РАЗДЕЛ 6. УНИФИЦИРОВАННЫЙ ЯЗЫК МОДЕЛИРОВАНИЯ UML

Тема: Основные элементы унифицированного языка моделирования UML

Список:

1. Общие сведения о визуальных языках моделирования.
2. Предметы в **UML**.
3. Отношения в **UML**.
4. Диаграммы в **UML**.
5. Механизмы расширения в **UML**.
6. Литература.

1. Общие сведения о визуальных языках моделирования

Для создания моделей анализа и проектирования объектно-ориентированных программных систем используют языки визуального моделирования. Появившись сравнительно недавно, в период с 1989 по 1997 год, эти языки уже имеют представительную историю развития.

В настоящее время различают три поколения языков визуального моделирования. И если первое поколение образовали 10 языков, то численность второго поколения уже превысила 50 языков. Среди наиболее популярных языков 2-го поколения можно выделить: язык Буча (G. Booch), язык Рамбо (J. Rumbaugh), язык Джекобсона (I. Jacobson), язык Коада-Йордона (Coad-Yourdon), язык Шлеера-Меллора (Shlaer-Mellor) и т.д. Каждый язык вводил свои выразительные средства, ориентировался на собственный синтаксис и семантику, иными словами — претендовал на роль единственного и неповторимого языка. В результате разработчики (и пользователи этих языков) перестали понимать друг друга. Возникла острая необходимость *унификации языков*.

Идея унификации привела к появлению языков 3-го поколения. В качестве стандартного языка третьего поколения был принят **Unified Modeling Language** (**UML**), создававшийся в 1994-1997 годах (основные разработчики — три «amigos» Г. Буч, Дж. Рамбо, И. Джекобсон).

UML — стандартный язык для написания моделей анализа, проектирования и реализации объектно-ориентированных программных систем. **UML** может использоваться для визуализации, спецификации, конструирования и документирования результатов программных проектов. **UML** — это не визуальный язык программирования, но его модели прямо транслируются в текст на языках программирования (Java, C++, Visual Basic, Ada 95, Object Pascal) и даже в таблицы для реляционной БД.

Словарь **UML** образуют три разновидности строительных блоков: *предметы*, *отношения*, *диаграммы*.

Предметы — это абстракции, которые являются основными элементами в модели, *отношения* связывают эти предметы, *диаграммы* группируют коллекции предметов.

2. Предметы в UML

В **UML** имеются четыре разновидности предметов:

1. структурные предметы,

2. предметы поведения,
3. группирующие предметы,
4. поясняющие предметы.

Эти предметы являются базовыми объектно-ориентированными строительными блоками. Они используются для написания моделей.

Структурные предметы являются существительными в **UML**-моделях. Они представляют статические части модели — понятийные или физические элементы.

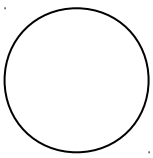
Существует восемь разновидностей структурных предметов:

1. *Класс* — описание множества объектов, которые разделяют одинаковые свойства, операции, отношения и семантику (смысл). Класс реализует один или несколько интерфейсов. Как показано на рис. 1, графически класс отображается в виде прямоугольника, обычно включающего секции с именем, свойствами (атрибутами) и операциями.

Студент
Имя Фамилия Отчество Номер зач. книжки
Сделать лаб. работу() Сдать аттестацию() Сдать зачёт() Сдать экзамен()

Рис. 1 Классы

2. *Интерфейс* — набор операций, которые определяют услуги класса или компонента. Интерфейс описывает поведение элемента, видимое извне. Интерфейс может представлять полные услуги класса или компонента или часть таких услуг. Интерфейс определяет набор спецификаций операций (их сигнатуры), а не набор реализаций операций. Графически интерфейс изображается в виде кружка с именем, как показано на рис. 2. Имя интерфейса обычно начинается с буквы «I». Интерфейс редко показывают самостоятельно. Обычно его присоединяют к классу или компоненту, который реализует интерфейс.



I Учёба

Рис. 2 Интерфейсы

3. *Кооперация* (сотрудничество) определяет взаимодействие и является совокупностью ролей и других элементов, которые работают вместе для обеспечения коллективного поведения более сложного, чем простая сумма всех элементов. Таким образом, кооперации имеют как структурное, так и поведенческое измерения. Конкретный класс может участвовать в нескольких кооперациях. Эти кооперации представляют реализацию паттернов (образцов), которые формируют систему. Как показано на рис. 3,

графически кооперация изображается как пунктирный эллипс, в который вписывается ее имя.

Рис 3. Кооперация

4. *Актер* — набор согласованных ролей, которые могут играть пользователи при взаимодействии с системой (ее элементами *Use Case*). Каждая роль требует от системы определенного поведения. Как показано на рис. 4, актер изображается как проволочный человечек с именем.

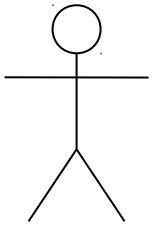


Рис. 4 Актёры

5. *Элемент Use Case* (Прецедент) — описание последовательности действий (или нескольких последовательностей), выполняемых системой в интересах отдельного актера и производящих видимый для актера результат. В модели элемент *Use Case* применяется для структурирования предметов поведения. Элемент *Use Case* реализуется кооперацией. Как показано на рис. 5, элемент *Use Case* изображается как эллипс, в который вписывается его имя.

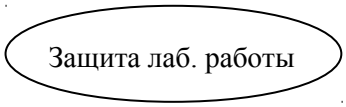


Рис. 5 Элементы Use Case

6. *Активный класс* — класс, чьи объекты имеют один или несколько процессов (или потоков) и поэтому могут инициировать управляющую деятельность. Активный класс похож на обычный класс за исключением того, что его объекты действуют одновременно с объектами других классов. Как показано на рис. 6, активный класс изображается как утолщенный прямоугольник, обычно включающий имя, свойства (атрибуты) и операции.

Преподаватель
Имя Фамилия Отчество
Начать приём зачёта() Начать приём экзамена()

Рис. 6 Активные классы

7. *Компонент* — физическая и заменяемая часть системы, которая соответствует набору интерфейсов и обеспечивает реализацию этого набора интерфейсов. В систему включаются как компоненты, являющиеся результатами процесса разработки (файлы исходного кода), так и различные разновидности используемых компонентов (COM+-компоненты, Java Beans). Обычно компонент — это физическая упаковка различных логических элементов (классов, интерфейсов и сотрудничеств). Как показано на рис. 7, компонент изображается как прямоугольник с вкладками, обычно включающий имя.

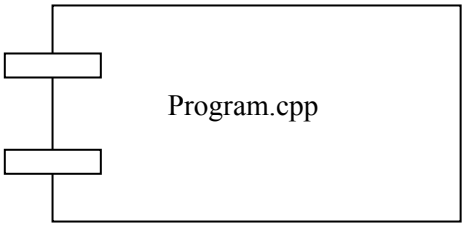


Рис. 7 Компоненты

8. *Узел* — физический элемент, который существует в период работы системы и представляет ресурс, обычно имеющий память и возможности обработки. В узле размещается набор компонентов, который может перемещаться от узла к узлу. Как показано на рис. 8, узел изображается как куб с именем.

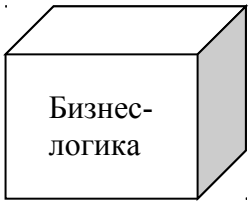


Рис. 8 Узлы

Предметы поведения — динамические части **UML**-моделей. Они являются глаголами моделей, представлением поведения во времени и пространстве. Существует две основные разновидности предметов поведения.

1. *Взаимодействие* — поведение, заключающее в себе набор сообщений, которыми обменивается набор объектов в конкретном контексте для достижения определенной цели. Взаимодействие может определять динамику как совокупности объектов, так и отдельной операции. Элементами взаимодействия являются сообщения, последовательность действий (поведение, вызываемое сообщением) и связи (соединения между объектами). Как показано на рис. 9, сообщение изображается в виде направленной линии с именем ее операции.

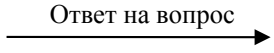


Рис. 9 Сообщения

2. *Конечный автомат* — поведение, которое определяет последовательность состояний объекта или взаимодействия, выполняемые в ходе его существования в ответ на события (и с учетом обязанностей по этим событиям). С помощью конечного автомата может определяться поведение индивидуального класса или кооперации классов. Элементами конечного автомата являются состояния, переходы (от состояния к

состоянию), события (предметы, вызывающие переходы) и действия (реакции на переход). Как показано на рис. 10, состояние изображается как закругленный прямоугольник, обычно включающий его имя и его подсостояния (если они есть).

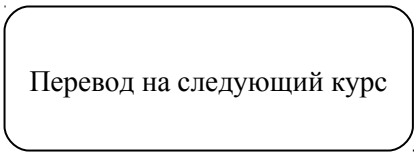


Рис. 10 Состояния

Эти два элемента — *взаимодействия* и *конечные автоматы* — являются базисными предметами поведения, которые могут включаться в **UML**-модели. Семантически эти элементы ассоциируются с различными структурными элементами (прежде всего с классами, сотрудничествами и объектами).

Группирующие предметы — организационные части **UML**-моделей. Это ящики, по которым может быть разложена модель. Предусмотрена одна разновидность группирующего предмета — *пакет*.

Пакет — общий механизм для распределения элементов по группам. В пакет могут помещаться структурные предметы, предметы поведения и даже другие группировки предметов. В отличие от компонента (который существует в период выполнения), пакет — чисто концептуальное понятие. Это означает, что пакет существует только в период разработки. Как показано на рис. 11, пакет изображается как папка с закладкой, на которой обозначено его имя и, иногда, его содержание.

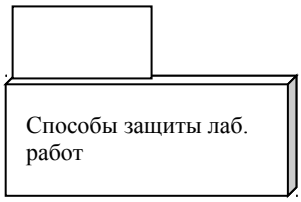


Рис. 11 Пакеты

Поясняющие предметы — разъясняющие части **UML**-моделей. Они являются замечаниями, которые можно применить для описания, объяснения и комментирования любого элемента модели. Предусмотрена одна разновидность поясняющего предмета — *примечание*.

Примечание — символ для отображения ограничений и замечаний, присоединяемых к элементу или совокупности элементов. Как показано на рис. 12, примечание изображается в виде прямоугольника с загнутым углом, в который вписывается текстовый или графический комментарий.

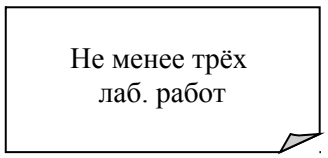


Рис. 12 Примечания

3. Отношения в UML

В UML имеются четыре разновидности отношений:

1. зависимость,
2. ассоциация,
3. обобщение,
4. реализация.

Эти отношения являются базовыми строительными блоками отношений. Они используются при написании моделей.

1. *Зависимость* — семантическое отношение между двумя предметами, в котором изменение в одном предмете (независимом предмете) может влиять на семантику другого предмета (зависимого предмета). Как показано на рис. 13, зависимость изображается в виде пунктирной линии, возможно направленной на независимый предмет и иногда имеющей метку.

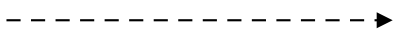


Рис. 13 Зависимости

2. *Ассоциация* — структурное отношение, которое описывает набор связей, являющихся соединением между объектами. *Агрегация* — это специальная разновидность ассоциации, представляющая структурное отношение между целым и его частями. Как показано на рис. 14, ассоциация изображается в виде сплошной линии, возможно направленной, иногда имеющей метку и часто включающей другие «украшения», такие как мощность и имена ролей.

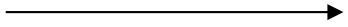


Рис. 14 Ассоциации

3. *Обобщение* — отношение специализации/обобщения, в котором объекты специализированного элемента (потомка, ребенка) могут заменять объекты обобщенного элемента (предка, родителя). Иначе говоря, потомок разделяет структуру и поведение родителя. Как показано на рис. 15, обобщение изображается в виде сплошной стрелки с полым наконечником, указывающим на родителя.

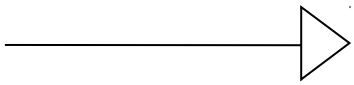


Рис. 15 Обобщения

4. *Реализация* — семантическое отношение между классификаторами, где один классификатор определяет контракт, который другой классификатор обязуется выполнять (к классификаторам относят классы, интерфейсы, компоненты, элементы *Use Case*, кооперации). Отношения реализации применяют в двух случаях: между интерфейсами и классами (или компонентами), реализующими их; между элементами *Use Case* и кооперациями, которые реализуют их. Как показано на рис. 16, реализация изображается как нечто среднее между обобщением и зависимостью.

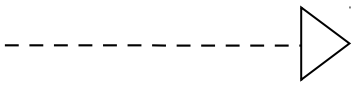


Рис. 16 Реализации

4. Диаграммы в UML

Диаграмма — графическое представление множества элементов, наиболее часто изображается как связный граф из вершин (предметов) и дуг (отношений). Диаграммы рисуются для визуализации системы с разных точек зрения, затем они отображаются в систему. Обычно диаграмма дает неполное представление элементов, которые составляют систему. Хотя один и тот же элемент может появляться во всех диаграммах, на практике он появляется только в некоторых диаграммах. Теоретически диаграмма может содержать любую комбинацию предметов и отношений, на практике ограничиваются малым количеством комбинаций, которые соответствуют пяти представлениям архитектуры ПС. По этой причине **UML** включает девять видов диаграмм:

1. диаграммы классов,
2. диаграммы объектов,
3. диаграммы Use Case (диаграммы прецедентов),
4. диаграммы последовательности,
5. диаграммы сотрудничества (кооперации),
6. диаграммы схем состояний,
7. диаграммы деятельности,
8. компонентные диаграммы,
9. диаграммы размещения (развертывания).

1. *Диаграмма классов* показывает набор классов, интерфейсов, сотрудничеств и их отношений. При моделировании объектно-ориентированных систем диаграммы классов используются наиболее часто. Диаграммы классов обеспечивают статическое проектное представление системы. Диаграммы классов, включающие активные классы, обеспечивают статическое представление процессов системы.

2. *Диаграмма объектов* показывает набор объектов и их отношения. Диаграмма объектов представляет статический «моментальный снимок» с экземпляров предметов, которые находятся в диаграммах классов. Как и диаграммы классов, эти диаграммы обеспечивают статическое проектное представление или статическое представление процессов системы (но с точки зрения реальных или фототипичных случаев).

3. *Диаграмма Use Case* (диаграмма прецедентов) показывает набор элементов Use Case, актеров и их отношений. С помощью диаграмм Use Case для системы создается статическое представление Use Case. Эти диаграммы особенно важны при организации и моделировании поведения системы, задании требований заказчика к системе.

4. *Диаграммы последовательности* и *диаграммы сотрудничества* — это разновидности диаграмм взаимодействия.

5. *Диаграмма взаимодействия* показывает взаимодействие, включающее набор объектов и их отношений, а также пересылаемые между объектами сообщения. Диаграммы взаимодействия обеспечивают динамическое представление системы.

6. *Диаграмма последовательности* — это диаграмма взаимодействия, которая выделяет упорядочение сообщений по времени.

7. *Диаграмма сотрудничества* (диаграмма кооперации) — это диаграмма взаимодействия, которая выделяет структурную организацию объектов, посылающих и принимающих сообщения. Диаграммы последовательности и диаграммы сотрудничества изоморфны, что означает, что одну диаграмму можно трансформировать в другую диаграмму.

8. *Диаграмма схем состояний* показывает конечный автомат, представляет состояния, переходы, события и действия. Диаграммы схем состояний обеспечивают динамическое представление системы. Они особенно важны при моделировании поведения интерфейса, класса или сотрудничества. Эти диаграммы выделяют такое поведение объекта, которое управляется событиями, что особенно полезно при моделировании реактивных систем.

9. *Диаграмма деятельности* — специальная разновидность диаграммы схем состояний, которая показывает поток от действия к действию внутри системы. Диаграммы деятельности обеспечивают динамическое представление системы. Они особенно важны при моделировании функциональности системы и выделяют поток управления между объектами.

10. *Компонентная диаграмма* показывает организацию набора компонентов и зависимости между компонентами. Компонентные диаграммы обеспечивают статическое представление реализации системы. Они связаны с диаграммами классов в том смысле, что в компонент обычно отображается один или несколько классов, интерфейсов или кооперации.

11. *Диаграмма размещения* (диаграмма развертывания) показывает конфигурацию обрабатывающих узлов периода выполнения, а также компоненты, живущие в них. Диаграммы размещения обеспечивают статическое представление размещения системы. Они связаны с компонентными диаграммами в том смысле, что узел обычно включает один или несколько компонентов.

5. Механизмы расширения в UML

UML — развитый язык, имеющий большие возможности, но даже он не может отразить все нюансы, которые могут возникнуть при создании различных моделей. Поэтому **UML** создавался как открытый язык, допускающий контролируемые расширения. Механизмами расширения в **UML** являются:

1. ограничения,
2. теговые величины,
3. стереотипы.

1. *Ограничение* (constraint) расширяет семантику строительного **UML**-блока, позволяя добавить новые правила или модифицировать существующие. Ограничение показывают как текстовую строку, заключенную в фигурные скобки {}. Например, на рис. 17 введено простое ограничение на свойство сумма класса Сессия Банкомата — его значение должно быть кратно 20. Кроме того, здесь показано ограничение на два элемента (две ассоциации), оно располагается возле пунктирной линии, соединяющей элементы, и имеет следующий смысл — владельцем конкретного счета не может быть и организация, и персона.

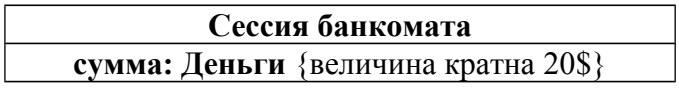


Рис. 17 Ограничения

2. *Теговая величина* (tagged value) расширяет характеристики строительного **UML**-блока, позволяя создать новую информацию в спецификации конкретного элемента. Теговую величину показывают как строку в фигурных скобках *{}*. Строка имеет вид имя теговой *величины* = *значение*. Иногда (в случае предопределенных тегов) указывается только имя теговой величины. Отметим, что при работе с продуктом, имеющим много реализаций, полезно отслеживать версию и автора определенных блоков. Версия и автор не принадлежат к основным понятиям UML. Они могут быть добавлены к любому строительному блоку (например, к классу) введением в блок новых теговых величин. Например, на рис. 18 класс *ТекстовыйПроцессор* расширен путем явного указания его версии и автора.

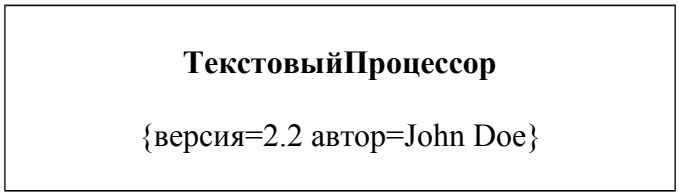


Рис. 17 Теги

3. *Стереотип* (stereotype) расширяет словарь языка, позволяет создавать новые виды строительных блоков, производные от существующих и учитывающие специфику новой проблемы. Элемент со стереотипом является вариацией существующего элемента, имеющей такую же форму, но отличающуюся по сути. У него могут быть дополнительные ограничения и теговые величины, а также другое визуальное представление. Он иначе обрабатывается при генерации программного кода. Отображают стереотип как имя, указываемое в двойных угловых скобках (или в угловых кавычках). Примеры элементов со стереотипами приведены на рис. 19. Стереотип «exception» говорит о том, что класс ПотеряЗначимости теперь рассматривается как специальный класс, которому, положим, разрешается только генерация и обработка сигналов исключений. Особые возможности метакласса получил класс ЭлементМодели. Кроме того, здесь показано применение стереотипа *«call»* к отношению зависимости (у него появился новый смысл).

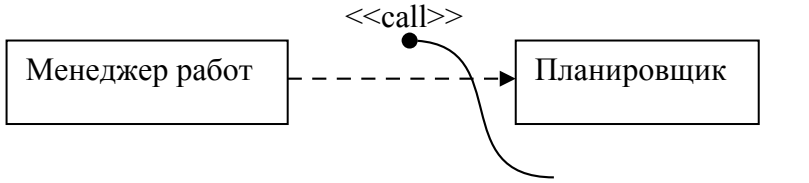


Рис. 18 Стереотипы

Таким образом, механизмы расширения позволяют адаптировать **UML** под нужды конкретных проектов и под новые программные технологии. Возможно добавление новых строительных блоков, модификация спецификаций существующих блоков и даже изменение их семантики. Конечно, очень важно обеспечить контролируемое введение расширений.

6. Литература

1. Технологии разработки программного обеспечения: Учебник/ С. Орлов. — СПб.: Питер, 2002. — 464 с.: ил.