11_Унифицированный язык визуального моделирования Unified Modeling Language (UML)

Диаграммы в UML. Классы и стереотипы классов. Ассоциативные классы. Основные элементы диаграмм взаимодействия — объекты, сообщения. Диаграммы состояний: начального состояния, конечного состояния, переходы. Вложенность состояний. Диаграммы внедрения: подсистемы, компоненты, связи. Стереотипы компонент. Диаграммы размещения.

Существует множество технологий и инструментальных средств, с помощью которых можно реализовать в некотором смысле оптимальный проект ИС, начиная с этапа анализа и заканчивая созданием программного кода системы. В большинстве случаев эти технологии предъявляют весьма жесткие требования к процессу разработки и используемым ресурсам, а попытки трансформировать их под конкретные проекты оказываются безуспешными. Эти технологии представлены CASE-средствами верхнего уровня или CASE-средствами полного жизненного цикла (upper CASE tools или full life-cycle CASE tools). Они не позволяют оптимизировать деятельность на уровне отдельных элементов проекта, и, как следствие, многие разработчики перешли на так называемые CASE-средства нижнего уровня (lower CASE tools). Однако они столкнулись с новой проблемой — проблемой организации взаимодействия между различными командами, реализующими проект.

Унифицированный язык объектно-ориентированного моделирования Unified Modeling Language (UML) явился средством достижения компромисса между этими подходами. Существует достаточное количество инструментальных средств, поддерживающих с помощью *UML* жизненный цикл информационных систем, и, одновременно, *UML* является достаточно гибким для настройки и поддержки специфики деятельности различных команд разработчиков.

Мощный толчок к разработке этого направления информационных технологий дало распространение *объектно-ориентированных* языков программирования в конце 1980-х — начале 1990-х годов. Пользователям хотелось получить единый язык моделирования, который объединил бы в себе всю мощь *объектно-ориентированного* подхода и давал бы четкую модель системы, отражающую все ее значимые стороны. К середине девяностых явными лидерами в этой области стали методы Booch (Grady Booch), OMT-2 (Jim Rumbaugh), OOSE — Object-Oriented Software Engineering (Ivar Jacobson). Однако эти три метода имели свои сильные и слабые стороны: OOSE был лучшим на стадии анализа проблемной области и анализа требований к системе, OMT-2 был наиболее предпочтителен на стадиях анализа и разработки информационных систем, Booch лучше всего подходил для стадий дизайна и разработки.

Все шло к созданию единого языка, который объединял бы сильные стороны известных методов и обеспечивал наилучшую поддержку моделирования. Таким языком оказался *UML*. Создание *UML* началось в октябре 1994 г., когда Джим Рамбо и Гради Буч из Rational Software Corporation стали работать над объединением своих методов OMT и Booch. Осенью 1995 г. увидела свет первая черновая версия объединенной методологии, которую они назвали Unified Method 0.8. После присоединения в конце 1995 г. к Rational Software Corporation Айвара Якобсона и его фирмы Objectory, усилия трех создателей наиболее распространенных *объектно-ориентированных* методологий были объединены и направлены на создание *UML*.

В настоящее время консорциум пользователей *UML* Partners включает в себя представителей таких грандов информационных технологий, как Rational Software, Microsoft, IBM, Hewlett-Packard, Oracle, DEC, Unisys, IntelliCorp, Platinum Technology.

UML представляет собой *объектно-ориентированный* язык моделирования, обладающий следующими основными характеристиками:

- является языком визуального моделирования, который обеспечивает разработку репрезентативных моделей для организации взаимодействия заказчика и разработчика ИС, различных групп разработчиков ИС;

- содержит механизмы расширения и специализации базовых концепций языка.

UML — это стандартная нотация визуального моделирования программных систем, принятая консорциумом Object Managing Group (OMG) осенью 1997 г., и на сегодняшний день она поддерживается многими *объектно-ориентированными* CASE-продуктами.
UML включает внутренний набор средств моделирования (модулей?) ("ядро"), которые сейчас приняты во многих методах и средствах моделирования. Эти концепции необходимы в большинстве прикладных задач, хотя не каждая концепция необходима в каждой части каждого приложения. Пользователям языка предоставлены возможности:

- строить модели на основе средств ядра, без использования механизмов расширения для большинства типовых приложений;
- добавлять при необходимости новые элементы и условные обозначения, если они не входят в ядро, или специализировать компоненты, систему условных обозначений (нотацию) и ограничения для конкретных предметных областей.

Синтаксис и семантика основных объектов UML
Классы
Классы — это базовые элементы любой *объектно-ориентированной* системы. *Классы* представляют собой описание совокупностей однородных объектов с присущими им свойствами — атрибутами, операциями, отношениями и семантикой.
В рамках модели каждому *классу* присваивается уникальное имя, отличающее его от других *классов*. Если используется составное имя (в начале имени добавляется имя пакета, куда входит *класс*), то имя *класса* должно быть уникальным в пакете.
Атрибут — это свойство *класса*, которое может принимать множество значений. Множество допустимых значений атрибута образует домен. Атрибут имеет имя и отражает некоторое свойство моделируемой сущности, общее для всех объектов данного *класса*. *Класс* может иметь произвольное количество атрибутов.
Операция — реализация функции, которую можно запросить у любого объекта *класса*. Операция показывает, что можно сделать с объектом. Исполнение операции часто связано с обработкой и изменением значений атрибутов объекта, а также изменением состояния объекта.
На рис. 11.1 приведено графическое изображение *класса* "Заказ" в нотации *UML*.

Рис. 11.1. Изображение класса в UML
Синтаксис *UML* для свойств *классов* (в отдельных программных средствах, например, в IBM *UML* Modeler, порядок записи параметров может быть иным):
<
признак видимости> <имя атрибута> : <тип данных>
= <значение по умолчанию>
<признак видимости> <имя операции> <(список аргументов)>
Видимость свойства указывает на возможность его использования другими *классами*. Один *класс* может "видеть" другой, если тот находится в области действия первого и между ними существует явное или неявное отношение. В языке *UML* определены три уровня видимости:

- public (общий) — любой внешний *класс*, который "видит" данный, может пользоваться его общими свойствами. Обозначаются знаком "+" перед именем атрибута или операции;

- protected (защищенный) — только любой потомок данного *класса* может пользоваться его защищенными свойствами. Обозначаются знаком "#";
- private (закрытый) — только данный *класс* может пользоваться этими свойствами. Обозначаются символом "-" .

Еще одной важной характеристикой атрибутов и операций *классов* является область действия. Область действия свойства указывает, будет ли оно проявлять себя по-разному в каждом экземпляре *класса*, или одно и то же значение свойства будет совместно использоваться всеми экземплярами:

- instance (экземпляр) — у каждого экземпляра *класса* есть собственное значение данного свойства;
- classifier (классификатор) — все экземпляры совместно используют общее значение данного свойства (выделяется на диаграммах подчеркиванием).

Возможное количество экземпляров *класса* называется его кратностью. В *UML* можно определять следующие разновидности *классов*:

- не содержащие ни одного экземпляра — тогда *класс* становится служебным (Abstract);
- содержащие ровно один экземпляр (Singleton);
- содержащие заданное число экземпляров;
- содержащие произвольное число экземпляров.

Принципиальное назначение *классов* характеризуют стереотипы. Это, фактически, классификация объектов на высоком уровне, позволяющая определить некоторые основные свойства объекта (пример стереотипа — *класс* "действующее лицо"). Механизм стереотипов является также средством расширения словаря *UML* за счет создания на основе существующих блоков языка новых, специфичных для решения конкретной проблемы.

Диаграммы классов

Классы в *UML* изображаются на ***диаграммах классов***, которые позволяют описать систему в статическом состоянии — определить типы объектов системы и различного рода статические связи между ними.

Классы отображают типы объектов системы.

Между *классами* возможны различные отношения, представленные на рис. 11.2:

- зависимости, которые описывают существующие между *классами* отношения использования;
- обобщения, связывающие обобщенные *классы* со специализированными;
- ассоциации, отражающие структурные отношения между объектами *классов*.

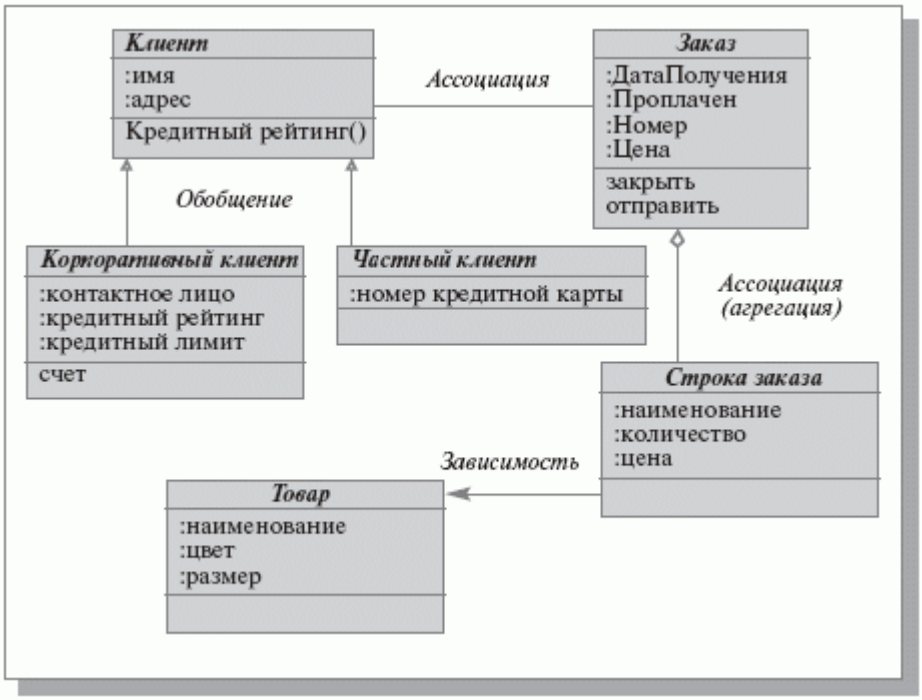

Рис. 11.2. Отображение связей между классами

Зависимостью называется отношение использования, согласно которому изменение в спецификации одного элемента (например, *класса* "товар") может повлиять на использующий его элемент (*класс* "строка заказа"). Часто зависимости показывают, что один *класс* использует другой в качестве аргумента.

Обобщение — это отношение между общей сущностью (родителем — *класс* "клиент") и ее конкретным воплощением (потомком — *классы* "корпоративный клиент" или "частный клиент"). Объекты *класса*-потомка могут использоваться всюду, где встречаются объекты *класса*-родителя, но не наоборот. При этом он наследует свойства родителя (его атрибуты и операции). Операция потомка с той же сигнатурой, что и у(?) родителя, замещает(?) операцию родителя; это свойство называют полиморфизмом. *Класс*, у которого нет родителей, но есть потомки, называется корневым. *Класс*, у которого нет потомков, называется листовым.

Ассоциация — это отношение, показывающее, что объекты одного типа неким образом связаны с объектами другого типа ("клиент" может сделать "заказ"). Если между двумя *классами* определена ассоциация, то можно перемещаться от объектов одного *класса* к объектам другого. При необходимости направление навигации может задаваться стрелкой. Допускается задание ассоциаций на одном *классе*. В этом случае оба конца ассоциации относятся к одному и тому же *классу*. Это означает, что с объектом некоторого *класса* можно связать другие объекты из того же *класса*. Ассоциации может быть присвоено имя, описывающее семантику отношений. Каждая ассоциация имеет две роли, которые могут быть отражены на диаграмме (рис. 11.3). Роль ассоциации обладает свойством множественности, которое показывает, сколько соответствующих объектов может участвовать в данной связи.

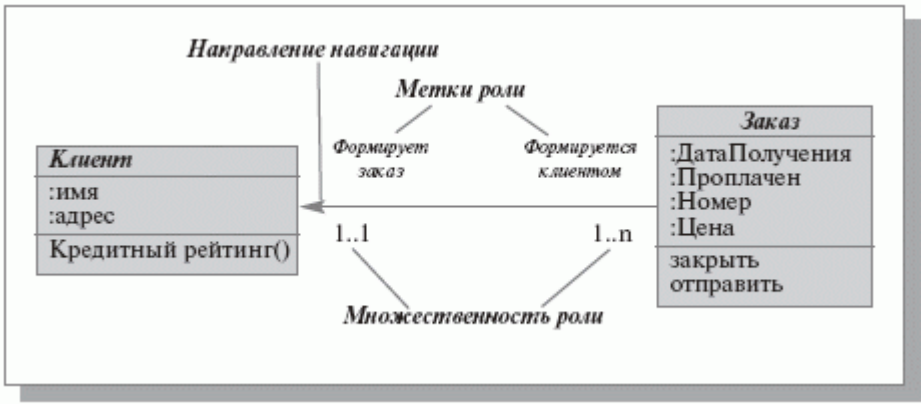

Рис. 11.3. Свойства ассоциации

рис. 11.3 иллюстрирует модель формирования заказа. Каждый заказ может быть создан единственным клиентом (множественность роли 1.1). Каждый клиент может создать один и более заказов (множественность роли 1..n). Направление навигации показывает, что каждый заказ должен быть "привязан" к определенному клиенту.

Такого рода ассоциация является простой и отражает отношение между равноправными сущностями, когда оба *класса* находятся на одном концептуальном уровне и ни один не является более важным, чем другой. Если приходится моделировать отношение типа "часть-целое", то используется специальный тип ассоциации — агрегирование. В такой ассоциации один из *классов* имеет более высокий ранг (целое — *класс* "заказ", рис. 11.2) и состоит из нескольких меньших по рангу *классов* (частей — *класс* "строка заказа"). В *UML* используется и более сильная разновидность агрегации — композиция, в которой объект-часть может принадлежать только единственному целому. В композиции жизненный цикл частей и целого совпадают, любое удаление целого обязательно захватывает и его части.

Для ассоциаций можно задавать атрибуты и операции, создавая по обычным правилам *UML* *классы* ассоциаций.

Диаграммы использования

Диаграммы использования описывают функциональность ИС, которая будет видна пользователям системы. "Каждая функциональность" изображается в виде "прецедентов использования" (use case) или просто прецедентов. Прецедент — это типичное взаимодействие пользователя с системой, которое при этом:

- описывает видимую пользователем функцию,
- может представлять различные уровни детализации,
- обеспечивает достижение конкретной цели, важной для пользователя.

Прецедент обозначается на диаграмме овалом, связанным с пользователями, которых принято называть действующими лицами (актеры, actors). Действующие лица используют систему (или используются системой) в данном прецеденте. Действующее лицо выполняет некоторую роль в данном прецеденте. На диаграмме изображается только одно действующее лицо, однако реальных пользователей, выступающих в данной роли по отношению к ИС, может быть много. Список всех прецедентов фактически определяет функциональные требования к ИС, которые лежат в основе разработки технического задания на создание системы.

На ***диаграммах прецедентов***, кроме связей между действующими лицами и прецедентами, возможно использование еще двух видов связей между прецедентами: "использование" и "расширение" (рис. 11.4). Связь типа "расширение" применяется, когда один прецедент подобен другому, но несет несколько большую функциональную нагрузку. Ее следует применять при описании изменений в нормальном поведении системы. Связь типа "использование" позволяет выделить некий фрагмент поведения системы и включать его в различные прецеденты без повторного описания.

На рис. 11.4 показано, что при исполнении прецедента "формирование заказа" возможно использование информации из предыдущего заказа, что позволит не вводить все

необходимые данные. А при исполнении прецедентов "оценить риск сделки" и "согласовать цену" необходимо выполнить одно и то же действие — рассчитать стоимость заказа.

Рис. 11.4. Связи на диаграммах прецедентов

Динамические аспекты поведения системы отражаются приведенными ниже диаграммами. В отличие от некоторых подходов объектного моделирования, когда и состояние, и поведение системы отображаются на *диаграммах классов*, *UML* отделяет описание поведения в ***диаграммы взаимодействия***. В *UML диаграммы классов* не содержат сообщений, которые усложняют их чтение. Поток сообщений между объектами выносится на *диаграммы взаимодействия*. Как правило, *диаграмма взаимодействия* охватывает поведение объектов в рамках одного варианта использования.

Прямоугольники на диаграмме представляют различные объекты и роли, которые они имеют в системе, а линии между *классами* отображают отношения (или ассоциации) между ними. Сообщения обозначаются ярлыками возле стрелок, они могут иметь нумерацию и показывать возвращаемые значения.

Существуют два вида *диаграмм взаимодействия*: диаграммы последовательностей и кооперативные диаграммы.

Диаграммы последовательностей

Этот вид диаграмм используется для точного определения логики сценария выполнения прецедента. Диаграммы последовательностей отображают типы объектов, взаимодействующих при исполнении прецедентов, сообщения, которые они посылают друг другу, и любые возвращаемые значения, ассоциированные с этими сообщениями. Прямоугольники на вертикальных линиях показывают "время жизни" объекта. Линии со стрелками и надписями названий методов означают вызов метода у объекта.

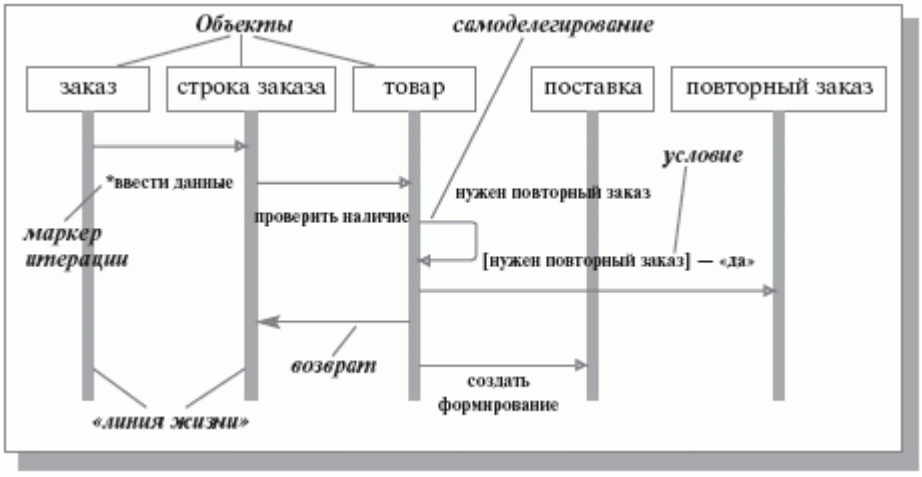

Рис. 11.5. Диаграмма последовательности обработки заказа

- вводятся строки заказа;
- по каждой строке проверяется наличие товара;

- если запас достаточен — инициируется поставка;
- если запас недостаточен — инициируется дозаказ (повторный заказ).

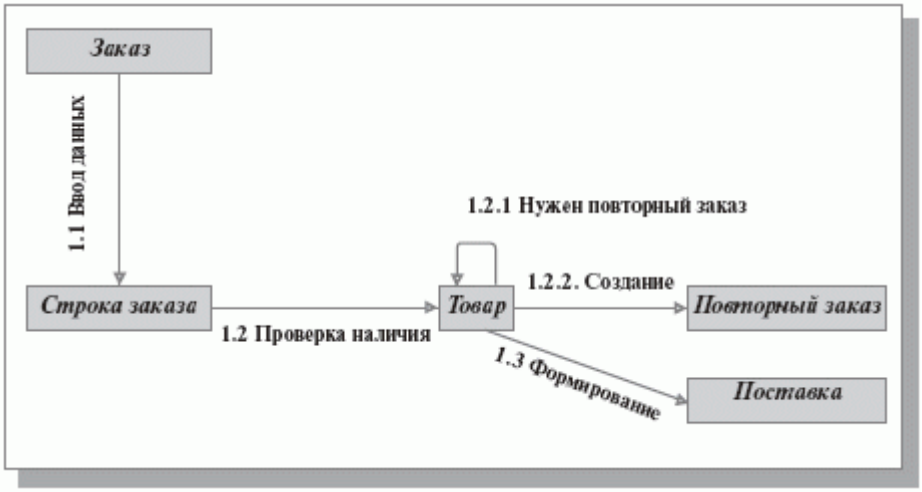

Рис. 11.6. Кооперативная диаграмма прохождения заказа

Сообщения появляются в той последовательности, как они показаны на диаграмме — сверху вниз. Если предусматривается отправка сообщения объектом самому себе (самоделегирование), то стрелка начинается и заканчивается на одной "линии жизни".

На диаграммы может быть добавлена управляющая информация: описание условий, при которых посылается сообщение; признак многократной отправки сообщения (маркер итерации); признак возврата сообщения.

Кооперативные диаграммы

На кооперативных диаграммах объекты (или *классы*) показываются в виде прямоугольников, а стрелками обозначаются сообщения, которыми они обмениваются в рамках одного варианта использования. Временная последовательность сообщений отражается их нумерацией.

Диаграммы состояний

Диаграммы состояний используются для описания поведения сложных систем. Они определяют все возможные состояния, в которых может находиться объект, а также процесс смены состояний объекта в результате некоторых событий.Эти диаграммы обычно используются для описания поведения одного объекта в нескольких прецедентах.

Прямоугольниками представляются состояния, через которые проходит объект во время своего поведения. Состояниям соответствуют определенные значения атрибутов объектов.

Стрелки представляют переходы от одного состояния к другому, которые вызываются выполнением некоторых функций объекта. Имеется также два вида псевдо-состояний: начальное состояние, в котором находится только что созданный объект, и конечное состояние, которое объект не покидает, как только туда перешел.

Переходы имеют метки, которые синтаксически состоят из трех необязательных частей (см. рис. 11.7):

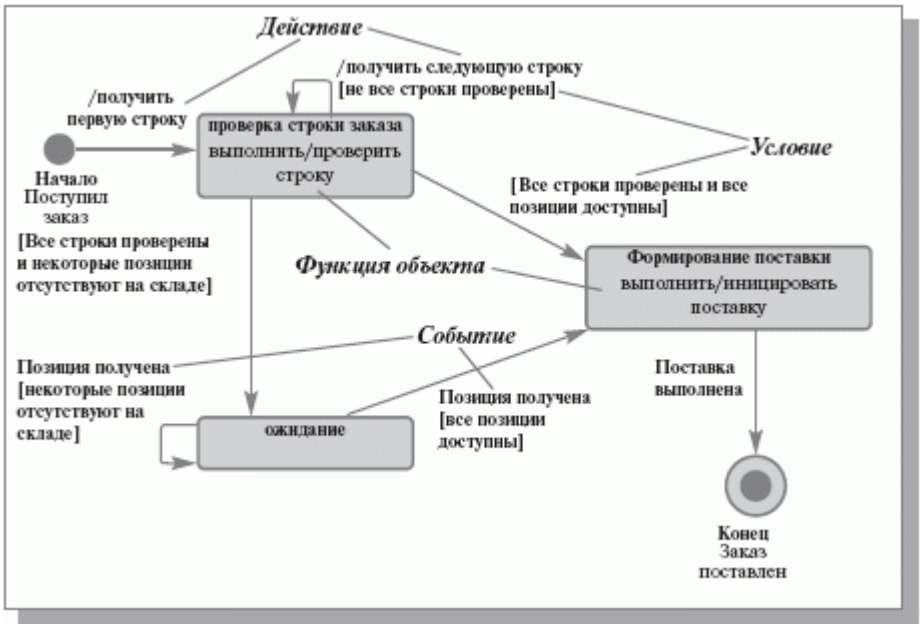

Рис. 11.7. Диаграмма состояний объекта «заказ»

<

Событие> <[Условие]> < / Действие>.

На диаграммах также отображаются функции, которые выполняются объектом в определенном состоянии. Синтаксис метки деятельности:

выполнить/< деятельность >.

Диаграммы деятельности

Диаграмма деятельности — это частный случай *диаграммы состояний*. На диаграмме деятельности представлены переходы потока управления от одной деятельности к другой внутри системы. Этот вид диаграмм обычно используется для описания поведения, включающего в себя множество параллельных процессов.

Основными элементами диаграмм деятельности являются (рис. 11.8):

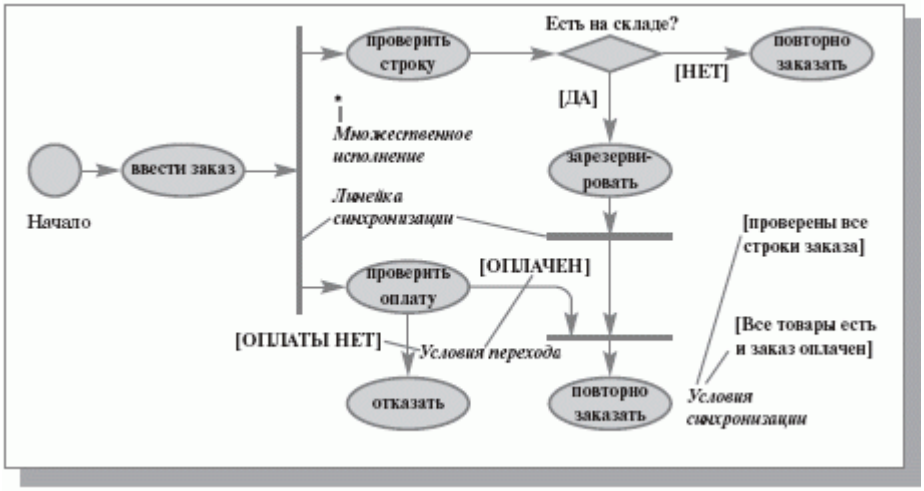

Рис. 11.8. Диаграмма деятельности — обработка заказа

- овалы, изображающие действия объекта;
- линейки синхронизации, указывающие на необходимость завершить или начать несколько действий (модель логического условия "И");
- ромбы, отражающие принятие решений по выбору одного из маршрутов выполнения процесса (модель логического условия "ИЛИ");
- стрелки — отражают последовательность действий, могут иметь метки условий.

На диаграмме деятельности могут быть представлены действия, соответствующие нескольким вариантам использования. На таких диаграммах появляется множество начальных точек, поскольку они отражают теперь реакцию системы на множество внешних

событий. Таким образом, диаграммы деятельности позволяют получить полную картину поведения системы и легко оценивать влияние изменений в отдельных вариантах использования на конечное поведение системы.
Любая деятельность может быть подвергнута дальнейшей декомпозиции и представлена в виде отдельной диаграммы деятельности или спецификации (словесного описания).

Диаграммы компонентов

Диаграммы компонентов позволяют изобразить модель системы на физическом уровне. Элементами диаграммы являются компоненты — физические замещаемые модули системы. Каждый компонент является полностью независимым элементом системы. Разновидностью компонентов являются узлы. Узел — это элемент реальной (физической) системы, который существует во время функционирования программного комплекса и представляет собой вычислительный ресурс, обычно обладающий как минимум некоторым объемом памяти, а часто еще и способностью обработки. Узлы делятся на два типа:

- устройства — узлы системы, в которых данные не обрабатываются.
- процессоры — узлы системы, осуществляющие обработку данных.

Для различных типов компонентов предусмотрены соответствующие стереотипы в языке *UML*.
Компонентом может быть любой достаточно крупный модульный объект, такой как таблица или экстент базы данных, подсистема, бинарный исполняемый файл, готовая к использованию система или приложение. Таким образом, *диаграмму компонентов* можно рассматривать как *диаграмму классов* в более крупном (менее детальном) масштабе. Компонент, как правило, представляет собой физическую упаковку логических элементов, таких как *классы*, интерфейсы и кооперации.
Основное назначение *диаграмм компонентов* — разделение системы на элементы, которые имеют стабильный интерфейс и образуют единое целое. Это позволяет создать ядро системы, которое не будет меняться в ответ на изменения, происходящие на уровне подсистем.
На рис. 11.9 показана упрощенная схема элементов фрагмента корпоративной системы. "Коробки" представляют собой компоненты — приложения или внутренние подсистемы. Пунктирные линии отражают зависимости между компонентами.

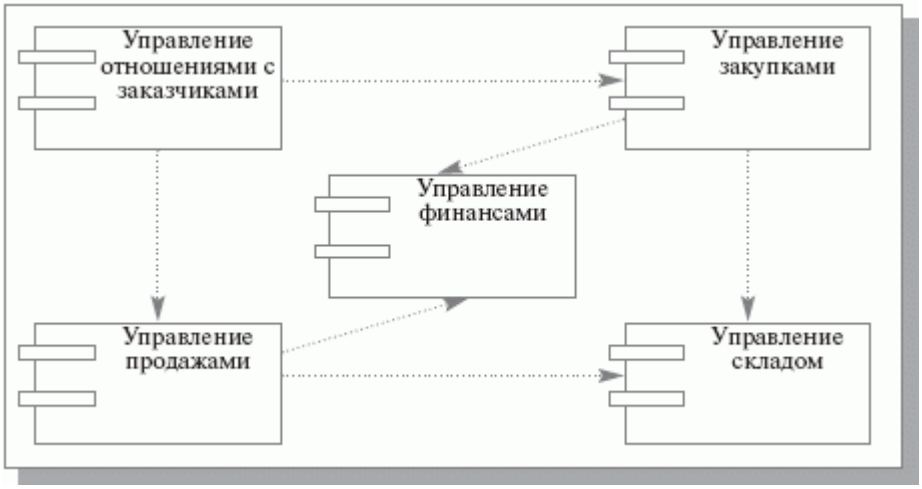

Рис. 11.9. Диаграмма компонентов фрагмента КИС

Каждый компонент диаграммы при необходимости документируется с помощью более детальной *диаграммы компонентов*, диаграммы сценариев или *диаграммы классов*.

Пакеты UML

Пакеты представляют собой универсальный механизм организации элементов в группы. В пакет можно поместить диаграммы различного типа и назначения. В отличие от компонентов, существующих во время работы программы, пакеты носят чисто концептуальный характер, то есть существуют только во время разработки. Изображается

пакет в виде папки с закладкой, содержащей, как правило, только имя и иногда — описание содержимого.
Диаграмма пакетов содержит пакеты *классов* и зависимости между ними. Зависимость между двумя пакетами имеет место в том случае, если изменения в определении одного элемента влекут за собой изменения в другом. По отношению к пакетам можно использовать механизм обобщения (см. выше раздел "*Диаграммы классов*").