3 Диаграммы пакетов

Пакеты предоставляют механизм группировки взаимосвязанных элементов UML и ограничения видимости их имен. Например, все элементы, связанные с 3D-моделированием, можно разместить в пакете с именем 3DGraphics. Диаграммы пакетов обеспечивают отличную возможность визуального представления зависимостей между частями вашей системы и часто используются для диагностики или определения порядка компиляции.

В пакетах могут группироваться практически любые элементы UML (в том числе и сами пакеты). Каждый пакет обладает именем, уточняющим область видимости каждого элемента пакета. Например, если класс с именем Timer принадлежит пакету Utilities, полное имя класса имеет вид Utilities::Timer. Элементы, входящие в один пакет, могут обращаться друг к другу без уточнения имен.

Представление

Пакет изображается в виде прямоугольника с «корешком» в левому верхнему углу. На рис. 3.1 изображен пакет Utilities.

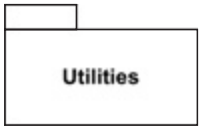

Рис. 3.1. Простой пакет

Существуют два разных способа показа элементов, входящих в пакет. В первом варианте элементы рисуются внутри большого прямоугольника. Если вы используете эту запись, запишите имя пакета на корешке. На рис. 3.2 показано содержимое пакета Utilities. Для обращения к классу Timer из-за границ пакета Utilities используется запись Utilities::Timer.

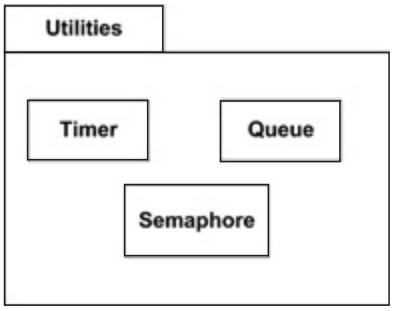

Рис. 3.2. Пакет Utilities содержит несколько классов

Во втором варианте записи пакет соединяется с каждым содержащимся в нем элементом сплошной линией. Ближний к пакету конец линии помечается кружком со знаком «+», обозначающим включение. При использовании этой записи имя пакета указывается в большом прямоугольнике, а не на корешке. Такой вариант записи позволяет отображать более подробную информацию о содержимом пакета. На рис. 3.3 показан тот же пакет Utilities с выделением классов.

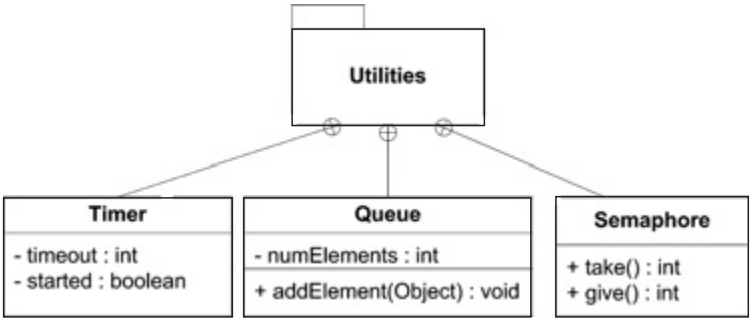

Рис. 3.3. Элементы, принадлежащие пакету, изображаются за пределами пакета

Изображать все элементы, содержащиеся в пакете, необязательно. Если элементы не показаны, не следует делать никаких допущений относительно содержимого пакета.

Видимость

Пакет может задавать уровень видимости входящих в него и импортированных элементов, однако элементы могут находиться только на одном из двух уровней: открытом или приватном. Открытая видимость означает, что элемент может использоваться за пределами пакета (с уточнением его имени именем пакета). Элементы с приватной видимостью могут использоваться только другими эле-

ментами того же пакета. Приватная видимость хорошо подходит для вспомогательных классов, задействованных в реализации подсистемы или компонента; для других частей системы эти классы должны быть недоступны.
Открытая видимость обозначается знаком «+» перед именем элемента, а приватная — знаком «–». На рис. 3.4 изображен пакет Utility с приватными вспомогательными классами.

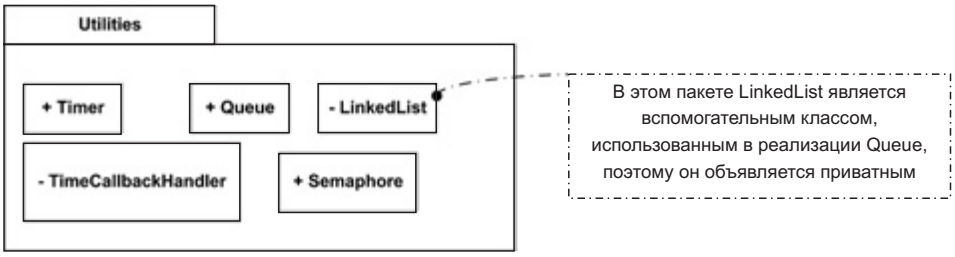

Рис. 3.4. Пакет Utilities с открытыми и приватными членами

Импортирование и доступность пакетов

При обращении к элементу одного пакета из другого пакета необходимо указывать уточненное имя элемента. Например, если класс Car входит в пакет Transportation, то для обращения к нему из другого пакета RoutePlanning имя Car уточняется до Transportation::Car.

Для упрощения работы с элементами других пакетов в UML предусмотрена возможность *импортирования* пакетов. Элементы импортированного пакета доступны в импортирующем пакете без уточнения имен. Таким образом, если пакет RoutePlanning импортирует пакет Transportation, вы сможете обращаться к Car из RoutePlanning без уточнения имени.
Импортирование пакета обозначается пунктирной линией со стрелкой, проведенной от импортирующего пакета к импортированному. Линия помечается ключевым словом «import». На рис. 3.5 изображен пакет RoutePlanning, импортирующий пакет Transportation.

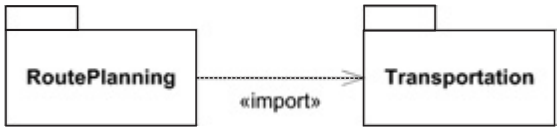

Рис. 3.5. Пакет RoutePlanning, импортирующий пакет Transportation

По умолчанию импортированные элементы обладают открытым уровнем видимости в импортирующем пакете. UML также позволяет указать, чтобы импор-

тированным элементам назначался приватный уровень видимости, то есть они не могли использоваться за пределами импортирующего пакета (включая все пакеты, которые могут импортировать этот пакет). Если импортированным элементам должен присваиваться приватный уровень видимости, вместо ключевого слова «import» используется ключевое слово «access». На рис. 3.6 изображен пакет RoutePlanning, который импортирует пакет Transportation и обращается к пакету Algorithms. Если пакет импортирует RoutePlanning, оба пакета могут использовать открытые элементы из Transportation, но не могут использовать ничего из Algorithms. Если пакет импортирует RoutePlanning, оба пакета могут использовать открытые элементы из Transportation, но содержимое Algorithms для них недоступно.

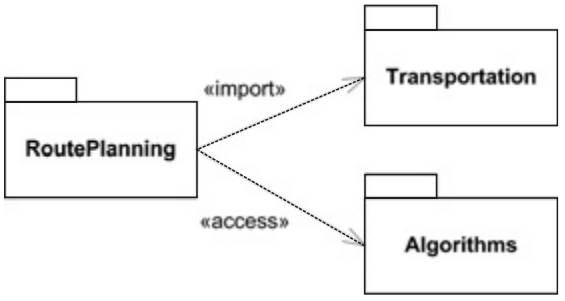

Рис. 3.6. RoutePlanning импортирует пакет Transportation, но ограничивается простым обращением к пакету Algorithms

Различия между импортированием и простым обращением к пакету означают, что реализация может значительно изменяться в зависимости от выбранного языка. Например, в C# и Java существуют явно определенные концепции пакетов и импортирования элементов между пакетами. Разработчики Java часто импортируют в свои программы пакет java.util, чтобы работать с классом Java Vector без указания полного имени. Однако в C++ существует более тонкая концепция пакетов — так называемые *пространства имен* (namespaces). Выбор конкретного способа представления пакетов в языке реализации часто предоставляется разработчику.

Слияние пакетов

В UML также поддерживается довольно непростая концепция слияния пакетов. Слияние пакетов отличается от импортирования тем, что *слияние* по определению приводит к установлению отношений между одноименными классами. Поддержка слияния пакетов появилась непосредственно в результате перехода от UML 1.x к 2.0. UML 2.0 определяет базовую концепцию элементов и позволяет специализированным разновидностям диаграмм расширять базовую концепцию без необходимости предоставлять для нее новое имя. Например, UML расширяет

некоторые основополагающие концепции поведенческого конечного автомата до протокольного конечного автомата, но при этом сохраняет исходное название.
При слиянии пакета с другим пакетом все классы с одинаковым типом и именем автоматически расширяют (или устанавливают отношение обобщения) с исходным классом. Например, UML может определить концепцию отношения include на обобщенном уровне, а затем специализировать ее для конкретных вариантов использования с сохранением названия include. Подобные расширения упрощают внутреннее строение модели UML, но редко используются в реальном программировании.
Слияние пакетов обозначается пунктирной линией с незамкнутой стрелкой от пакета, выполняющего слияние (подключающего пакета), к пакету, задействованному в слиянии (подключаемому пакету). Линия помечается ключевым словом «merge». На рис. 3.7 представлен пример слияния двух пакетов.

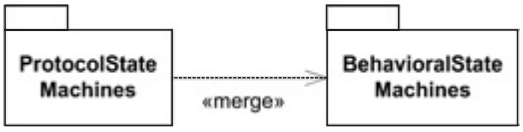

Рис. 3.7. Пакет ProtocolStateMachines подключает элементы из пакета BehavioralStateMachines, чтобы добавить их к содержащимся в нем классам

Правила слияния пакетов:

- приватные члены пакета не участвуют в слиянии;
- классы подключающего пакета, которые по именам и типам совпадают с классами подключаемого пакета, вступают в отношение обобщения с подключаемыми классами. Обратите внимание: это может привести к множественному наследованию, но в UML оно не запрещено;
- чтобы обратиться к исходной версии классов, уточните ее имя именем исходного пакета;
- классы, существующие только в одном из двух пакетов (исходном или подключаемом), остаются без изменений;
- субпакеты, входящие в подключаемый пакет, добавляются в исходный пакет, если они уже не существуют в нем;
- если в исходном пакете уже существует субпакет с тем же именем, происходит слияние двух субпакетов;
- все операции импортирования из подключаемого пакета преобразуются в операции импортирования из исходного пакета. Импортированные элементы не сливаются (то есть не вступают в отношения обобщения). Если импортированный элемент конфликтует с элементом исходного пакета, предпочтение отдается элементу исходного пакета, а для обращения к импортированному элементу требуется полное уточнение имени.

Разновидности диаграмм пакетов

В этом разделе представлены два варианта применения диаграмм пакетов с классами, и одно применение пакетов с вариантами использования. Хотя основными побудительными причинами для использования пакетов являются иерархическая структура и упрощение, сам термин «упрощение» понимают по-разному. В него могут вкладывать следующий смысл:

- простота построения и тестирования;
- более удобное отслеживание, прозрачность проекта;
- работа на обобщенном, абстрактном уровне без низкоуровневого «шума»;
- снижение вероятности конфликтов между распределенными группами;
- простота рефакторинга и расширения.

Вероятно, какая-то часть аудитории понимает упрощение как более доступную сложность. Если ваши пакеты не обеспечивают баланс этих разнообразных потребностей, скорее всего, вы будете получать жалобы на чрезмерную сложность модели, какими бы благородными мотивами вы при этом ни руководствовались.

Структурирование проекта с использованием диаграмм пакетов

Пакеты классов обеспечивают логическую организацию системы в фазе конструирования. Они предоставляют информацию для руководства и внешних участников, а также структуру, содержащую все классы. Диаграммы пакетов классов являются UML-эквивалентом классических блочных диаграмм.

Разные стороны рассматривают проект с разных позиций. Программисты думают в контексте языка программирования и «тактического» проектирования. Архитекторы думают в контексте зависимостей, рисков, технологии, построения, тестирования и принципов ООП. Руководители проектов думают в контексте рисков, отслеживания, ресурсов, потребностей, принадлежности, необходимых навыков и соблюдения сроков. Хотя все эти факторы важны, а при проектировании хорошей пакетной схемы учитываются все потребности, архитекторы склонны выделять пакетные структуры верхнего уровня с прицелом на контрольные функции управления проектом. Например, руководитель проекта может нарисовать примерно такую пакетную диаграмму веб-приложения, как показано на рис. 3.8.

Диаграмма на рис. 3.8 сама по себе передает очень мало полезной информации. К ней должен прилагаться текстовый документ с описанием основных принципов разбиения на пакеты. Такой документ, например, может содержать список следующего вида:

- web — требует специальных навыков: HTML, CSS и Struts, технология представления; максимум зависимостей;
- database — требует навыков управления базами данных и моделирования; минимум зависимостей;

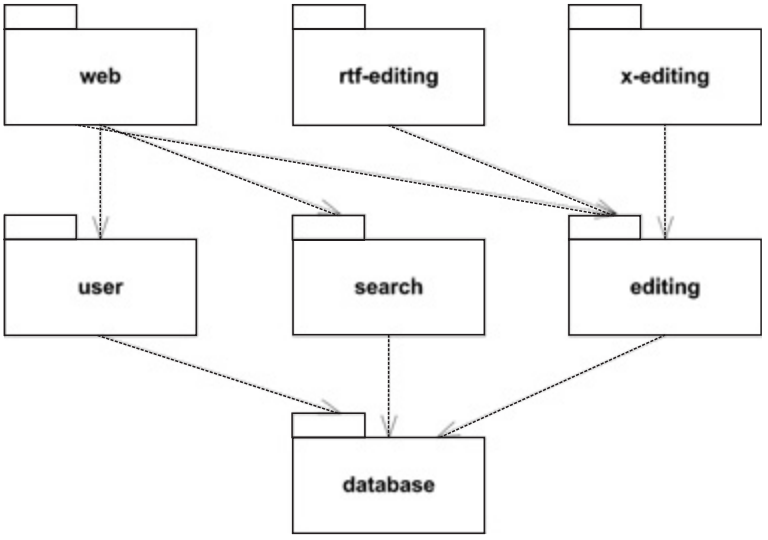

Рис. 3.8. Пакеты верхнего уровня, определяющие веб-приложение

- user — будет создаваться сторонней группой разработчиков;
- search — требует знакомства с технологией и методами использования поисковых систем; автономная подсистема;
- editing — основные функции редактирования, включаемые в первую версию; разные навыки, отдельная команда;
- rtf-editing — функции редактирования, запланированные к выходу версии 2.
- x-editing — функции редактирования, запрашиваемые конкретным клиентом. Функциональность может быть отозвана или отложена в зависимости от клиента. Риски не зависят от других компонентов.

На рис. 3.8 не показан полный набор пакетов системы. Того, что показано, достаточно только для отслеживания и руководства проектом. Программисты создают пакеты более низкого уровня, если того потребует архитектура кода. Однако руководство распределяет ресурсы и отслеживает прогресс на уровне относительно крупных проектных блоков, показанных на рис. 3.8, не отвлекаясь на добавление или рефакторинг внутренних пакетов и содержимого.

Пакеты вариантов использования

Пакеты вариантов использования структурируют функциональное поведение системы на стадии анализа. На рис. 3.9 показаны основные функциональные области системы управления контентом (CMS). Пакеты предоставляют доступную информацию для сотрудников, не входящих в аналитическую группу. Руководители могут обсуждать проект на соответствующем уровне детализации, не отвлекаясь на технические подробности.

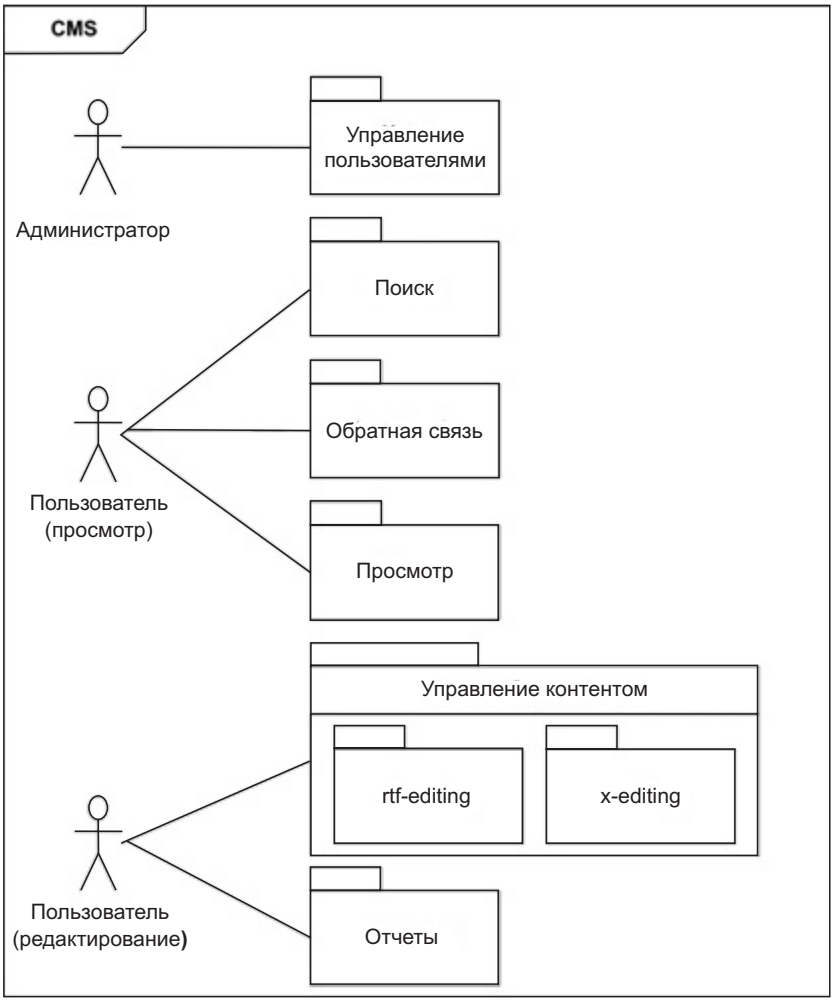

Рис. 3.9. Основные пакеты вариантов использования

Пакеты системы CMS, показанной на рис. 3.9, создавались по следующим соображениям:

- пакеты нетривиального редактирования отделяют поставку предварительной и расширенных версий от поставки, ориентированной на конкретного клиента;
- простые пакеты взаимодействия, просмотра и обратной связи содержат базовые функции с низким риском;
- построение отчетов содержит более сложные функции;
- поиск и управление пользователями обеспечивают разделение сложных функций.

Как и в случае с пакетами классов на рис. 3.8, управление проектом осуществляется на уровне пакетов вариантов использования. Помните, что идентификация пакетов вариантов использования означает идентификацию потребительской полезности.

Направленные графы зависимостей

Направленные графы зависимостей в пакетах системы отражают аспекты, не связанные с функциональностью и относящиеся к удобству построения, тестируемости и ошибкоустойчивости. Предоставляемая ими информация предназначена для различных ролей участников разработки программного проекта.

Если все зависимости следуют в одном направлении, как на рис. 3.10, без циклов и «петель», граф называется *ациклическим*. Ациклические графы являются признаком качественного проекта.

Направленные графы зависимостей помогают избежать проблем со сценариями сборки и тестированием проекта. Взглянув на рис. 3.10, мы видим, что рефакторинг timer аннулирует результаты тестирования и теоретически может привести к нарушениям сборки visualizers, threads, controllers и top; следовательно, при изменении пакета timer необходима осторожность.

Направленные графы зависимостей также помогают распределять работу над проектом среди его участников. Например, из рис. 3.10 видно, что зависимости между threads и visualizers отсутствуют. Таким образом, работу над этими двумя пакетами можно поручить разным группам, и они не будут мешать друг другу.

Со временем проекты развиваются; появляются новые зависимости, что может объясняться как недостаточно глубоким предварительным анализом, так и целесообразностью. Один из таких случаев показан на рис. 3.11; три двусторонние зависимости нарушают желательную, ациклическую природу графа.

- Utils зависит от threads.
- threads зависит от controllers.
- samplers зависит от threads.

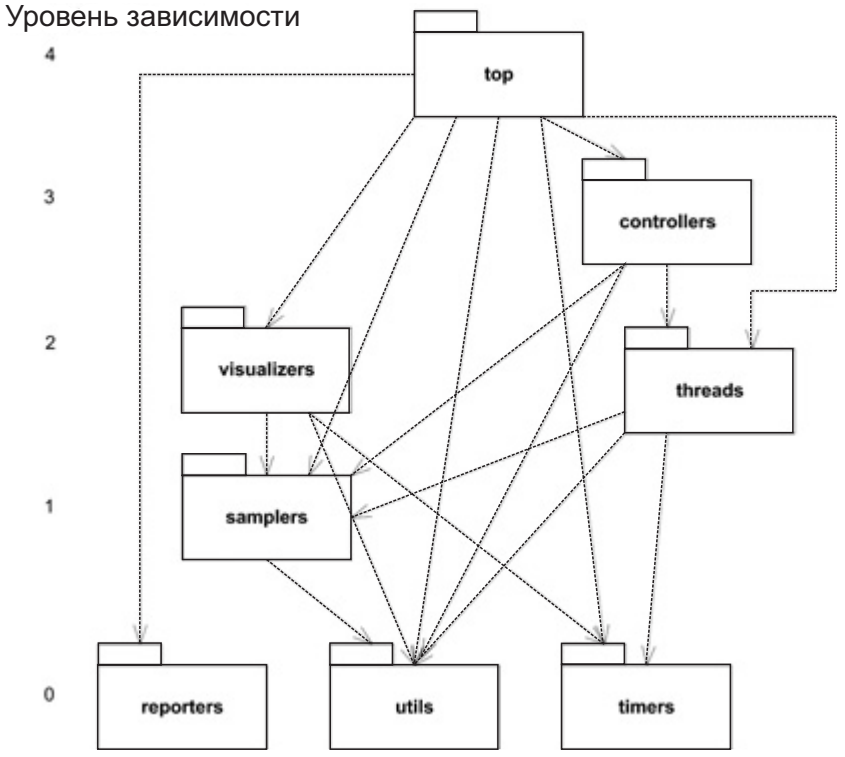

Рис. 3.10. Граф зависимостей JMeter без циклов

Дело в том, что на рис. 3.10 каждая из этих зависимостей уже существовала в обратном направлении. В результате на уровне 1 рис. 3.11 появляется сплетение двунаправленных стрелок, которое заметно отличается от четких, однозначных переходов на рис. 3.11. Все это оборачивается проблемами для разработчиков, группы тестирования, а скорее всего, и для руководства с пользователями. Изменения в threads будут отражаться на работе группы, ответственной за controllers и utils — и наоборот, потому что изменения в одном пакете потребуют изменений в коде или модульного тестирования других пакетов. Как следствие, полная стабилизация всех пакетов может наступить лишь после нескольких итераций[1].

[1] С теорией и метрическими характеристиками, относящимися к пакетам и их зависимостям, можно познакомиться в статье Джона Лакоса «Large-Scale C++ Software Design». Также обращайтесь к статье Роберта С. Мартина по адресу: http://c2.com/cgi/wiki?PrinciplesOfObjectOrientedDesign.