UML Основы

Краткое руководство
по стандартному языку
объектного моделирования

Третье издание

Мартин Фаулер

Санкт-Петербург
2005

11

Диаграммы деятельности

Диаграммы деятельности – это технология, позволяющая описывать логику процедур, бизнес-процессы и потоки работ. Во многих случаях они напоминают блок-схемы, но принципиальная разница между диаграммами деятельности и нотацией блок-схем заключается в том, что первые поддерживают параллельное процессы.

Диаграммы деятельности подвергались самым большим изменениям при смене версий языка UML, поэтому неудивительно, что они были снова изменены и существенно расширены в UML 2. В UML 1 диаграммы деятельности рассматривались как особый случай диаграмм состояний. Это вызвало немало трудностей у специалистов, моделирующих потоки работ, для которых хорошо подходят диаграммы деятельности. В UML 2 это ограничение было ликвидировано.

На рис. 11.1 показан пример простой диаграммы деятельности. Мы стартуем с начального узла (initial node), а затем выполняем операцию `Receive Order` (Принять заказ). Затем идет ветвление (fork), которое имеет один входной поток и несколько выходных параллельных потоков.

Из рис. 11.1 видно, что операции `Fill Order` (Заполнить заявку), `Send Invoice` (Послать счет) и следующие за ними выполняются параллельно. По существу, в данном случае это означает, что последовательность операций не имеет значения. Я могу заполнить заявку, послать счет, доставить товар (`Delivery`), а затем получить оплату (`Receive Payment`); или я могу послать счет, получить оплату, заполнить заявку, а затем доставить товар. Посмотрите на рисунок.

Я могу также выполнять операции поочередно. Я беру со склада первую позицию заказа, печатаю счет, беру вторую позицию заказа, кладу счет в конверт и т. д. Или я могу что-то делать одновременно: печатать счет одной рукой, а другой рукой брать что-нибудь со склада. Согласно диаграмме, любая из этих последовательностей действий допустима.

Диаграмма деятельности позволяет любому, кто выполняет данный процесс, выбирать порядок действий. Другими словами, диаграмма

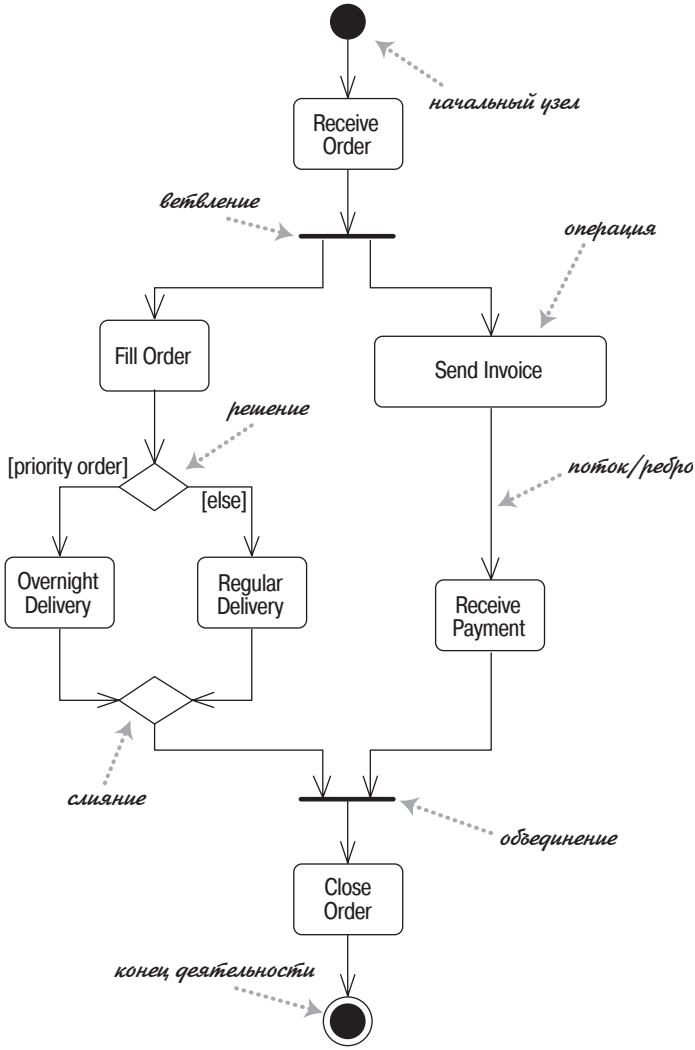

Рис. 11.1. Простая диаграмма деятельности

только устанавливает правила обязательной последовательности действий, которым я должен следовать. Это важно для моделирования бизнес-процессов, поскольку эти процессы часто выполняются параллельно. Такие **диаграммы также полезны при разработке параллельных алгоритмов, в которых независимые потоки могут выполнять работу параллельно.**

При наличии параллелизма необходима синхронизация. Мы не закрываем заказ, пока он не оплачен и не доставлен. **Это показывается с помощью объединения (join)** перед операцией `Close Order` (Закрыть заказ). Исходящий из объединения поток выполняется только в том слу-

чае, когда все входящие потоки достигли объединения. Поэтому вы можете закрыть заказ, только когда принята оплата и заказ доставлен.

В UML 1 действовали определенные правила для балансировки ветвлений и объединений, так как диаграммы деятельности представляли особый случай диаграмм состояний. В UML 2 в такой балансировке уже нет необходимости.

Заметьте, что **узлы на диаграмме деятельности называются операциями (actions), а не активностями (activities). Строго говоря, деятельность относится к последовательности действий, поэтому диаграмма представляет деятельность, состоящую из операций.**

Условное поведение схематически обозначается с помощью решений (decisions) и слияний (merges). Решение, которое в UML 1 называлось ветвью, имеет один входящий поток и несколько защищенных выходных потоков. **Каждый выходной поток имеет защиту – условное выражение, помещенное в квадратные скобки. Каждый раз при достижении решения выбирается только один из выходных потоков, поэтому защиты должны быть взаимно исключающими.** Применение `[else]` в качестве защиты означает, что поток `[else]` используется в том случае, когда другие защиты данного решения принимают ложное значение.

На рис. 11.1 решение располагается после операции заполнения заявки. Если у вас срочный заказ, то он выполняется в течение суток (`Overnight Delivery`); в противном случае производится обычная доставка (`Regular Delivery`).

Слияние (merge) имеет несколько входных потоков и один выходной. Слияние обозначает завершение условного поведения, которое было начато решением.

На моей диаграмме каждая операция имеет один входящий в нее поток и один выходящий. В UML 1 подразумевалось, что несколько входящих потоков имеют слияние. Другими словами, операция выполнялась, если запускался любой поток. В UML 2 это было изменено, так что вместо слияния предполагается объединение; таким образом, операция выполняется, только если все потоки пройдены. Поэтому я рекомендую применять операции с единственным входным потоком и единственным выходным, а также явно показывать все объединения и слияния; это избавит вас от путаницы.

Декомпозиция операции

Операции могут быть разбиты на вложенные деятельности (subactivities). Я могу взять алгоритм доставки, показанный на рис. 11.1, и определить его как самостоятельную деятельность (рис. 11.2), а затем вызвать его как операцию (рис. 11.3).

Операции могут быть реализованы или как вложенные деятельности или как методы классов. Вложенную деятельность можно обозначить

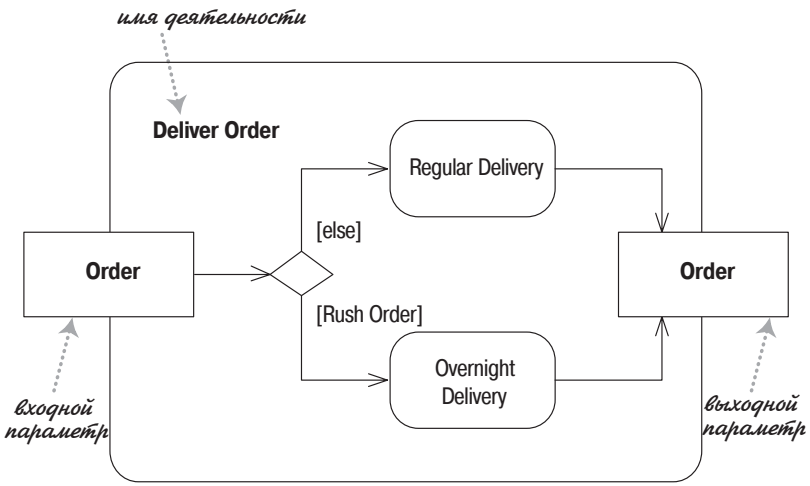

Рис. 11.2. Дополнительная диаграмма деятельности

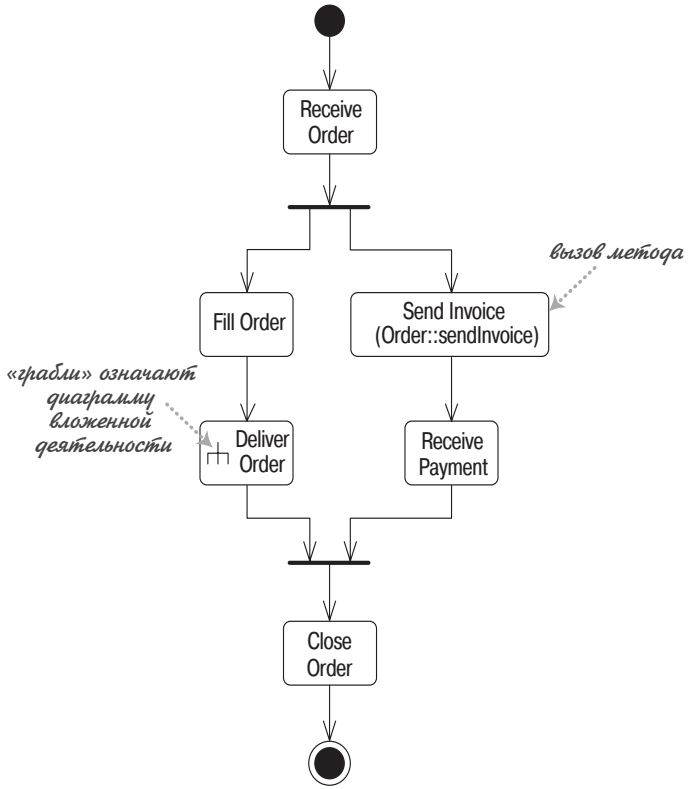

Рис. 11.3. Деятельность из рис. 11.1 модифицирована для вызова деятельности из рис. 11.2

с помощью символа «граблей». Вызов метода отображается с помощью синтаксиса `имя-класса::имя-метода`**.** Можно также вставить в символ операции фрагмент кода, если поведение представлено не единственным вызовом метода.

Разделы

Диаграммы деятельности рассказывают о том, что происходит, но ничего не говорят о том, кто какие действия выполняет. В программировании это означает, что диаграмма не отражает, какой класс является ответственным за ту или иную операцию. В моделировании бизнес-процессов это означает, что не отражено распределение обязанностей между подразделениями фирмы. Это не всегда представляет собой трудность; часто имеет смысл сконцентрироваться на том, что происходит, а не на том, кто какую роль играет в данном сценарии.

Можно разбить диаграмму деятельности на разделы (partitions), чтобы показать, кто что делает, то есть какие операции выполняет тот или иной класс или подразделение предприятия. На рис. 11.4 приведен простой пример, показывающий, как операции по обработке заказа могут быть распределены между различными подразделениями.

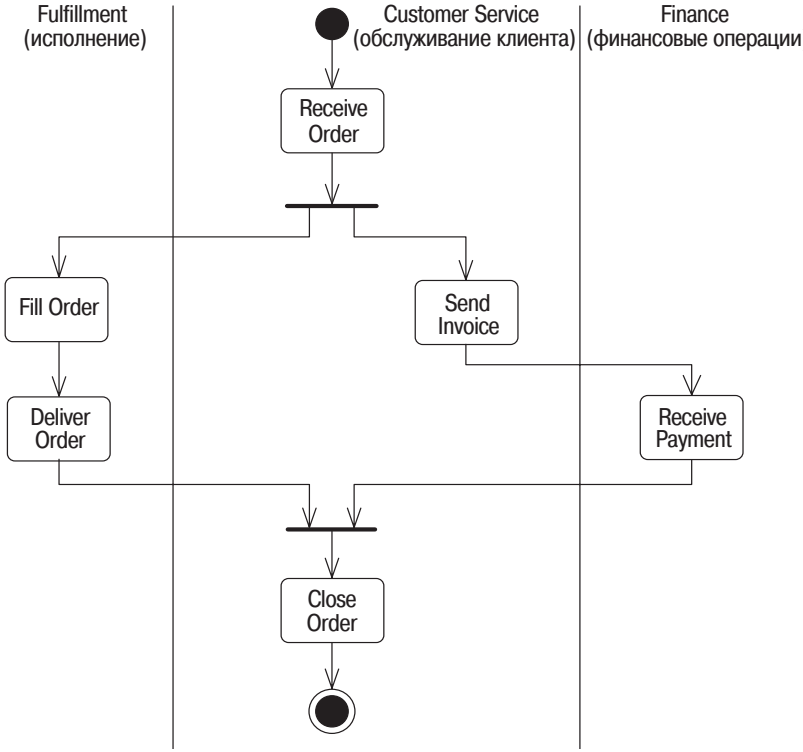

Рис. 11.4. Разбиение диаграммы деятельности на разделы

На рис. 11.4 представлено простое одномерное разбиение. Этот способ по понятным причинам часто называют **плавательными дорожками** (swim lanes), и такая форма была единственной в UML 1. В UML 2 сетка может быть двумерной, поэтому «плавательная» метафора больше не содержит воды. Кроме того, можно взять каждое измерение и разделить строчки на столбцы, создавая тем самым иерархическую структуру.

Сигналы

В простом примере на рис. 11.1 диаграммы деятельности имеют четко определенную стартовую точку, соответствующую вызову программы или процедуры. Кроме того, **операции могут отвечать на** сигналы.

Временной сигнал (time signal) приходит по прошествии времени. Такие сигналы могут означать конец месяца в отчетном периоде или приходить каждую секунду в контроллере реального времени.

На рис. 11.5 показана деятельность, в которой ожидаются два сигнала. Сигнал показывает, что данная деятельность принимает сообщение о событии от внешнего процесса. Это означает, что деятельность постоянно прослушивает эти сигналы, а диаграмма определяет, как деятельность на них реагирует.

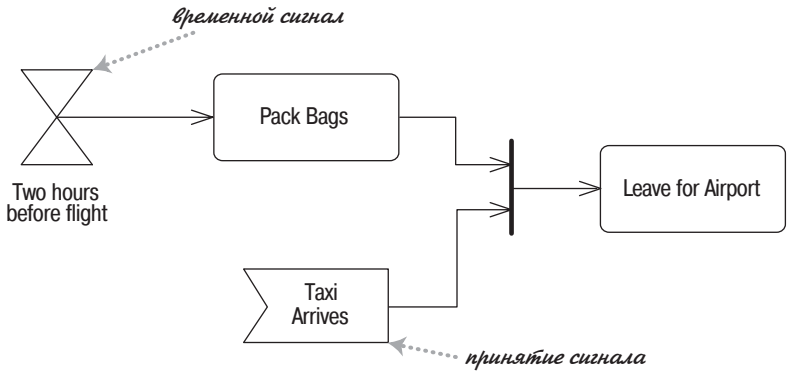

Рис. 11.5. Сигналы в диаграмме деятельности

В случае, показанном на рис. 11.5, до моего отлета остается два часа (`Two hours before flight`), и мне пора собирать багаж. Если я упакую его раньше времени, то все равно не смогу уехать, пока не прибудет такси. Если такси приходит (`Taxi Arrives`) до того, как я успею собрать багаж (`Pack Bags`), то оно должно ждать меня, пока я не закончу.

Мы можем как принимать сигналы, так и посылать их. Это полезно, когда мы посылаем сообщение, а затем должны ожидать ответа, перед тем как продолжить. На рис. 11.6 показан хороший пример этого процесса, основанный на общей идиоме таймаутов. Заметим, что в этой

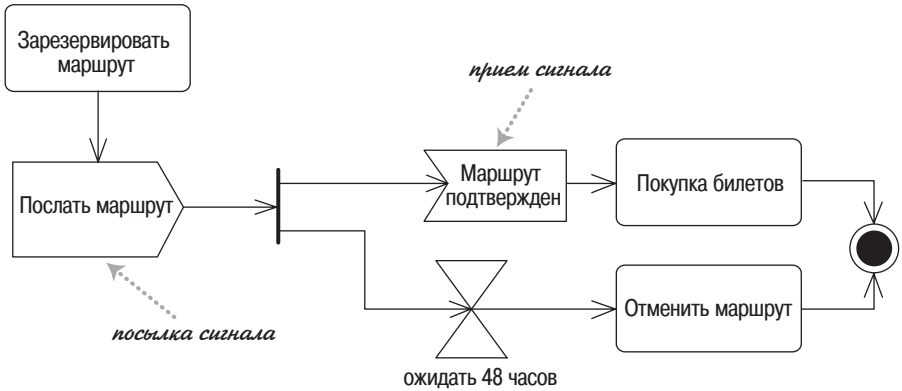

Рис. 11.6. Отправка и прием сигналов

гонке участвует два потока: первый, достигший финального состояния, выигрывает и прерывает выполнение другого потока.

Хотя блоки приема сигналов только ожидают внешнего события, мы можем также показать входящий в него поток. Это означает, что мы не начинаем прослушивание до тех пор, пока поток не инициирует прием.

Маркеры

Смельчаки, рискнувшие погрузиться в дьявольскую глубину спецификаций UML, обнаружат, что в разделе, посвященном активности, много говорится о маркерах (tokens), их создании и использовании. **Начальный узел создает маркер, который затем передается следующей процедуре, которая выполняется и передает маркер следующей процедуре. В точку ветвления приходит один маркер, а ветвление порождает маркер для каждого исходящего потока. Наоборот, в точке объединения при появлении отдельного маркера ничего не происходит до тех пор, пока не соберутся все маркеры, затем порождается маркер для исходящего потока.**

Можно визуализировать маркеры с помощью монеток или счетчиков, перемещающихся по диаграмме. В случае более сложных диаграмм деятельности маркеры часто облегчают визуализацию.

Потоки и ребра

В UML 2 параллельно употребляются термины поток (flow) и ребро (edge) для обозначения связи между двумя операциями. Самый простой вид ребра – это обычная стрелка между двумя операциями. Если хотите, можете присвоить ей имя, но в большинстве случаев простой стрелки будет достаточно.

При возникновении трудностей с разводкой линий можно воспользоваться разъемами (connectors), которые позволят вам не рисовать линии на всем их протяжении. Разъемы изображаются парами: один для входного и один для выходного потоков, при этом они должны иметь одну и ту же метку. Я предпочитаю без необходимости не применять разъемы, поскольку они нарушают визуализацию потока управления.

Простейшие ребра передают маркер, имеющий значение только для управления потоком. Однако по ребрам можно передавать объекты; тогда объекты будут играть роль маркеров как передатчиков данных. Передачу объекта вдоль ребра можно показать, помещая на ребро прямоугольник класса. Можно также изображать контакты (pins) на операциях, хотя использование контактов имеет некоторые хитрости, о которых я кратко расскажу.

Все способы, показанные на рис. 11.7, эквивалентны. Вы вправе выбрать тот способ, который лучше всего отражает то, что вы хотите сообщить. В большинстве случаев вполне достаточно простой стрелки.

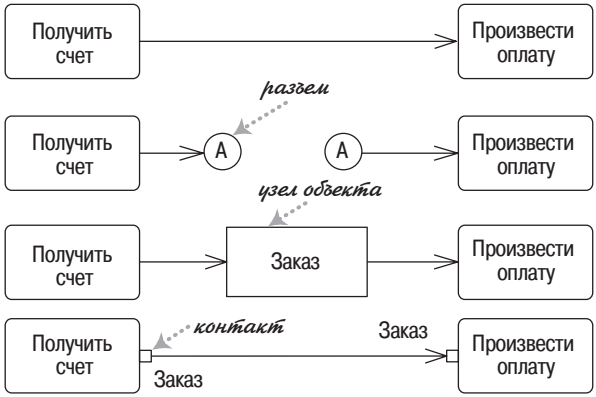

Рис. 11.7. Четыре способа представления ребер

Контакты и преобразования

Процедуры, как и методы, могут иметь параметры. Показывать на диаграмме деятельности информацию о параметрах не обязательно, но при желании можно отобразить параметры с помощью **контактов** (pins). Если процедура разбивается на части, то контакты должны соответствовать прямоугольникам параметров на разделенной диаграмме.

Если требуется нарисовать точную диаграмму деятельности, то необходимо обеспечить соответствие выходных параметров одной процедуры входным параметрам другой. Если они не совпадают, то можно указать преобразование (transformation) (рис. 11.8) для перехода от одной процедуры к другой. Преобразование должно представлять собой

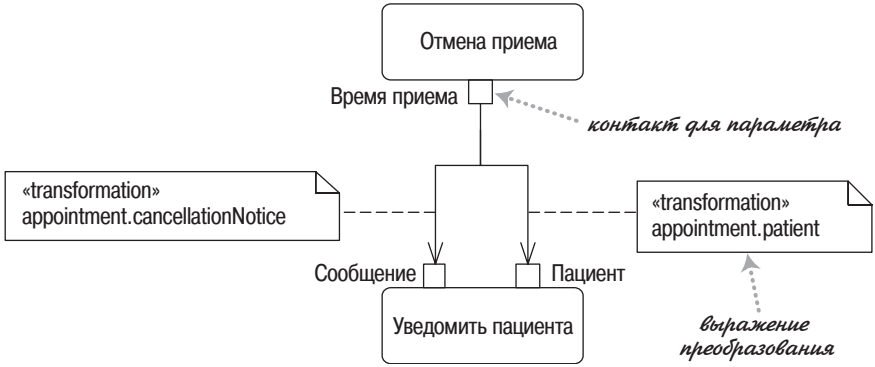

Рис. 11.8. Преобразование потока

выражение, свободное от сторонних эффектов: главное, чтобы запрос с выходного контакта предоставлял тип объекта, соответствующий входному контакту.

Показывать контакты на диаграмме деятельности не обязательно. Контакты удобны, когда требуется увидеть данные, принимаемые и передаваемые различными процедурами. При моделировании бизнес-процессов посредством контактов можно отображать ресурсы, которые потребляются и производятся различными процедурами.

С помощью контактов можно без опаски показать несколько потоков, входящих в одну и ту же операцию. Нотация контактов усиливает предположение о наличии последующего объединения, а в UML 1 вообще нет контактов, поэтому не возникает путаницы с более ранними допущениями.

Области расширения

При работе с диаграммами деятельности часто сталкиваешься с ситуациями, когда выход одной операции инициирует многочисленные вызовы другой операции. Есть несколько способов показать это, но лучше всего подходит область расширения. Область расширения (expansion region) отмечает область диаграммы деятельности, где операции выполняются один раз для каждого элемента коллекции.

На рис. 11.9 процедура `Choose Topics` (Выбрать темы) генерирует список тем. Затем каждый элемент этого списка становится маркером для входа процедуры `Write Article` (Написать статью). Подобным образом каждая операция `Review Article` (Рецензировать статью) генерирует единственную статью, которая добавляется к выходному списку области расширения. Когда все маркеры в области расширения достигают выходной коллекции, область расширения генерирует единственный маркер для списка, который передается процедуре `Publish Newsletter` (Опубликовать информационный бюллетень).

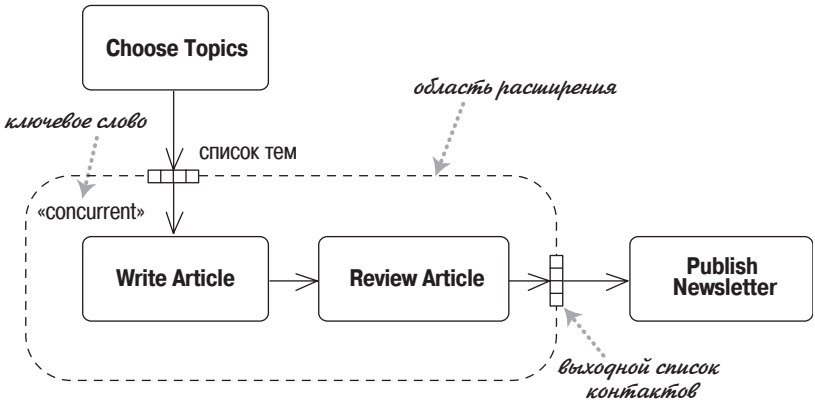

Рис. 11.9. Область расширения

В данном случае в выходной и входной коллекциях одинаковое количество элементов. Однако в выходной коллекции может оказаться меньше элементов, чем во входной; в таком случае область расширения действует как фильтр.

На рис. 11.9 все статьи пишутся и рецензируются параллельно, что от- мечено ключевым словом «concurrent». Область расширения также мо- жет быть итеративной. Итеративные области должны полностью обра- батывать каждый входной элемент за один раз.

Если есть только одна операция, которую надо вызывать несколько раз, то применяется нотация, показанная на рис. 11.10. Такой способ записи предполагает параллельное расширение, поскольку оно наиболее общее. Эта нотация соответствует концепции динамического параллелизма, принятой в UML 1.

Рис. 11.10. Нотация для единственной процедуры в области расширения

Окончание потока

При получении нескольких маркеров, как в случае с областью расширения, поток часто останавливается, даже если вся активность в целом не завершена. Окончание потока (flow final) означает завершение конкретного потока без завершения всей активности.

На рис. 11.11 показана модифицированная версия рис. 11.9, в которой статьи могут быть отвергнуты. Если статья отклонена, то маркер ликвидируется окончанием потока. В отличие от окончания активно-

сти, в данном случае остальная активность может продолжаться. Этот подход позволяет областям расширения действовать в качестве фильтров, в результате чего выходная коллекция будет меньше входной.

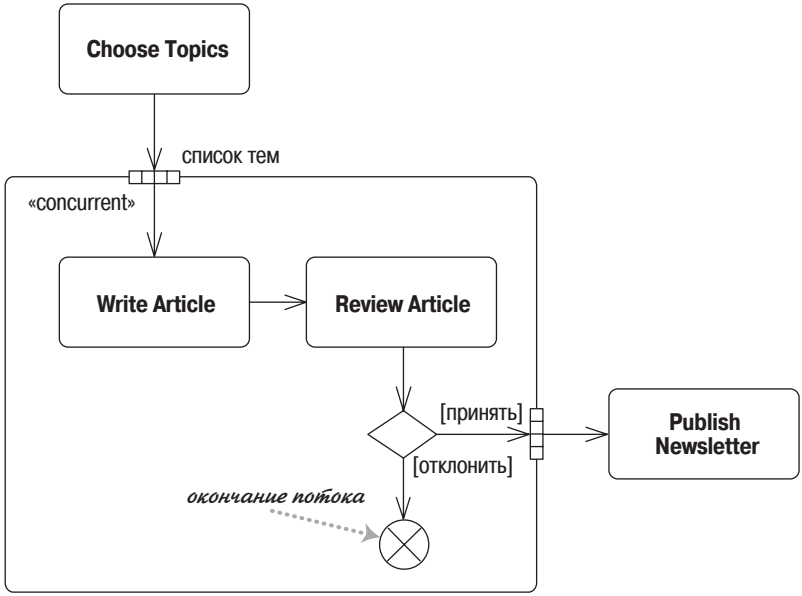

Рис. 11.11. Окончание потока в активности

Описания объединений

По умолчанию объединение разрешает выполнение выходного потока, когда все входные потоки достигли объединения. (Или, говоря более формальным языком, оно порождает маркер выходного потока, когда приходят маркеры всех входных потоков.) В некоторых случаях, в частности, когда есть поток с несколькими маркерами, полезно иметь более сложное правило.

Описание объединения (join specification) – это логическое выражение, присоединенное к объединению. Каждый раз, когда в объединение прибывает маркер, вычисляется описание объединения, и если его значение истинное, то порождается маркер. Поэтому на рис. 11.12, независимо от того, выбираю ли я напиток (`Select Drink`) или кидаю монетку (`Insert Coin`), автомат оценивает определение объединения. Автомат утоляет мою жажду, только если я кинул достаточное количество денег. Если, как в данном случае, вы хотите показать, что вы приняли маркер в каждом входном потоке, то необходимо именовать потоки и включить их в описание объединения.

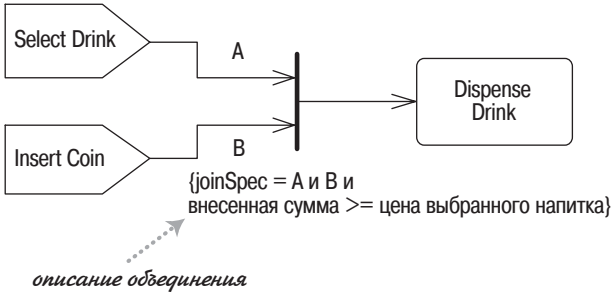

Рис. 11.12. Описание объединения

И еще немного

Я должен подчеркнуть, что эта глава лишь слегка затронула диаграммы деятельности. Учитывая объем языка UML, можно написать целую книгу только об одной этой технологии. На самом деле я думаю, что диаграммы деятельности могли бы стать отличной темой для книги, посвященной пристальному рассмотрению нотаций и их применению.

Жизненно важный вопрос заключается в том, как широко они применяются. **В настоящий момент диаграммы деятельности не относятся к наиболее распространенным технологиям языка UML, и все их предшественники в области моделирования потоков также не пользовались успехом.** Технология диаграмм еще не достигла необходимого уровня для описания поведения таким способом. С другой стороны, в многочисленных сообществах есть проявления скрытой потребности, которую могли бы удовлетворить стандартные приемы.

Когда применяются диаграммы деятельности

Самое большое достоинство диаграмм деятельности заключается в том, что они поддерживают и стимулируют применение параллельных процессов. Именно благодаря этому они представляют собой мощное средство моделирования потоков работ. Множество импульсов к развитию UML 2 пришло от людей, вовлеченных в эти потоки работ.

Можно также применять диаграмму деятельности в качестве совместимой с языком UML блок-схемы. Хотя это позволяет разрабатывать блок-схемы, близкие к UML, но вряд ли это очень захватывающий процесс. В принципе, можно воспользоваться преимуществами, предоставляемыми ветвлением и объединением, для описания параллельных алгоритмов одновременно выполняющихся программ. Хотя сам я не очень активно применял параллельные циклы, но у меня также нет достаточного количества подтверждений этого от людей, имеющих большой опыт их применения. Я думаю, **причина в том, что сложность**

параллельного программирования состоит в противостоянии данных параллельных процессов, а диаграммы деятельности не могут оказать большой помощи в этом вопросе.

В наибольшей степени их мощь может проявиться в случае применения UML как языка программирования. Здесь диаграммы деятельности являются ценным инструментом для представления логики поведения систем.

Мне часто приходилось видеть, как диаграммы деятельности применялись для описания прецедентов. Опасность такого подхода в том, что часто эксперты в предметной области с трудом могут им следовать. Если дело обстоит так, то лучше обойтись обычной текстовой формой.

Где найти дополнительную информацию

Хотя диаграммы деятельности всегда были довольно сложным инструментом, а в UML 2 они стали еще сложнее, тем не менее не существует хорошей книги, в которой бы они описывались достаточно глубоко. Я надеюсь, что этот пробел когда-нибудь будет восполнен.

Различные ориентированные на потоки технологии по своему стилю напоминают диаграммы деятельности. Одна из лучших (но едва ли широко известная) – это **Petri Nets**, информацию о которой можно найти на веб-сайте *http://www.daimi.au.dk/PetriNets/*.

12

Коммуникационные диаграммы

Коммуникационные диаграммы (communication diagrams) – это особый вид диаграмм взаимодействия, акцентированных на обмене данными между различными участниками взаимодействия. Вместо того чтобы рисовать каждого участника в виде линии жизни и показывать последовательность сообщений, располагая их по вертикали, как это делается в диаграммах последовательности, коммуникационные диаграммы допускают произвольное размещение участников, позволяя рисовать связи, показывающие отношения участников, и использовать нумерацию для представления последовательности сообщений.

В UML 1.x эти диаграммы назывались диаграммами кооперации (collaboration diagrams). Это подходящее название, и я подозреваю, что оно будет существовать, пока люди не привыкнут к новому названию. (Между этим понятием и кооперацией есть различие (*стр. 161*); отсюда изменение названия.)

На рис. 12.1 приведена коммуникационная диаграмма для централизованного управления, показанного на **рис. 4.2**. С помощью коммуникационной диаграммы можно увидеть, как участники связаны друг с другом.

Кроме отображения связей, которые представляют собой экземпляры ассоциаций, можно также показать временные связи, возникающие только в контексте взаимодействия. В данном случае связь `«local»` (локальная) от объекта `Order` (Заказ) к объекту `Product` (Продукт) – это локальная переменная, а другими временными связями являются `«parameter»` (параметр) и `«global»` (глобальная). Эти ключевые слова употреблялись в UML 1, но пропали из UML 2. Они полезны, поэтому я на- деюсь, что разработчики от них не откажутся.

Стиль нумерации на рис. 12.1 простой и общеупотребительный, но в языке UML он не разрешен. В соответствии с правилами UML необходимо придерживаться вложенной десятичной нумерации, как показано на рис. 12.2.

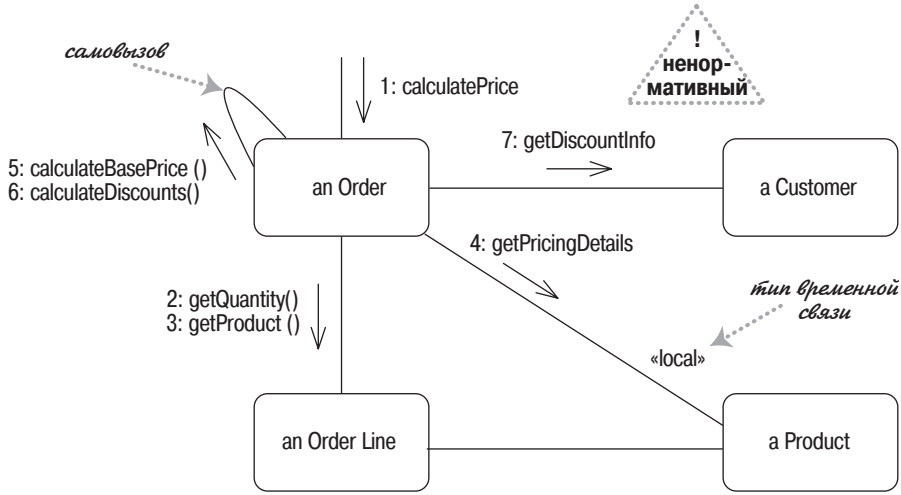

Рис. 12.1. *Коммуникационная диаграмма системы централизованного управления*

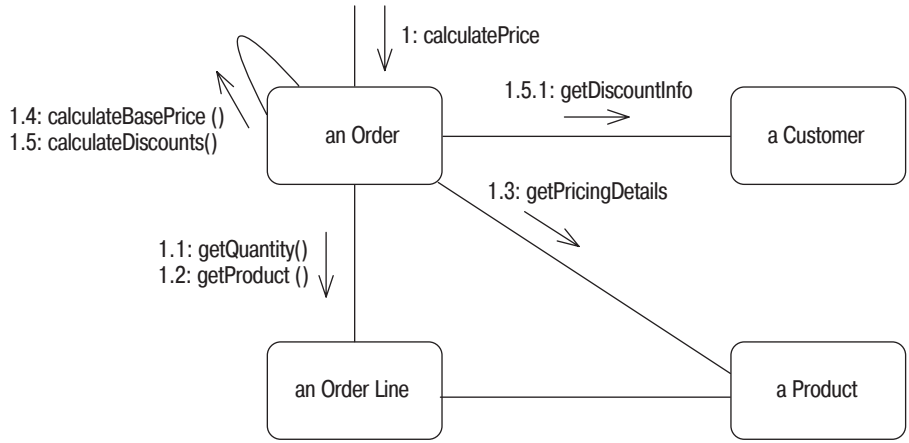

Рис. 12.2. *Коммуникационная диаграмма с вложенной десятичной нумерацией*

Вложенная десятичная нумерация нужна, потому что требуется исключить неопределенность при самовызовах. На рис. 4.2 четко показано, что метод `getDiscountInfo` **вызывается из метода** `calculateDiscount`**. Однако в случае применения линейной нумерации, как на рис. 12.1, нельзя будет сказать, вызывается ли метод** `getDiscountInfo` **из метода** `calculateDiscount` **или из более общего метода** `calculatePrice`**. Схема вложенной нумерации позволяет обойти эту трудность.**

Несмотря на ее неправомерность, многие специалисты предпочитают линейную схему нумерации. **Вложенная нумерация может быть очень**

сложной, в частности потому, что вызовы могут иметь большой уровень вложенности, что приводит к такой последовательности номеров, как 1.1.1.2.1.1. В таких случаях лекарство от неопределенности хуже болезни.

Кроме чисел в сообщениях можно увидеть и буквы. Эти символы обозначают различные потоки управления. Так, A5 и B2 могут быть различными потоками; сообщения 1a1 и 1b1 могут быть различными потоками, параллельно вложенными в сообщение 1. Буквы, обозначающие потоки, можно увидеть также и на диаграммах последовательности, хотя это и не дает визуального представления о параллельности.

Коммуникационные диаграммы не имеют точно определенных нотаций для управляющей логики. Они допускают применение маркеров итерации и защиты, но не позволяют полностью определить алгоритм управления. Не существует также специальных обозначений для создания и удаления объектов, но ключевые слова `«create»` **и** `«delete»` **соответствуют общепринятым соглашениям.**

Когда применяются коммуникационные диаграммы

Основной вопрос, связанный с коммуникационными диаграммами, заключается в том, в каких случаях надо предпочесть их, а не более общие диаграммы последовательности. Ведущую роль в принятии такого решения играют личные предпочтения: у людей разные вкусы. Чаще всего именно это определяет тот или иной выбор. В целом, большинство специалистов, по-видимому, предпочитает диаграммы последовательности, а что касается меня, то я поддерживаю большинство.

Более рациональный подход утверждает, что **диаграммы последовательности удобнее, если вы хотите подчеркнуть последовательность вызовов, а коммуникационные диаграммы лучше выбрать, когда надо акцентировать внимание на связях. Многие специалисты считают, что коммуникационные диаграммы проще модифицировать на доске, поэтому они хорошо подходят для рассмотрения вариантов,** хотя я в таких случаях обычно предпочитаю CRC-карточки.