на диаграмме. Если клиенты обоих типов применяют одни и те же варианты использования, этого не требуется. Если действующие лица задействуют одни и те же варианты использования, но с некоторым отличием, их также не стоит включать в обобщение. Небольшие различия можно до-

кументировать в потоке событий варианта использования.

Описав основных элементов диаграмм Вариантов Использования, рассмотрим процесс их выявления.

В начале работы над проектом возникает естественный вопрос: «Как обнаружить варианты использования?». Можно обратится к документации заказчика, рассмотреть области использования системы на высоком уровне или документы концептуального характера. Нужно стараться учесть мнение каждого из заинтересованных лиц проекта и определить, что они ожидают от готового продукта. Каждому из них можно задать следующие вопросы.

1. Что он хочет делать с системой?
2. Будет ли он с ее помощью работать с информацией (вводить, получать, обновлять, удалять)?
3. Нужно ли будет информировать систему о каких-либо внешних событиях?
4. Должна ли система в свою очередь информировать пользователя о каких-либо изменениях или событиях?

Как уже упоминалось ранее, варианты использования - это не зависящее от реализации высокоуровневое представление того, что пользователь ожидает от системы. Рассмотрим каждый фрагмент этого определения.

Прежде всего, варианты использования не зависят от реализации. Нужно составлять варианты использования так, чтобы их можно было реализовать на любых языках (Java, C++, Visual Basic и т.п.). Варианты использования заостряют внимание на том, что должна делать система, а не на том, как она должна это делать.

Варианты использования - это высокоуровневое представление системы. Если, например, в вашей модели содержится 3000 вариантов использования, Вы теряете преимущество простоты. Создаваемый набор вариантов использования должен предоставить пользователям возможность увидеть всю систему целиком на самом высоком уровне. Поэтому вариантов использования не должно быть слишком много, чтобы клиенту не пришлось долго блуждать по страницам документации с целью выяснения того, что будет делать система. В то же время вариантов использования должно быть достаточно для полного описания поведения системы. Модель типичной системы содержит от 20 до 50 вариантов использования.

Наконец, варианты использования заостряют внимание на том, что пользователи хотят получить от системы. Каждый вариант использования должен представлять собой завершенную транзакцию между пользователем и системой. Названия вариантов использования должны быть деловыми, а не техническими терминами. Если в примере с АТМ назвать вариант использования «Интерфейс с банковской системой, осуществляющий перевод денег с

кредитной карточки и наоборот», это будет не приемлемо. Лучше дать название «Оплатить по карточке» - так будет понятнее для заказчика. Варианты использования обычно называют глаголами или глагольными фразами, описывая при этом, что пользователь видит как конечный результат процесса. Его не интересует, сколько других систем задействовано в варианте использования, какие конкретные шаги надо предпринять и сколько строчек кода требуется написать, чтобы заплатить по счету карточкой Visa. Для него важно только, чтобы оплата была произведена. Нужно заострять внимание на результате, который потребитель ожидает от системы, а не на действиях, которые нужно предпринять для достижения этого результата.

Как убедиться, что обнаружены все варианты использования? Для этого следует ответить на следующие вопросы.

1. Присутствует ли каждое функциональное требование хотя бы в одном варианте использования? Если требование не нашло отражение в варианте использования, оно не будет реализовано.
2. Учтено ли, как с системой будет работать каждое заинтересованное лицо?
3. Какую информацию будет передавать системе каждое заинтересованное лицо?
4. Какую информацию будет получать от системы каждое заинтересованное лицо?
5. Учтены ли проблемы, связанные с эксплуатацией? Кто-то должен будет запускать готовую систему и выключать ее.
6. Учтены ли все внешние системы, с которыми будет взаимодействовать данная?
7. Какой информацией каждая внешняя система будет обмениваться с данной?

Приведем пример создания диаграммы Вариантов Использования для системы обработки заказов. Система должна обеспечивать возможность добавления новых заказов, изменения старых, выполнения заказов, проверки и возобновления инвентарных описей. При получении заказа система должна послать сообщение бухгалтерской системе, которая выписывает счет. Если требуемого товара нет на складе, заказ должен быть отклонен.

Основными вариантами использования системы будут следующие: Ввести новый заказ, Изменить существующий заказ, Напечатать инвентарную опись, Обновить инвентарную опись, Оформить заказ, Отклонить заказ.

После выявления вариантов использования рассматривают возможные роли в системе и определяют действующих лиц. В данном примере будут следующие действующие лица: Продавец, Управляющий магазином, Клерк магазина, Бухгалтерская система.

На следующем шаге устанавливаются связи между выявленными элементами системы. За исключением одного случая, все связи однонаправленные ассоциации. Так как вариант использования «Отклонить заказ» при необходимости дополняет функциональные возможности варианта использования «Оформить заказ» между ними будет связь расширения.

Полученная диаграмма Вариантов Использования показана на рис. 2.38.

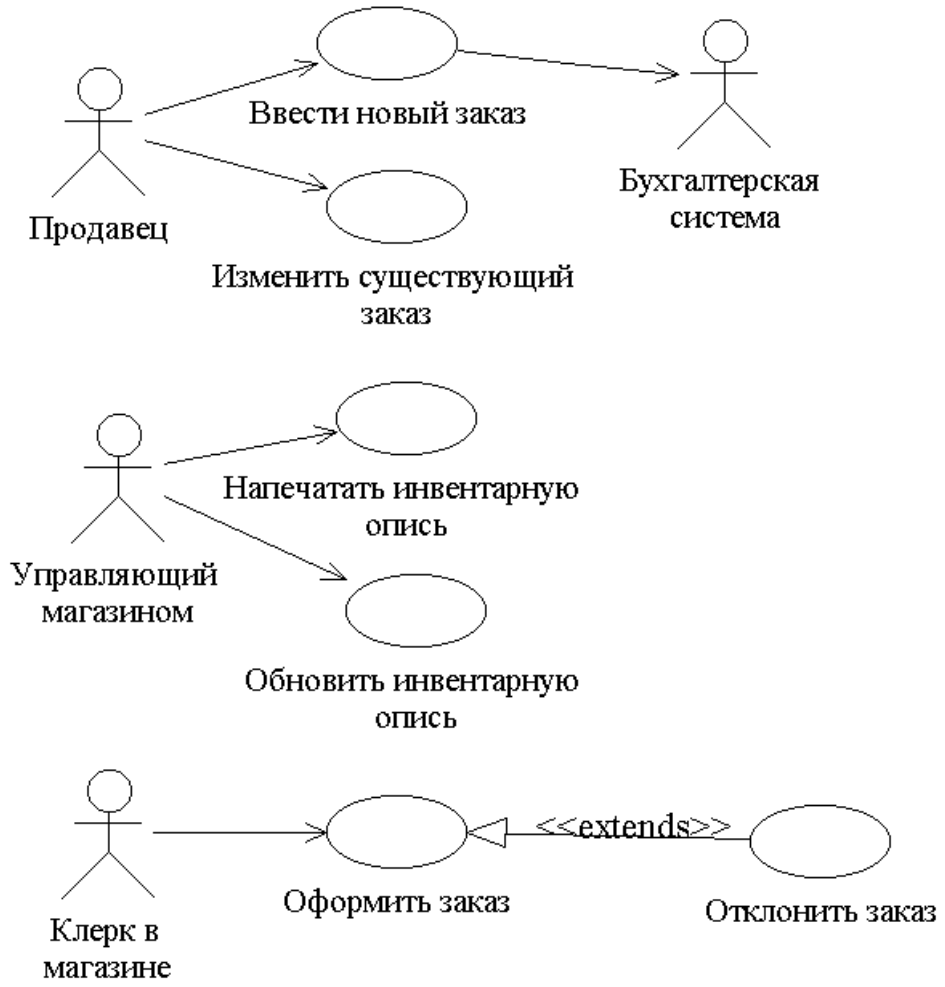


Рис. 2.38. Диаграмма вариантов использования системы обработки заказов

2.7.3. Потоки событий

Варианты использования начинают описывать, что должна будет делать система. Но чтобы фактически разработать систему, потребуются более конкретные детали. Они определяются в документе, называемом ***потоком со-***

бытий (flow of events). Целью потока событий является документирование процесса обработки данных, реализуемого в рамках варианта использования. Этот документ подробно описывает, что будут делать пользователи системы и сама система.

Поток событий не должен зависеть от реализации. Составляя этот документ, можно не учитывать на каком языке будет написана система (C++, PowerBuilder, Java или другом). Поток событий уделяет внимание тому, что (а не как) будет делать система. Обычно поток событий содержит:

1) краткое описание;
2) предусловия (pre-conditions);
3) основной поток событий;
4) альтернативный поток событий;
5) постусловия (post-conditions).

Рассмотрим эти составные части подробнее.

Каждый вариант использования должен иметь связанное с ним ***краткое описание*** того, что он будет делать. Например, вариант использования «Перевести деньги» системы АТМ 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».* Следует делать описание коротким и «к месту», при этом оно должно определять типы пользователей, выполняющих вариант использования, и ожидаемый ими конечный результат. Во время работы над проектом (особенно, если проект длинный) эти описания будут напоминать членам команды, почему тот или иной вариант использования был включен в проект и что он должен делать. Четко документируя цели каждого варианта использования, можно избежать противоречий среди разработчиков.

Предусловия варианта использования - это такие условия, которые должны быть выполнены, прежде чем вариант использования начнет свою работу. Например, таким условием может быть выполнение другого варианта использования или наличие у пользователя прав доступа, требуемых для запуска данного варианта использования. Не у всех вариантов использования бывают предварительные условия. Ранее упоминалось, что диаграммы Вариантов Использования не должны отражать порядок их выполнения. Однако с помощью предусловий можно документировать и такую информацию.

Конкретные детали вариантов использования отражаются в ***основном*** и ***альтернативном потоках событий***. Поток событий поэтапно описывает, что должно происходить во время выполнения заложенной в вариант использования функциональности. Поток событий уделяет внимание тому, что (а не как) будет делать система, причем описывает это с точки зрения пользователя. Первичный и альтернативный потоки событий содержат:

1) описание того, каким образом запускается вариант использования;
2) различные пути выполнения варианта использования;
3) нормальный, или основной, поток событий варианта использования;
4) отклонения от основного потока событий (так называемые альтернативные потоки);
5) потоки ошибок;
6) описание того, каким образом завершается вариант использования.

В качестве примера приведем потоки событий для варианта использования «Снять деньги» системы АТМ.

Основной поток.

1. Вариант использования начинается, когда клиент вставляет свою карточку в АТМ.

2. АТМ выдает приветствие и предлагает клиенту ввести свой персональный идентификационный номер.

3. Клиент вводит номер.

4. АТМ подтверждает введенный номер. Если номер не подтверждается, выполняется альтернативный поток событий А1.

5. АТМ выводит список доступных действий:

– «Положить деньги на счет»;

– «Снять деньги со счета»;

– «Перевести деньги».

6. Клиент выбирает пункт «Снять деньги».

7. АТМ запрашивает, сколько денег нужно снять.

8. Клиент вводит требуемую сумму.

9. АТМ определяет, достаточно ли на счету денег. Если денег недостаточно, выполняется альтернативный поток А2. Если во время подтверждения суммы возникают ошибки, выполняется поток ошибок Е1.

10. АТМ вычитает требуемую сумму из счета клиента.

11. АТМ выдает клиенту требуемую сумму наличными.

12. АТМ возвращает клиенту его карточку.

13. Вариант использования завершается.

Альтернативный поток А1: ввод неправильного идентификационного номера.

1. АТМ информирует клиента, что идентификационный номер введен неправильно.

2. АТМ возвращает клиенту его карточку.

3. Вариант использования завершается.

Альтернативный поток А2: недостаточно денег на счету.

1. АТМ информирует клиента, что денег на его счету недостаточно.

2. АТМ возвращает клиенту его карточку.

3. Вариант использования завершается.

Поток ошибок E1: ошибка в подтверждении запрашиваемой суммы.

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

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

3. АТМ возвращает клиенту его карточку.

4. Вариант использования завершается.

Документируя поток событий, можно использовать нумерованные списки (как это сделано в данном примере), ненумерованные списки, разбитый на параграфы текст, а также блок-схемы. Поток событий должен быть согласован с определенными ранее требованиями.

Описывая поток событий, нужно учитывать, что с этим документом будут работать все участники проекта. При его изучении заказчики будут проверять, соответствует ли он их ожиданиям, аналитики - соответствует ли он требованиям к системе. Менеджер проекта определяет из потока событий, что будет создано и делает оценку проекта.

Работая над потоком, нужно избегать детальных обсуждений того, как он будет реализован. Подобно кулинарному рецепту, в котором просто указывается: «Добавьте сахар», а не расписывается: «Откройте шкаф. Возьмите пакетик с сахаром. Возьмите стакан. Насыпьте сахар в стакан…». Аналогично, составляя поток событий, можно написать: «Проверить идентификационный номер пользователя», но не нужно указывать, что для этого необходимо обращаться к специальной таблице в базе данных. Нужно уделять внимание обмену информацией между пользователями и системой, но не подробному описанию ее реализации.

Постусловиями называются такие условия, которые должны быть выполнены после завершения варианта использования. Например, в конце варианта использования нужно установить флажок. Как и в случае предусловий, с помощью постусловий можно вводить сведения о порядке выполнения вариантов использования системы. Если после одного из вариантов использования должен всегда выполняться другой, это можно описать как постусловие. Такие условия имеются не у каждого варианта использования.