Классика программирования

Гради Буч
Джеймс Рамбо
Ивар Якобсон

Введение в UML

от создателей языка

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО

УДК 004.434:004.94UML
ББК 32.973.26-018.1
Б90

Буч Г., Рамбо Д., Якобсон И.
Б90 Введение в UML от создателей языка. 2-е изд.: Пер. с англ. Мухин Н. – М.: ДМК Пресс, 2015. – 496 с.: ил.

ISBN 978-5-97060-157-0

Унифицированный язык моделирования (Unified Modeling Language, UML) является графическим языком для визуализации, специфицирования, конструирования и документирования систем, в которых большая роль принадлежит программному обеспечению. С помощью UML можно разработать детальный план создаваемой системы, содержащий не только ее концептуальные элементы, такие как системные функции и бизнес-процессы, но и конкретные особенности, например классы, написанные на специальных языках программирования, схемы баз данных и программные компоненты многократного использования.

Предлагаемое вашему вниманию руководство пользователя содержит справочный материал, дающий представление о том, как можно использовать UML для решения разнообразных проблем моделирования. В книге подробно, шаг за шагом, описывается процесс разработки программных систем на базе данного языка.

Издание адресовано читателям, которые уже имеют общее представление об объектно-ориентированных концепциях (опыт работы с конкретными объектно-ориентированными языками или методиками не требуется, хотя желателен). В первую очередь руководство предназначено для разработчиков, занятых созданием моделей UML. Тем не менее, книга будет полезна всем, кто осваивает, создает, тестирует или выпускает в свет программные системы.

УДК 004.434:004.94UML
ББК 32.973.26-018.1

ISBN 0-321-26797-4 (англ.)
ISBN 978-5-97060-157-0 (рус.)

Содержание

Глава 2. Введение в UML

В этой главе:

- Обзор UML
- Три шага к пониманию UML
- Архитектура программного обеспечения
- Процесс разработки программного обеспечения

Итак, унифицированный язык моделирования (Unified Modeling Language – UML) – это стандартный инструмент для разработки «чертежей» программного обеспечения. Его можно использовать для визуализации, спецификации, конструирования и документирования артефактов программных систем.

UML подходит для моделирования любых систем – от информационных систем масштаба предприятия до распределенных Web-приложений и даже встроенных систем реального времени. Это очень выразительный язык, предназначенный для представления системы со всех точек зрения, относящихся к ее разработке и внедрению. Несмотря на богатство выразительных средств, UML прост для понимания и применения. Изучение эффективного использования UML начинается с формирования концептуальной модели языка, и в этой связи необходимо ознакомиться с *тремя основными элементами*: базовыми строительными блоками UML, правилами, определяющими, как эти блоки могут сочетаться между собой, а также некоторыми общими механизмами языка.

UML – всего лишь язык, и как таковой представляет только одну из составляющих процесса разработки программного обеспечения. Хотя UML не зависит от моделируемых процессов, лучше всего применять его в тех случаях, когда процесс моделирования основан на применении вариантов использования, сконцентрирован на архитектуре системы, является итеративным и пошаговым.

Обзор UML

UML – это язык для *визуализации*, *специфицирования*, *конструирования* и *документирования* артефактов программных систем.

Язык представляет словарь и правила комбинирования входящих в него слов в целях коммуникации. *Язык моделирования* – это язык, словарь и правила которого сосредоточены на концептуальном и физическом представлении системы. UML – стандартное средство представления «чертежей» программного обеспечения.

Основные принципы моделирования обсуждаются в главе 1.

Моделирование необходимо для понимания системы. При этом ни одна модель не является абсолютно достаточной. Напротив, чтобы понять большинство систем, кроме самых тривиальных, часто требуется множество взаимосвязанных моделей. В отношении программных систем это означает, что необходим язык, средствами которого можно описать архитектуру системы с различных точек зрения, причем на протяжении всего жизненного цикла ее разработки.

Словарь и правила такого языка, как UML, говорят о том, *как* создавать и читать хорошо согласованные модели, но не говорит о том, *какие* именно модели в каких случаях требуется создавать. Это задача всего процесса разработки программного обеспечения. Хорошо организованный процесс должен сам подсказать, какие потребуются рабочие продукты, какие ресурсы понадобятся для их создания и управления ими, как их использовать для оценки выполненной работы и управления проектом в целом.

UML — язык визуализации

С точки зрения многих программистов, промежуток времени между размышлениями о реализации проекта и их изложением в коде стремится к нулю. Вы думаете – значит, вы кодируете. И действительно, некоторые вещи лучше всего выражаются непосредственно в коде на языке программирования, потому что текст программ – самый прямой и короткий путь написания выражений и алгоритмов. Но и в этих случаях программист на самом деле занимается моделированием, хотя и делает это мысленно. Он может даже делать наброски некоторых идей – на доске или салфетке. Однако при этом возникают некоторые проблемы. Во-первых, обсуждение таких концептуальных моделей с другими участниками разработки чревато ошибками и непониманием, если только все участники дискуссии не говорят на одном языке. Как правило, при разработке проектов предприятиям приходится создавать в этих целях свои собственные языки, и вам трудно понять, о чем идет речь, если вы посторонний или новичок в группе. Во-вторых, некоторые вещи, касающиеся программных систем, трудно выразить, пытаясь строить модели лишь средствами текстовых языков программирования. Например, назначение иерархии классов можно понять, внимательно изучив код всех классов в иерархии, но воспринять всю структуру целиком не получится. Аналогично, изучая код, можно исследовать физическое представление

и распределение объектов в Web-ориентированной системе, но нельзя сразу «схватить» его целиком. В-третьих, если разработчик, который писал этот код, никогда не воплощал в нем модели, существовавшие в его голове, то информация о них может быть потеряна навсегда и в лучшем случае частично восстановлена на основе существующей реализации, когда этот разработчик перейдет на другую работу.

Описание моделей на UML позволяет решить третью проблему: явная модель облегчает общение.

Некоторые вещи лучше моделировать в тексте, другие – графически. В действительности во всех интересных системах существуют структуры, которые невозможно выразить на языке программирования. UML – графический язык, позволяющий решить вторую из описанных выше проблем.

UML – нечто большее, чем просто набор графических символов. Каждый из этих символов имеет четко определенную семантику. И это значит, что один разработчик может описать модель на UML, а другой разработчик и даже инструментальное средство – однозначно интерпретировать ее. Это решает первую из упомянутых проблем.

UML — язык специфицирования

В данном контексте *специфицирование* – это построение точных, недвусмысленных и полных моделей. В частности, UML позволяет специфицировать все важные решения, касающиеся анализа, дизайна и реализации, принимаемые в процессе разработки и внедрения программных систем.

UML — язык конструирования

UML не является визуальным языком программирования, но его модели могут быть непосредственно ассоциированы с различными языками программирования. А это значит, что существует возможность отобразить UML-модель на такой язык, как Java, C++ или Visual Basic, а при необходимости даже на таблицы реляционной базы данных либо объекты, хранящиеся в объектно-ориентированной базе данных. Те вещи, которые проще выразить графически, выражаются на UML, а те, что легче выразить в виде текста, – на языке программирования.

Моделирование структуры систем обсуждается в частях II и III.

Отображение модели на язык программирования позволяет осуществить *прямое проектирование* (forward engineering) – генерацию кода на языке программирования из модели UML. Обратное также возможно: вы можете восстановить модель UML на основе существующей реализации. В *обратном проектировании* (reverse engineering) нет никакой магии. Если только вы не закодировали информацию в реализации, она теряется при переходе от модели к коду.

Поэтому обратное проектирование, выполняемое инструментальными средствами, все же требует определенного вмешательства человека. Комбинация этих двух путей – прямого и обратного проектирования – обеспечивает возможность работы как с графическим, так и с текстовым представлениями; при этом обеспечивается согласованность между ними.

В дополнение к прямому отображению UML благодаря своей выразительности и однозначности позволяет непосредственно исполнять модели, имитируя поведение проектируемых систем, а также управляя действующими системами.

UML – язык документирования

Успешные компании, специализирующиеся на программном обеспечении, помимо исполняемого кода производят и другие продукты, включая следующие (но не ограничиваясь ими):

- требования;
- архитектуру;
- проектные решения (дизайн);
- исходный код;
- проектные планы;
- тесты;
- прототипы;
- релизы (версии).

В зависимости от уровня культуры разработки, принятой в компании, некоторые из этих продуктов выражаются более формально, чем другие. Перечисленные продукты – это не только поставляемые составные части проектов; они необходимы для управления, оценки результатов и взаимодействия в процессе разработки системы и после ее внедрения.

UML предназначен для документирования архитектуры системы и всех ее деталей. Кроме того, это язык для выражения требований к системе и описания тестов. И наконец, он подходит для моделирования работ на этапе проектирования и управления версиями.

Где может использоваться UML?

UML прежде всего предназначен для моделирования и разработки программных систем. Наиболее эффективно его применение в следующих областях:

- корпоративные информационные системы;
- банковские и финансовые услуги;
- телекоммуникации;
- транспорт;

- оборона, авиация и космонавтика;
- розничная торговля;
- медицинская электроника;
- наука;
- распределенные Web-сервисы.

Но сфера применения UML не ограничена моделированием программного обеспечения. Его выразительность позволяет вести работу и над непрограммными системами – в частности, продумывать документооборот юридической системы, структуру и функционирование системы здравоохранения, системы управления воздушным движением, а также проектировать аппаратные средства.

Концептуальная модель UML

Чтобы понять UML, вам необходимо сформировать концептуальную модель языка, а это требует изучения трех основных элементов: строительных блоков языка, правил, определяющих их сочетания, и некоторых общих для всего языка механизмов. Лишь усвоив эти идеи, вы сможете читать UML-модели и создавать их самостоятельно. По мере приобретения опыта в использовании UML вы сможете строить концептуальные модели, используя более развитые языковые средства.

Строительные блоки UML

Словарь UML включает три вида строительных блоков:

1. Сущности.
2. Связи.
3. Диаграммы.

Сущности (things) – это абстракции, которые являются основными элементами модели, *связи* (relationships) соединяют их между собой, а диаграммы (diagrams) группируют представляющие интерес наборы сущностей.

Есть четыре вида сущностей UML:

1. Структурные.
2. Поведенческие.
3. Группирующие.
4. Аннотирующие.

Все они представляют собой базовые объектно-ориентированные строительные блоки моделей UML. Вы используете их для описания хорошо согласованных моделей.

Структурные сущности – «имена существительные» в моделях UML. Это в основном статические части модели, представляющие

либо концептуальные, либо физические элементы. В совокупности структурные сущности называются *классификаторами* (classifiers).

Классы обсуждаются в главах 4 и 9.

Класс (class) – это описание набора объектов с одинаковыми атрибутами, операциями, связями и семантикой. Класс реализует один или несколько интерфейсов. Графически класс изображается в виде прямоугольника, обычно включающего имя, атрибуты и операции, как показано на рис. 2.1.

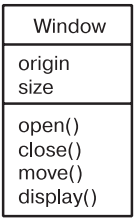

Рис. 2.1. Классы

Интерфейсы обсуждаются в главе 11.

Интерфейс (interface) – это набор операций, который специфицирует сервис (набор услуг) класса или компонента. Таким образом, интерфейс описывает видимое извне поведение элемента. Может представлять полное поведение класса или компонента либо только часть такого поведения. Определяет набор спецификаций операций (то есть их сигнатуру), но никогда не определяет детали их реализации. Объявление интерфейса изображается как класс с ключевым словом «interface» над его именем; атрибуты несущественны, за исключением иногда показываемых констант. Интерфейс, однако, редко существует сам по себе. Интерфейс, представляемый классом для внешнего мира, изображается в виде маленького круга, соединенного линией с рамкой класса. Интерфейс, запрашиваемый классом от некоторого другого класса, представлен маленьким полукругом, соединенным с рамкой класса линией, как показано на рис. 2.2.

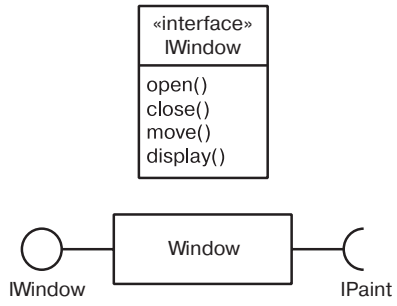

Рис. 2.2. Интерфейсы

Кооперации обсуждаются в главе 28.

Кооперация (collaboration) определяет взаимодействие и представляет собой совокупность ролей и других элементов, которые функционируют вместе, обеспечивая некоторое совместное поведение, представляющее нечто большее, чем сумма поведений отдельных элементов. Кооперации имеют как структурное, так и поведенческое измерения. Конкретный класс или объект может участвовать в нескольких кооперациях. Последние, таким образом, представляют собой реализацию *образцов* (patterns), составляющих систему. Кооперация изображается в виде эллипса, нарисованного пунктирной линией, иногда включающего в себя лишь ее имя, как на рис. 2.3.

Рис. 2.3. Кооперации

Варианты использования обсуждаются в главе 17.

Вариант использования (use case) – это описание последовательности действий, выполняемых системой и приносящих значимый результат конкретному *действующему лицу* (actor). Варианты использования применяются для структурирования поведенческих сущностей модели. Реализуются посредством кооперации. Графически вариант использования представлен эллипсом, нарисованным сплошной линией (обычно он включает в себя только имя, как показано на рис. 2.4).

Рис. 2.4. Варианты использования

Оставшиеся три сущности – активные классы, компоненты и узлы (nodes) – все подобны классам; это говорит о том, что они также описывают наборы сущностей, разделяющих одни и те же атрибуты, операции, связи и семантику. Однако эти три понятия в достаточной степени различаются и необходимы для моделирования определенных аспектов объектно-ориентированных систем, поэтому нуждаются в специальном рассмотрении.

Активные классы обсуждаются в главе 23.

Активный класс – это класс, объекты которого являются владельцами одного или нескольких процессов или потоков (threads) и, таким образом, могут инициировать управляющие воздействия. Активный класс во всем подобен простому классу, за исключением того, что его объекты представляют собой элементы, поведение которых осуществляется параллельно с поведением других

элементов. Изображается как класс с двойными боковыми линиями; обычно включает в себя имя, атрибуты и операции, как показано на рис. 2.5.

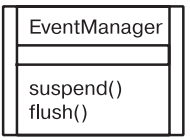

Рис. 2.5. Активные классы

Компоненты и внутренние структуры обсуждаются в главе 15.

Компонент – это модульная часть системы, которая скрывает свою реализацию за набором внешних интерфейсов. Компоненты системы, разделяющие общие интерфейсы, могут замещать друг друга, сохраняя при этом одинаковое логическое поведение. Реализация компонента может быть выражена объединением частей и коннекторов; при этом части могут включать в себя более мелкие компоненты. Графически компонент представлен как класс со специальной пиктограммой в правом верхнем углу (см. рис. 2.6).

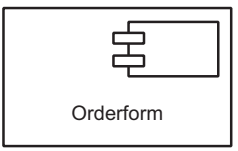

Рис. 2.6. Компоненты

Оставшиеся два элемента – артефакты и узлы – также различаются. Они представляют собой физические сущности, в отличие от предыдущих пяти, относящихся к сущностям логическим или концептуальным).

Артефакты обсуждаются в главе 26.

Артефакт (artifact) – это физическая и замещаемая часть системы, несущая физическую информацию («биты»). В системе вы можете встретить разные виды артефактов, таких как файлы исходного кода, исполняемые программы и скрипты. Обычно артефакт представляет собой физический пакет с исходным или исполняемым кодом. Изображается как прямоугольник, снабженный ключевым словом «artifact», расположенным над его именем (рис. 2.7).

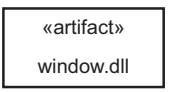

Рис. 2.7: Артефакты

Узлы описываются в главе 27.

Узел (node) – это физический элемент, который существует во время исполнения и представляет вычислительный ресурс, обычно имеющий по меньшей мере некоторую память и часто – вычислительные возможности. Набор компонентов может находиться на узле, а также мигрировать с одного узла на другой. Узел изображается в виде куба, обычно содержащего лишь его имя, как показано на рис. 2.8.

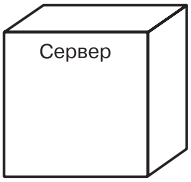

Рис. 2.8. Узлы

Все эти элементы – классы, интерфейсы, кооперации, варианты использования, активные классы, компоненты, артефакты и узлы – являются базовыми структурными сущностями, которые могут быть включены в UML-модель. Существуют также различные вариации: действующие лица (actors), сигналы и утилиты (разновидность классов), процессы и потоки (разновидности активных классов), приложения, документы, файлы, библиотеки, страницы и таблицы (разновидности артефактов).

Варианты использования, которые применяются для структурирования поведенческих сущностей в модели, обсуждаются в главе 17, взаимодействия (interactions) – в главе 16.

Поведенческие сущности – динамические части моделей UML. Это «глаголы» моделей, представляющие поведение во времени и пространстве. Всего существует три основных вида поведенческих сущностей.

Первый из них – *взаимодействие* (interaction) – представляет собой поведение, которое заключается в обмене сообщениями между наборами объектов или ролей в определенном контексте для достижения некоторой цели. Поведение совокупности объектов или индивидуальная операция могут быть выражены взаимодействием. Взаимодействие включает множество других элементов – таких как сообщения, действия (actions) и коннекторы (соединения между объектами). Сообщение изображается в виде линии со стрелкой, почти всегда сопровождаемой именем операции (см. рис. 2.9).

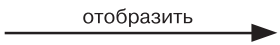

Рис. 2.9. Сообщения

Автоматы обсуждаются в главе 22.

Вторая из поведенческих сущностей – *автомат* (state machine) – представляет собой поведение, характеризуемое последовательностью состояний объекта, в которых он оказывается на протяжении своего

жизненного цикла в ответ на события, вместе с его реакцией на эти события. Поведение индивидуального класса или кооперации классов может быть описано в терминах автомата. Автомат включает в себя множество других элементов: состояния, переходы (из одного состояния в другое), события (сущности, которые инициируют переходы), а также действия (реакции на переходы). Графически состояние представлено прямоугольником с закругленными углами, обычно с указанием имени и подсостояний, если таковые есть (см. рис. 2.10).

Рис. 2.10. Состояния

Третья из поведенческих сущностей – *деятельность* (activity) – специфицирует последовательность шагов процесса вычислений. Во взаимодействии внимание сосредоточено на наборе взаимодействующих объектов, в автомате – на жизненном цикле одного объекта; для деятельности же в центре внимания – последовательность шагов безотносительно к объектам, выполняющим каждый шаг. Отдельный шаг деятельности называется *действием* (action). Изображается оно в виде прямоугольника с закругленными углами, включающего имя, которое отражает его назначение. Состояния и действия различаются по контекстам.

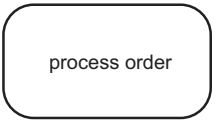

Рис. 2.11. Действия

Эти три элемента – взаимодействия, автоматы и деятельности – представляют собой базовые поведенческие сущности, которые вы можете включить в UML-модель. Семантически эти элементы обычно связаны с различными структурными элементами – в первую очередь, классами, кооперациями и объектами.

Группирующие сущности – организационная часть моделей UML. Это «ящики», по которым можно разложить модель. Главная из группирующих сущностей – пакет.

Пакеты обсуждаются в главе 12.

Пакет (package) – это механизм общего назначения для организации проектных решений, который упорядочивает конструкции реализации. Структурные сущности, поведенческие сущности и даже

другие группирующие сущности могут быть помещены в пакет. В отличие от компонентов (существующих только во время исполнения), пакеты полностью концептуальны, то есть существуют лишь на этапе разработки. Пакет изображается в виде папки с закладкой, обычно только с указанием имени, но иногда и содержимого (см. рис. 2.12).

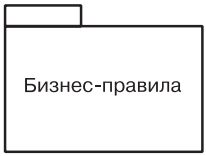

Рис. 2.12. Пакеты

Пакеты – основная группирующая сущность, с помощью которой вы можете организовать UML-модель. Существуют и такие вариации, как каркасы (frameworks), модели, подсистемы (разновидность пакетов).

Аннотирующие сущности – это поясняющие части UML-моделей, иными словами, комментарии, которые вы можете применить для описания, выделения и пояснения любого элемента модели. Главная из аннотирующих сущностей – *примечание* (note). Это простой символ, служащий для описания ограничений и комментариев, относящихся к элементу либо набору элементов. Графически представлен прямоугольником с загнутым углом; внутри помещается текстовый или графический комментарий (рис. 2.13).

Рис. 2.13. Примечания

Этот элемент – базовая аннотирующая сущность, которую вы можете включить в UML-модель. Обычно вы будете использовать его для снабжения диаграмм ограничениями или комментариями, которые лучше всего выражаются в виде формального или неформального текста. Существуют также разные вариации этого элемента, такие как требования, которые специфицируют некоторое желательное поведение с точки зрения, внешней по отношению к модели.

Существует четыре типа **связей в UML**:

1. Зависимость.
2. Ассоциация.
3. Обобщение.
4. Реализация.

Эти связи представляют собой базовые строительные блоки для описания отношений в UML, используемые для разработки хорошо согласованных моделей.

Зависимости обсуждаются в главах 5 и 10.

Первая из них – *зависимость* (dependency) – семантически представляет собой связь между двумя элементами модели, в которой изменение одного элемента (независимого) может привести к изменению семантики другого элемента (зависимого). Графически представлена пунктирной линией, иногда со стрелкой; может быть снабжена меткой (см. рис. 2.14).

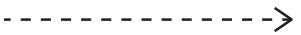

Рис. 2.14. Зависимости

Ассоциации обсуждаются в главах 5 и 10.

Вторая, *ассоциация* (association), – это структурная связь между классами, которая описывает набор связей, существующих между объектами – экземплярами классов. *Агрегация* (aggregation) – особая разновидность ассоциации, представляющая структурную связь целого с его частями. Изображается сплошной линией, иногда со стрелкой; иногда снабжена меткой и часто содержит другие пометки, такие как мощность и конечные имена (см. рис. 2.15).

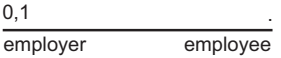

Рис. 2.15. Ассоциации

Обобщения рассматриваются в главах 5 и 10.

Третья связь – *обобщение* (generalization) – выражает специализацию или обобщение, в котором специализированный элемент (потомок) строится по спецификациям обобщенного элемента (родителя). Потомок разделяет структуру и поведение родителя. Графически обобщение представлено в виде сплошной линии с пустой стрелкой, указывающей на родителя (см. рис. 12.16).

Рис. 2.16. Обобщения

Реализации обсуждаются в главе 10.

Четвертая – *реализация* (realization) – это семантическая связь между классификаторами, когда один из них специфицирует соглашение, которого второй обязан придерживаться. Вы встретите связи реализации в двух случаях: между интерфейсами и классами или компонентами, которые реализуют эти интерфейсы, а также между вариантами использования и реализующими их кооперациями. Связь реализации в графическом исполнении – гибрид связей обобщения и зависимости (см. рис. 2.17).