18 марта 2023 г.

Гаврилов А. Г. Новоселова О. В.

Современные технологии и средства разработки программного обеспечения (второй семестр)

Раздел 6. Современные технологии и средства программной реализации программного продукта

Тема 17
Проектирование реализации программного продукта на конкретном языке программирования. Построение диаграммы компонентов ПП с помощью языка моделирования UML

Содержание

1 Основные понятия

Компонент — это логическая замещаемая часть системы, которая соответствует некоторому набору интерфейсов и обеспечивает их реализацию.

Хорошие компоненты определяют четкие абстракции с хорошо определенными интерфейсами, что дает возможность легко заменять старые компоненты па совместимые с ними новые.

Интерфейсы соединяют логические модели с моделями дизайна. Так, например, вы можете специфицировать интерфейс класса в логической модели, и тот же интерфейс

будет поддерживаться некоторым компонентом дизайна, реализующим его.

Интерфейсы позволяют вам реализовывать компонент с использованием более мелких компонентов путем соединения их портов.

Интерфейс — это набор операций, которые специфицируют сервис, предоставляемый либо требуемый классом или компонентом.

Компонент — замещаемая часть системы, которая соответствует набору интерфейсов и обеспечивает его реализацию.

Порт — специфическое «окно» в инкапсулированный компонент, принимающее сообщения для компонента и от него в соответствии с заданным интерфейсом.

Внутренняя структура — реализация компонента, представленная набором частей, соединенных друг с другом конкретным способом.

Часть — спецификация роли, составляющей часть реализации компонента. В экземпляре компонента присутствует экземпляр, соответствующий части.

Коннектор — связь коммуникации между двумя частями или портами в контексте компонента.

2 Компоненты и интерфейсы

Итак, интерфейс — набор операций, используемый для спецификации сервиса класса или компонента. Связь между компонентом и интерфейсом имеет важное значение. Все основанные на компонентах средства операционных систем (такие, как COM+, CORBA и Enterprise Java Beans) используют интерфейсы в качестве элементов, связывающих компоненты друг с другом.

Чтобы сконструировать систему на основе компонентов, необходимо произвести ее декомпозицию, специфицируя интерфейсы, которые представляют основные соединения. Затем понадобится определить компоненты, реализующие интерфейсы, вместе с другими компонентами, которые имеют доступ к сервисам первых по средством своих интерфейсов. Этот механизм позволяет вам развернуть систему, сервисы которой в определенной мере независимы от местоположения и, как будет показано в следующем разделе, заменяемы.

Интерфейс, который реализован компонентом, называется предоставляемым (то есть данный компонент предоставляет интерфейс в виде сервиса другим компонентам). Компонент может декларировать множество предоставляемых интерфейсов. Интерфейс, который он использует, называется требуемым: ему соответствует данный компонент, когда запрашивает сервисы от других компонентов. Компонент может соответствовать множеству требуемых интерфейсов. Бывают компоненты, которые одновременно предоставляв ют и требуют интерфейсы.

Как показано на рис. 1, компонент изображается в виде прямоугольника с двузубчатой пиктограммой в правом верхнем углу. Внутри прямоугольника указывается имя компонента. У него могут быть атрибуты и операции, которые, впрочем, на диаграммах часто опускаются. Компонент может показывать сеть внутренней структуры, о чем пойдет речь чуть ниже.

Связь между компонентом и его интерфейсами выражается одним из двух способов. Первый более употребителен – интерфейс изображается в сокращенной, пиктографической форме. Предоставляемый интерфейс выглядит как кружок, соединенный линией с компонентом («леденец на палочке»). Требуемый интерфейс – полукруг, так же соединенный с компонентом («гнездо»). В обоих случаях имя интерфейса указано рядом с символом. Второй способ – изображение интерфейса в его расширенной форме (возможно, с указанием его операций). Компонент, который реализует интерфейс, соединяется с

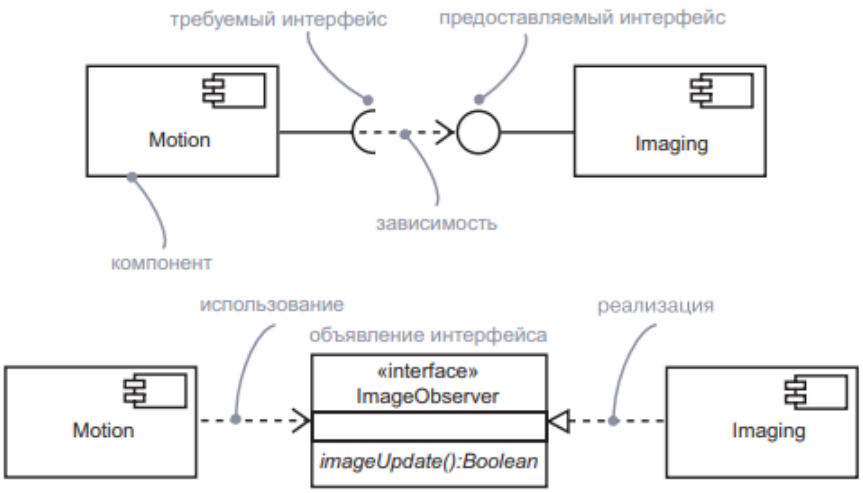

Рис. 1. Компоненты и интерфейсы

ним посредством полной связи реализации. Компонент, который имеет доступ к сервисам другого компонента через интерфейс, соединяется с интерфейсом связью зависимости.

Определенный интерфейс может быть предоставлен одним компонентом и затребован другим. Поскольку этот интерфейс находится между двумя компонентами, их прямая зависимость друг от друга разрушается. Компонент, использующий данный интерфейс, будет работать правильно независимо от того, какой компонент его реализует. Конечно, компонент может быть использован в этом контексте тогда и только тогда, когда все его требуемые интерфейсы реализованы в качестве предоставляемых интерфейсов в других компонентах.

Интерфейсы применяются на многих уровнях, как и другие элементы. Интерфейс уровня дизайна, который используется или реализован компонентом, на уровне реализации будет отображен на интерфейс, используемый или реализованный артефактом, который воплощает этот компонент.

3 Заменяемость

Основное назначение каждого средства операционной системы, основанной на компонентах, – обеспечить сборку системы из бинарных заменяемых артефактов. Это значит, что вы можете проектировать систему, используя компоненты и затем реализуя их в виде артефактов. Вы можете даже развивать систему, добавляя новые компоненты и заменяя старые, но не перестраивая всю ее целиком. Интерфейсы – ключевое средство, обеспечивающее такие возможности. В работающей системе допускается применение любых артефактов, которые реализуют компоненты, согласованные и пре- доставляющие нужный интерфейс. Расширение системы возможно за счет создания компонентов, предоставляющих новые сервисы через другие интерфейсы, которые прочие компоненты, в свою очередь, могут обнаружить и использовать. Эта семантика проясняет цели определения компонентов в UML. Компонент соответствует набору интерфейсов и обеспечивает его реализацию, что позволяет замещать его – как в логическом дизайне, так и в основанной на нем физической реализации.

Компонент замещаем – это значит, что его можно заменять другим компонентом, который соответствует тем же интерфейсам (в процессе проектирования вы выбираете иной компонент вместо данного). Обычно механизм вставки и замены артефакта в исполнимой системе прозрачен для пользователя компонента и допускается объектными моделями (та-

кими как COM+ и Enterprise Java Beans), которые требуют небольшой промежуточной трансформации, или осуществляется инструментами, автоматизирующими этот механизм.

Компонент – это часть системы – он редко используется сам по себе. Чаще он объединен с другими компонентами, то есть вовлечен в архитектурный или технологический контекст, где предполагается его использовать. Компонент логически и физически согласован и, таким образом, представляет структурный и/или поведенческий фрагмент более крупной системы. Во множестве систем он может быть использован повторно. Таким образом, компонент представляет собой фундаментальный строительный блок, на основе которого может быть спроектирована и составлена система. Это определение рекурсивно: то, что на одном уровне абстракции является системой, может быть компонентом на другом, более высоком уровне.

Наконец, как уже отмечалось в предыдущих разделах, компонент соответствует набору интерфейсов и обеспечивает его реализацию.

4 Организация компонентов

Вы можете организовать компоненты тем же способом, что и классы, – группируя их в пакеты. Кроме того, допускается организация компонентов путем установления между ними связей зависимости, обобщения, ассоциации (включая агрегацию) и реализации. Одни компоненты могут быть построены из других. Об этом пойдет речь чуть ниже, в разделе «Внутренняя структура».

5 Порты

Интерфейсы удобны для описания общего поведения компонента, но им не присуща «индивидуальность»: реализация компонента должна лишь гарантировать, что все операции во всех предоставляемых интерфейсах реализованы. Для более полного контроля над реализацией можно использовать порты.

Порт (port) – это своеобразное «окно» в инкапсулированный компонент. Все взаимодействие с таким компонентом на входе и на выходе происходит через порты. Выражаемое внешне поведение компонента представляет собой сумму его портов – ни более ни менее. Вдобавок к этому порт наделен уникальностью. Один компонент может взаимодействовать с другим через определенный порт. При этом их коммуникации полностью описываются интерфейсами, которые поддерживает порт, даже если компонент поддерживает другие интерфейсы. В реализации внутренние части компонента могут взаимодействовать друг с другом через специфический внешний порт, поэтому каждая часть может быть независима от требований других. Порты позволяют разделять интерфейсы компонента на дискретные пакеты и использовать обособленно. Инкапсуляция и независимость, обеспечиваемая портами, повышают степень заменяемости компонента.

Порт схематически представлен маленьким квадратом на боковой грани компонента – это отверстие в границе инкапсуляции компонента. Как предоставляемый, так и требуемый интерфейс может быть соединен с символом порта. Предоставляемый интерфейс изображает сервис, который может быть запрошен извне через данный порт, а требуемый интерфейс – сервис, который порт должен получить от какого либо другого компонента. У каждого порта есть имя, а следовательно, он может быть идентифицирован по компоненту и имени. Последнее могут использовать внутренние части компонента для идентификации порта, через который следует отправлять и получать сообщения. Имя компонента вместе с именем порта идентифицирует порт для использования его другими компонентами.

Порты – это часть компонента. Экземпляры портов создаются и уничтожаются вместе с экземпляром компонента, которому они принадлежат. Порты также могут иметь множественность; это означает возможность существования нескольких экземпляров порта внутри экземпляра компонента. Каждый порт компонента имеет соответствующий массив экземпляров. Хотя все экземпляры портов в массиве удовлетворяют одному и тому же интерфейсу и принимают запросы одних и тех же видов, они могут находиться в различных состояниях и иметь разные значения данных. Например, каждый экземпляр в массиве может иметь свой уровень приоритета (экземпляр порта с наибольшим уровнем приоритета обслуживается первым).

На рис. 2 представлена модель компонента Ticket Seller (Продавец билетов) с портами. У каждого порта есть имя и необязательный тип, показывающий, каково назначение данного порта. Компонент имеет порты для продажи билетов, объявлений и обслуживания кредитных карт.

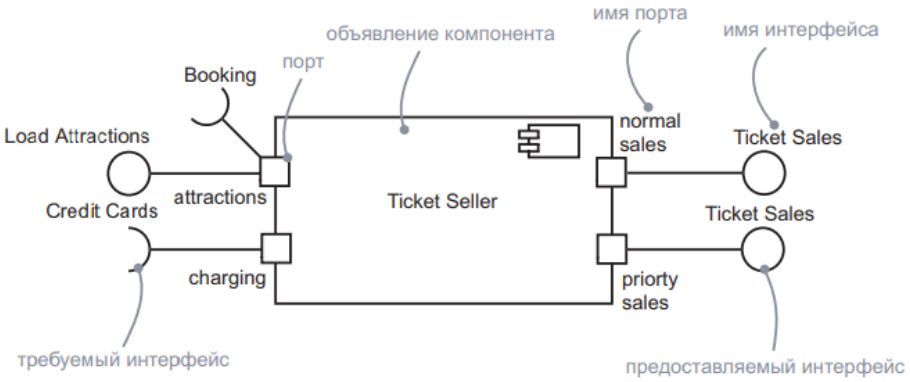

Рис. 2. Порты компонента

Есть два порта для продажи – один для обычных покупателей и один для привилегированных. Оба предоставляют один и тот же интерфейс типа Ticket Sales (Продажа билетов). Порт обслуживания кредитных карт имеет требуемый интерфейс; любой компонент, который его предоставляет, может удовлетворить его. Порт объявлений имеет как предоставляемый, так и требуемый интерфейсы. Используя интерфейс Load Attractions (Информация о развлечениях), театр может передавать афиши и другую информацию о спектаклях в базу данных, используемую для продажи билетов. При помощи интерфейса Booking (Заказ) компонент – продавец билетов может запрашивать у театров сведения о наличии билетов и приобретать их.

6 Внутренняя структура

Компонент может быть реализован как единый фрагмент кода, но в больших системах желательно иметь возможность строить крупные компоненты из малых, которые используются в качестве строительных блоков. Внутренняя структура компонента содержит части, которые вкупе с соединениями между ними составляют его реализацию. Во многих случаях внутренние части могут быть экземплярами более мелких компонентов, связанных статически через порты для обеспечения необходимого поведения, без необходимости для автора модели специфицировать дополнительную логику.

Часть – это единица реализации компонента, которой присвоены имя и тип. В экземпляре компонента содержится по одному или по несколько экземпляров каждой части определенного типа. Часть имеет множественность в пределах компонента. Если эта множественность больше единицы, то в экземпляре компонента может быть ряд экземпляров

компонента данного типа. Если множественность представлена не одним целым числом, то количество экземпляров части может варьироваться в разных экземплярах компонента. Экземпляр компонента создается с минимальным количеством частей (прочие при необходимости добавляются позднее). Атрибут класса – это разновидность части: он имеет тип и множественность, и у каждого экземпляра класса есть один или несколько экземпляров атрибута данного типа.

На рис. 3 показан компонент–компилятор, состоящий из частей четырех видов. В их числе – лексический анализатор, синтаксический анализатор (parser), генератор кода и от одного до трех оптимизаторов. Более полная версия компилятора может быть сконфигурирована с разными уровнями оптимизации; в данной версии необходимый оптимизатор может выбираться во время исполнения.

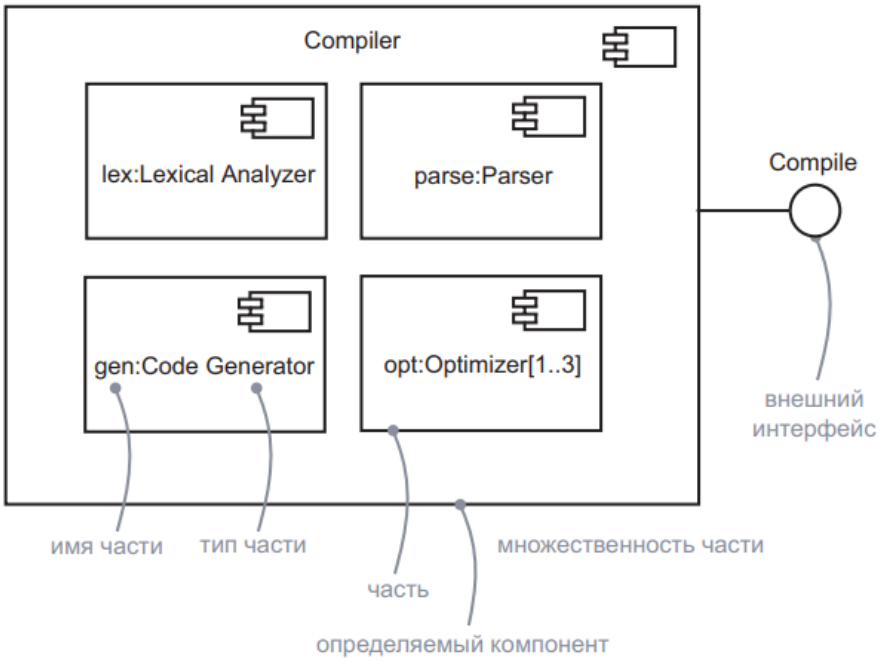

Рис. 3. Части компонента

Отметим, что часть – это не то же самое, что класс. Каждая часть идентифицируется по ее имени так же, как в классе различаются атрибуты. Допустимо наличие нескольких частей одного и того же типа, но вы можете различать их по именам и они предположительно выполняют разные функции внутри компонента. Например, компонент Air Ticket Sales (Продажа авиабилетов) на рис. 4 может включать отдельные части Sales для постоянных и обычных клиентов; обе они работают одинаково, но первая обслуживает только привилегированных клиентов, дает больше шансов избежать очередей и предоставляет некоторые льготы. Поскольку это компоненты одинакового типа, их потребуется различать по именам. Другие два компонента типов SeatAssignment (УказаниеМест) и InventoryManagement (УправлениеСписками) не требуют имен, поскольку присутствуют в одном экземпляре внутри компонента Air Ticket Sales.

Если части представляют собой компоненты с портами, вы можете связать их друг с другом через эти порты. Два порта могут быть подключены друг к другу, если один из них предоставляет данный интерфейс, а другой требует его. Подключение портов подразумевает, что для получения сервиса требующий порт вызовет предоставляющий . Преимущества портов и интерфейсов в том, что кроме них больше ничего не важно; если интерфейсы совместимы, то порты могут быть подключены друг к другу. Инструментальные средства способны автоматически генерировать код вызова одного компонента

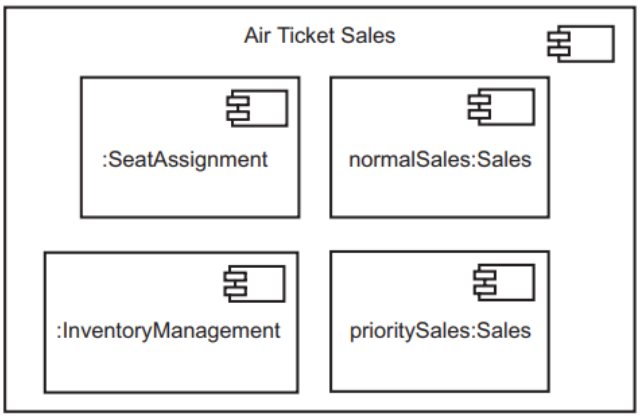

Рис. 4. Части одного типа

от другого. Также их можно подключить к другим компонентам, предоставляющим те же интерфейсы, когда таковые появятся и станут доступны. «Проводок» между двумя портами называется коннектором. В экземпляре охватывающего компонента он представляет просто ссылку (link) или временную ссылку (transient link). Простая ссылка – это экземпляр обычной ассоциации. Временная ссылка представляет связь использования между двумя компонентами. Вместо обычной ассоциации она может быть обеспечена параметром процедуры или локальной переменной, которая служит целью операции. Преимущество портов и интерфейсов в том, что эти два компонента не обязаны ничего «знать» друг о друге на этапе дизайна, до тех пор пока совместимы их интерфейсы.

Коннекторы изображаются двумя способами (рис. 5). Если два компонента явно связаны друг с другом (либо напрямую, либо через порты), достаточно провести линию между ними или их портами. Если же два компонента подключены друг к другу, потому что имеют совместимые интерфейсы, вы можете использовать нотацию «шарик–гнездо», чтобы показать, что между этими компонентами не существует постоянной связи, хотя они и соединены внутри объемлющего компонента. Вы в любое время подставите вместо каждого из них любой другой компонент, если он удовлетворяет интерфейсу.

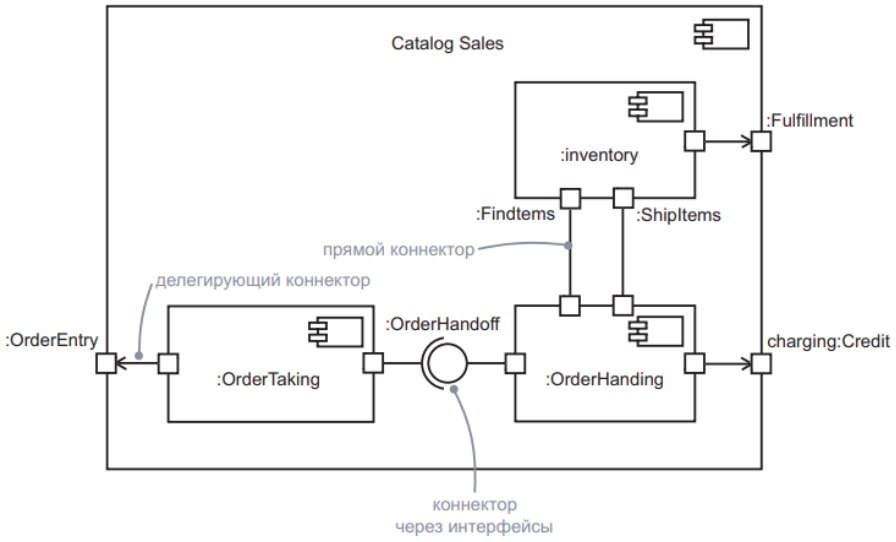

Рис. 5. Коннекторы

Также можно связать внутренние порты с внешними портами объемлющего компонента. В таком случае речь идет о делегирующем коннекторе, поскольку сообщения из внешнего порта делегируются внутреннему. Подобная связь изображается стрелочкой, на-

правленной от внутреннего порта к внешнему. Вы можете представить ее двояко, как вам больше нравится. Во первых, можно считать, что внутренний порт – то же самое, что и внешний; он вынесен на границу и допускает подключение извне. Во вторых, примите во внимание, что любое сообщение, пришедшее на внешний порт, немедленно передается на внутренний, и наоборот. Это неважно – поведение будет одинаковым в любом случае.

В примере на рис. 5 показано использование внутренних портов и разных видов коннекторов. Внешние запросы, приходящие на порт OrderEntry (ПередачаЗаказа), делегируются на внутренний порт подкомпонента OrderTaking (ПриемЗаказа). Этот компонент, в свою очередь, отправляет свой вывод на свой порт OrderHandoff (ПереводЗаказа). Последний подключен по схеме «шарик-гнездо» к подкомпоненту OrderHandling (УправлениеЗаказами). Данный вид подключения предполагает, что у компонентов не существует знаний друг о друге; вывод может быть подсоединен к любому другому компоненту, который соответствует интерфейсу OrderHandOff. Компонент OrderHandling взаимодействует с компонентом Inventory (Опись) для поиска элементов на складе. Это взаимодействие выражено прямым коннектором; поскольку никаких интерфейсов не показано, можно предположить, что данное подключение более плотно, то есть обеспечивает более сильную связь. Как только элемент найден на складе, компонент OrderHandling обращается к внешнему сервису Credit (Кредит) – об этом свидетельствует делегирующий коннектор к внешнему порту changing (изменение).

Как только внешний сервис Credit дает ответ, компонент OrderHandling связывается с другим портом ShipItems (ДеталиДоставки) компонента Inventory, чтобы подготовить пересылку заказа. Компонент Inventory обращается к внешнему сервису Fullfillment (Исполнение) для осуществления доставки.

Отметим, что диаграмма компонентов показывает структуру и возможные способы доставки сообщений компонента. Последовательность сообщений в компоненте она, однако, не отражает. Последовательности и другие виды динамической информации могут быть представлены на диаграмме взаимодействия.

Внутренняя структура, включая порты, части и коннекторы, может быть использована как реализация любых классов – не обязательно компонентов. На самом деле нет большой разницы в семантике между классами и компонентами. Однако зачастую удобно применять соглашение о том, что компоненты используются для инкапсуляции концепций, имеющих внутреннюю структуру (в частности, таких, которые не отображаются непосредственно на отдельный класс в реализации).

7 Типичные приемы моделирования

7.1 Моделирование структурированных классов

Структурированный класс может быть использован для моделирования структур данных, в которых части имеют контекстно-зависимые соединения, существующие только в пределах класса. Обычные атрибуты или ассоциации могут определять составные части класса, но части не могут быть связаны друг с другом на простой диаграмме классов. Класс, внутренняя структура которого показана с помощью частей и коннекторов, позволяет избежать этой проблемы.

Чтобы смоделировать структурированный класс, необходимо:

- Идентифицировать внутренние части класса и их типы.
- Дать каждой части имя, отражающее ее назначение в структурированном классе, а не ее тип.

- Нарисовать коннекторы между частями, которые обеспечивают коммуникацию или имеют контекстно-зависимые связи.
- Не стесняться использовать другие структурированные классы как типы частей, но помнить, что подключение к частям нельзя осуществлять внутри другого структурированного класса – можно подключаться только к их внешним портам.

7.2 Моделирование программного интерфейса API

Если вы разрабатываете систему, состоящую из частейFком понентов, вам часто требуется видеть интерфейсы прикладного программирования (application programming interfaces, API), посредством которых эти части связываются друг с другом. API представляют программные соединения в системе, которые можно смоделировать, используя интерфейсы и компоненты.

По сути, API – это интерфейс, который реализуется одним или несколькими компонентами. Как разработчик, вы на самом деле заботитесь только о самом интерфейсе – какие именно компоненты реализуют операции интерфейса, неважно до тех пор, пока какой-нибудь компонент реализует их. С точки зрения управления конфигурацией системы, однако, эти реализации важны, потому что вам необходимо быть уверенным в том, что когда вы опубликуете API, то будет доступна некая соответствующая ему реализация. К счастью, с помощью UML можно смоделировать оба представления.

Операции, ассоциированные с любым семантически насыщенным API, будут встречаться достаточно часто, потому в большинстве случаев вам не понадобится визуализировать все их сразу. На- против, вы будете стремиться к тому, чтобы оставлять операции на периферии ваших моделей и использовать интерфейсы в качестве дескрипторов, посредством которых можно будет найти все эти наборы операций. Если вы хотите конструировать исполнимые системы на основе таких API, то вам понадобится детализировать модели настолько, чтобы инструменты разработки были способны производить компиляцию в соответствии со свойствами интерфейсов. Наряду с сигнатурой каждой операции вы, возможно, захотите отобразить варианты использования, объясняющие, как нужно применять каждый интерфейс.

Чтобы смоделировать API, необходимо:

- Идентифицировать соединения в системе и смоделировать каждое из них в виде интерфейса, собирая вместе образующие его атрибуты и операции.
- Показать только те свойства интерфейса, которые важны для визуализации в данном контексте; в противном случае скрыть их, сохраняя только в спецификации интерфейса для последующей ссылки (по мере необходимости).
- Смоделировать реализацию каждого API лишь в той мере, в которой она важна для показа конкретной конфигурации.

На рис. 6 представлен API компонента анимации. Вы видите здесь четыре интерфейса, составляющие API: IApplication (IПриложение), IModels (IМодели), IRendering (IВизуализация) и IScripts (ICце нарии). Другие компоненты могут использовать один или несколько таких интерфейсов по необходимости.

8 Советы

Компоненты позволяют вам инкапсулировать части вашей системы, чтобы уменьшить количество зависимостей, сделать их явными, а также повысить взаимозаменяемость и

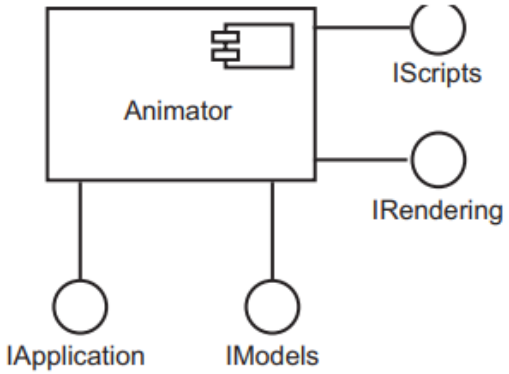

Рис. 6. Моделирование API

гибкость на случай, если система должна будет изменяться в будущем. Хороший компонент наделен следующими характеристиками:

- инкапсулирует сервис с хорошо очерченным интерфейсом и границами;
- имеет внутреннюю структуру, которая допускает возможность ее описания;
- не комбинирует несвязанной функциональности в пределах одной единицы;
- организует внешнее поведение, используя интерфейсы и пор ты в небольшом количестве;
- взаимодействует только через объявленные порты.

Если вы решили показать реализацию компонента, используя вложенные подкомпоненты, примите во внимание следующее:

- не злоупотребляйте использованием подкомпонентов. Если их слишком много, чтобы они легко уместились на одной странице, применяйте дополнительные уровни декомпозиции некоторых из них;
- убедитесь, что подкомпоненты взаимодействуют только через определенные порты и коннекторы;
- определите, какие подкомпоненты непосредственно взаимодействуют с внешним миром, и моделируйте их делегирующими коннекторами.

Когда вы изображаете компонент в UML:

- дайте ему имя, которое ясно описывает его назначение. Таким же образом именуйте интерфейсы;
- присваивайте имена подкомпонентам и портам в случае, если их значение нельзя определить по их типам или же в модели присутствует множество частей одного типа;
- скрывайте ненужные детали. Вы не должны показывать все детали реализации на диаграмме компонентов;
- показывайте динамику компонентов, используя диаграммы взаимодействия.

Список литературы

[1] Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя. 2-е изд.: Пер. с англ. Мухин Н. – М.: ДМК Пресс, 2006. – 496 с.: ил.