Лекция №6

ФОРМАЛИЗАЦИЯ И АЛГОРИТМИЗАЦИЯ ПРОЦЕССОВ. ЯЗЫКИ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

Имитационное моделирование начинается с изучения моделируемой системы и описания ее в виде логических схем и функциональных взаимосвязей. Однако в конечном счете встает задача описания модели на том языке, который понятен компьютеру. Рабочий язык цифровых устройств — двоичный код представления данных или код, кратный 2, например восьмеричный или шестнадцатеричный. Для общения пользователей с машиной такие языки очень неудобны, поэтому стали развиваться специальные языки программирования. К сожалению, за последние годы было создано столько универсальных и специализированных языков, что почти невозможно определить, какой из них оптимален для данного конкретного приложения. В 1972 г. только в США применялось более 170 языков. Чаще всего язык, которым стремится воспользоваться исследователь, выбирается не потому, что он является самым неподходящим, а лишь потому, что этот язык исследователь знает лучше других. Модель можно описать на любом универсальном алгоритмическом языке, однако применение специализированных языков имитационного моделирования может дать значительные преимущества в смысле легкости, производительности и эффективности.

Многократно повторяя эти этапы, аналитики пришли к выводу, что многие ситуации, описываемые имитационными моделями, можно охарактеризовать как системы с материальными потоками. Многие программы описывали процессы, функционально сходные друг с другом, что привело несколько к идее создания специализированных языков. В своем развитии эти языки постепенно прошли через ряд стадий: от языков типа ассемблера с некоторыми специальными свойствами к более широким проблемно-ориентированным языкам, получившим значительное распространение на коммерческом рынке пользователей, и, наконец к сложным специализированным языкам имитационного моделирования. Для целей имитации подходит любой алгоритмический язык программирования, но языки, специально созданные для имитации на ЭВМ, обладают рядом особых преимуществ:

1) снижают трудоемкость написания программы;
2) обеспечивают более строгое следование выбранной концепции;
3) помогают четко классифицировать элементы системы;|
4) обеспечивают гибкость, необходимую для изменения программы;
5) обеспечивают возможность различать элементы одного класса по их характеристикам или свойствам;
6) описывают взаимосвязи между элементами и внешней по отношению к ним средой;
7) позволяют корректировать число элементов модели в соответствии с изменением внутренних условий системы.

Эмшоф и Сиссон полагают, что в любом имитационном исследовании требуется язык имитационного моделирования со специфическими свойствами, отличающийся от универсального алгебраического или специализированного языка программирования. Среди этих дополнительных требований нужно отметить следующие:

— способность генерировать случайные числа;

— возможность генерировать случайные переменные;

— возможность «продвигать» время либо на одну единицу, либо до следующего события;

— способность накапливать выходные данные;

— способность проводить статистический анализ накапливаемых данных;

— способность распределять выходные данные по заранее заданным форматам;

— возможность выявлять и регистрировать логические несоответствия и другие ситуации, связанные с ошибками.

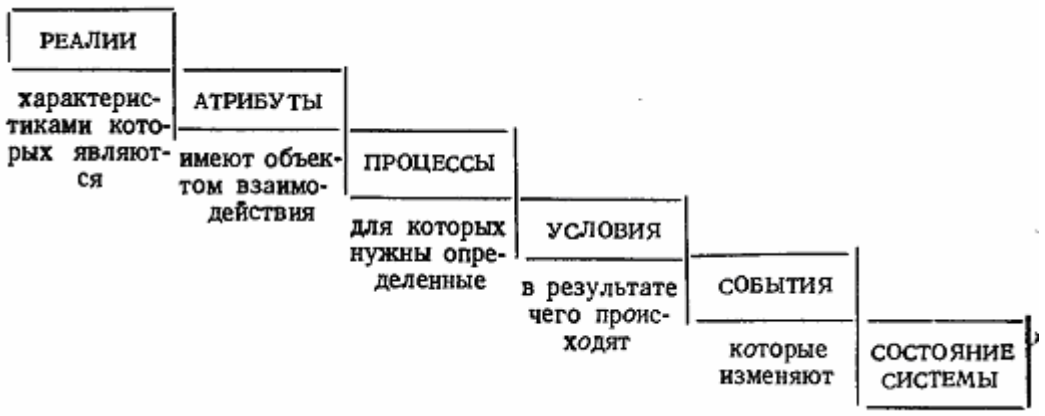
Преимущества и недостатки различных языков программирования

Универсальные языки	
Преимущества	*Недостатки*
1. Минимум ограничений на выходной формат	1. Значительное время, затрачиваемое на программирование
2. Широкая распространенность	2. Значительное время, затрачиваемое на отладку программ
Специализированные языки	
1. Меньшие затраты времени на программирование	1. Необходимость твердо придерживаться ограничений на выходные форматы
2. Более эффективные методы выявления ошибок	2. Меньшая гибкость моделей, большие затраты машинного времени на моделирование
3. Краткость, точность выражения понятий, характеризующих имитируемые процессы	
4. Возможность заранее строить для пользователей стандартные подпрограммы, которые могут применяться в любой имитационной модели	
5. Автоматическое формирование определенных типов данных, необходимых в процессе имитационного моделирования	
6. Удобство накопления и представления выходных данных	
7. Обеспечение управления и контроля над распределением машинной памяти в процессе имитационного моделирования	

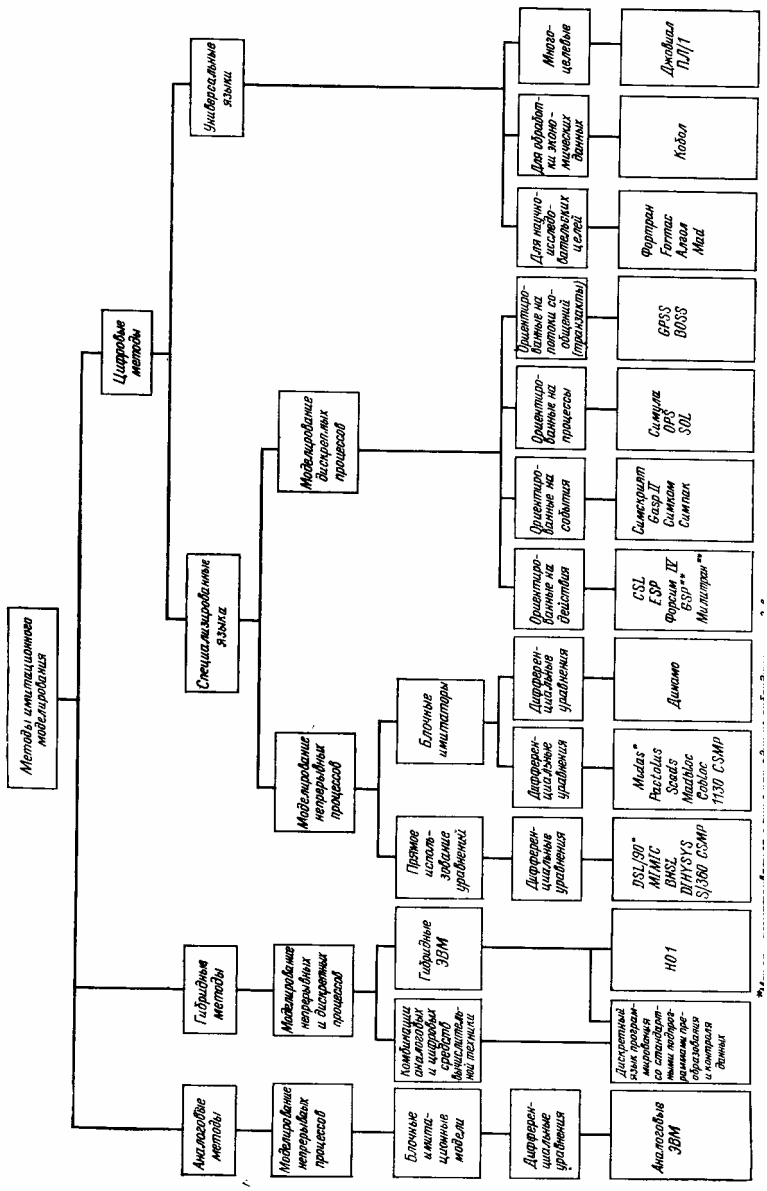
Под «мировоззрением» применительно к языкам имитационного моделирования подразумевается заложенный разработчиком используемого языка принцип описания систем, подлежащих моделированию. Все языки, которые применяются в имитационном моделировании, имеют определенное мировоззрение, и его надо учитывать при создании модели. Мировоззрение типичного языка имитации дискретных процессов можно было бы представить следующим образом:

1. Реальный мир отображается в языке в виде множества некоторых реалий, которые можно модифицировать или списывать с помощью характеристик, называемых атрибутами.
2. Реалии взаимодействуют с конкретными процессами реального мира, которые соответствуют конкретным условиям, определяющим последовательность этих взаимодействий.
3. Взаимодействия рассматриваются как события, происходящие внутри системы и влекущие за собой изменение состояния системы.

«Мировоззрение» типичного языка имитационного моделирования можно представить в виде такой схемы:



Классификация языков



CASE-средства. Общая характеристика и классификация

Современные CASE-средства (Computer Aided Software Engineering) охватывают обширную область поддержки многочисленных технологий проектирования ИС: от простых средств анализа и документирования до полномасштабных средств автоматизации, покрывающих весь жизненный цикл ПО.

Наиболее трудоемкими этапами разработки ИС являются этапы анализа и проектирования, в процессе которых CASE-средства обеспечивают качество принимаемых технических решений и подготовку проектной документации. При этом большую роль играют методы визуального представления информации. Это предполагает построение структурных или иных диаграмм в реальном масштабе времени, использование многообразной цветовой палитры, сквозную проверку синтаксических правил. Графические средства моделирования предметной области позволяют разработчикам в наглядном виде изучать существующую ИС, перестраивать ее в соответствии с поставленными целями и имеющимися ограничениями.

В разряд CASE-средств попадают как относительно дешевые системы для персональных компьютеров с весьма ограниченными возможностями, так и дорогостоящие системы для неоднородных вычислительных платформ и операционных сред. Так, современный рынок программных средств насчитывает около 300 различных CASE-средств, наиболее мощные из которых так или иначе используются практически всеми ведущими западными фирмами.

Обычно к CASE-средствам относят любое программное средство, автоматизирующее ту или иную совокупность процессов жизненного цикла ПО и обладающее следующими основными характерными особенностями:

- мощные графические средства для описания и документирования ИС, обеспечивающие удобный интерфейс с разработчиком и развивающие его творческие возможности;
- интеграция отдельных компонент CASE-средств, обеспечивающая управляемость процессом разработки ИС;
- использование специальным образом организованного хранилища проектных метаданных (репозитория).

Интегрированное CASE-средство (или комплекс средств, поддерживающих полный ЖЦ ПО) содержит следующие компоненты;

- репозиторий, являющийся основой CASE-средства. Он должен обеспечивать хранение версий проекта и его отдельных компонентов, синхронизацию поступления информации от различных разработчиков при групповой разработке, контроль метаданных на полноту и непротиворечивость;
- графические средства анализа и проектирования, обеспечивающие создание и редактирование иерархически связанных диаграмм (DFD, ERD и др.), образующих модели ИС;
- средства разработки приложений, включая языки 4GL и генераторы кодов;
- средства конфигурационного управления;
- средства документирования;
- средства тестирования;
- средства управления проектом;
- средства реинжиниринга.

Все современные CASE-средства могут быть классифицированы в основном по типам и категориям. Классификация по типам отражает функциональную ориентацию CASE-средств на те или иные процессы ЖЦ. Классификация по категориям определяет степень интегрированности по выполняемым функциям и включает отдельные локальные средства, решающие небольшие автономные задачи (tools), набор частично интегрированных средств, охватывающих большинство этапов жизненного цикла ИС

(toolkit) и полностью интегрированные средства, поддерживающие весь ЖЦ ИС и связанные общим репозиторием. Помимо этого, CASE-средства можно классифицировать по следующим признакам:

- применяемым методологиям и моделям систем и БД;
- степени интегрированности с СУБД;
- доступным платформам.

Классификация по типам в основном совпадает с компонентным составом CASE-средств и включает следующие основные типы:

- средства анализа (Upper CASE), предназначенные для построения и анализа моделей предметной области (Design/IDEF (Meta Software), BPwin (Logic Works));
- средства анализа и проектирования (Middle CASE), поддерживающие наиболее распространенные методологии проектирования и использующиеся для создания проектных спецификаций (Vantage Team Builder (Cayenne), Designer/2000 (ORACLE), Silverrun (CSA), PRO-IV (McDonnell Douglas), CASE.Аналитик (МакроПроджект)). Выходом таких средств являются спецификации компонентов и интерфейсов системы, архитектуры системы, алгоритмов и структур данных;
- средства проектирования баз данных, обеспечивающие моделирование данных и генерацию схем баз данных (как правило, на языке SQL) для наиболее распространенных СУБД. К ним относятся ERwin (Logic Works), S-Designor (SDP) и DataBase Designer (ORACLE). Средства проектирования баз данных имеются также в составе CASE-средств Vantage Team Builder, Designer/2000, Silverrun и PRO-IV;
- средства разработки приложений. К ним относятся средства 4GL (Uniface (Compuware), JAM (JYACC), PowerBuilder (Sybase), Developer/2000 (ORACLE), New Era (Informix), SQL Windows (Gupta), Delphi (Borland) и др.) и генераторы кодов, входящие в состав Vantage Team Builder, PRO-IV и частично - в Silverrun;
- средства реинжиниринга, обеспечивающие анализ программных кодов и схем баз данных и формирование на их основе различных моделей и проектных спецификаций. Средства анализа схем БД и формирования ERD входят в состав Vantage Team Builder, PRO-IV, Silverrun, Designer/2000, ERwin и S-Designor. В области анализа программных кодов наибольшее распространение получают объектно-ориентированные CASE-средства, обеспечивающие реинжиниринг программ на языке C++ (Rational Rose (Rational Software), Object Team (Cayenne)).

Вспомогательные типы включают:

- средства планирования и управления проектом (SE Companion, Microsoft Project и др.);
- средства конфигурационного управления (PVCS (Intersolv));
- средства тестирования (Quality Works (Segue Software));
- средства документирования (SoDA (Rational Software)).

На сегодняшний день Российский рынок программного обеспечения располагает следующими наиболее развитыми CASE-средствами:

- Vantage Team Builder (Westmount I-CASE);
- Designer/2000;
- Silverrun;
- ERwin+BPwin;
- S-Designor;
- CASE.Аналитик.

Унифицированный язык моделирования (UML)

В работах, описывающих средства разработки программных проектов, то и дело мелькает сокращение UML, которое означает *Unified Modeling Language* - Унифицированный Язык Моделирования. Естественно возникает вопрос: что это за новый язык, и нужно ли с ним знакомиться? В статье делается попытка ответить на этот вопрос. Краткий ответ прост - знакомиться с UML в той или иной степени придется, поскольку UML – это стандартная нотация визуального моделирования программных систем, принятая консорциумом Object Managing Group (OMG) осенью 1997 г., и на сегодняшний день она поддерживается многими объектно-ориентированным CASE продуктами, включая Rational Rose.

Визуальное моделирование

Итак, что же такое UML, и почему этому языку моделирования уделяется в последнее время столь большое внимание? Нужно ли его изучать? Как его использовать при разработке программных проектов?

Дело в том, что в последнее время наблюдается общее повышение интереса ко всем аспектам, связанным с разработкой сложных программных приложений. Для многих компаний корпоративное программное обеспечения и базы данных (БД) представляют *стратегическую ценность*. Существует высокая заинтересованность в разработке и верификации методов и подходов, позволяющих автоматизировать создание сложных программных информационных систем (ИС). Известно, что систематическое использование таких методов позволяет значительно улучшить качество, сократить стоимость и время поставки ИС. В настоящее время эти методы включают в себя:

- компонентную технологию разработки моделей ИС,
- визуальное программирование (RAD средства),
- использование образцов (patterns) при проектировании ИС,
- визуальное представление различных аспектов проекта (визуальное моделирование, CASE - средства)

Визуальные модели широко используются в существующих технологиях управления проектированием систем, сложность, масштабы и функциональность которых постоянно возрастают. В практике эксплуатации ИС постоянно приходится решать такие задачи как: физическое перераспределение вычислений и данных, обеспечение параллелизма вычислений, репликация БД, обеспечение безопасности доступа к ИС, оптимизация балансировки нагрузки ИС, устойчивость к сбоям и т.п.

Построение *модели* корпоративной ИС до ее программной разработки или до начала проведения архитектурной реконструкции столь же необходимо, как наличие проектных чертежей перед строительством большого здания. Хорошие модели ИС позволяют наладить *плодотворное взаимодействие* между заказчиками, пользователями и командой разработчиков. Визуальные модели обеспечивают ясность представления выбранных архитектурных решений и позволяют понять разрабатываемую систему во всей ее полноте. Сложность разрабатываемых систем продолжает увеличиваться, и поэтому возрастает актуальность использования «хороших» методов моделирования ИС. Язык моделирования, как правило, включает в себя:

- элементы модели - фундаментальные концепции моделирования и их семантику;
- нотацию - визуальное предоставление элементов моделирования;
- принципы использования - правила применения элементов в рамках построения тех или иных типов моделей ИС.

Построение визуальных моделей позволяет решить сразу несколько типичных проблем. Во-первых, и это главное, технология визуального моделирования, позволяет работать со сложными и очень сложными системами и проектами. И не важно, преобладает ли в проекте «техническая сложность» (статическая) или «динамическая

сложность управления». Сложность программных систем возрастает по мере создания новых версий. И в какой-то момент наступает «эффект критической массы», когда дальнейшее развитие ИС становиться невозможным, поскольку уже никто не представляет в целом «что и почему происходит». Происходит *потеря управлением проектом*. Внешней причиной или толчком возникновения этого неприятного эффекта может послужить, например, увольнение ведущего программиста или системного аналитика.

Во-вторых, визуальные модели позволяют содержательно организовать общение между заказчиками и разработчиками. Шутка о том, что «заказчик что-то хочет, но точно не знает, чего именно», с завидным постоянством часто оказывается былью. А если на начальном этапе работы над проектом ИС заказчик думает, что точно знает, что хочет, то, как правило, и об этом свидетельствует богатый опыт, его требования изменяются («плывут») в ходе выполнения проекта. С одной стороны, аппетит приходит во время еды, а с другой, высокая динамика бизнеса объективно заставляет менять требования к разрабатываемой (или поддерживаемой) ИС.

Визуальное моделирование не способно раз и навсегда решить все проблемы, однако его использование существенно облегчает достижения таких целей как:

- повышение качества программного продукта,
- сокращение стоимости проекта,
- поставка системы в запланированные сроки.

Как разбивать сложную систему на части

При проектировании сложной ИС ее разбивают на части, каждая из которых затем рассматривается отдельно. Возможны два различных способа такого разбиения ИС на подсистемы: структурное (или функциональное) разбиение и объектная (компонентная) декомпозиция.

Суть функционального разбиения хорошо отражена в известной формуле:

“Программа=Данные + Алгоритмы”.

При ***функциональной декомпозиции*** программной системы ее структура может быть описана блок-схемами, узлы которых представляют собой “обрабатывающие центры” (функции), а связи между узлами описывают движение данных.

Объектное разбиение в последнее время называют ***компонентным,*** что нашло отражение в специальном термине: «разработка, основанная на компонентах» (Component Based Development - CBD). При этом используется иной принцип декомпозиции - система разбивается на “активные сущности” – объекты или компоненты, которые взаимодействуют друг с другом, ***обмениваясь сообщениями*** и, выступая друг к другу в отношении “клиент/сервер”. Сообщения, которые может принимать объект, определены в его интерфейсе. В этом смысле посылка сообщения «объекту-серверу» эквивалентна вызову соответствующего метода объекта.

Так вот, если при проектировании информационная система разбивается на объекты (компоненты), то UML может быть использован для ее визуального моделирования. Если используется функциональная декомпозиция ИС, то UML не нужен, и следует использовать другие (структурные) нотации.

С точки зрения визуального моделирования, UML можно охарактеризовать следующим образом. UML предоставляет выразительные средства для создания визуальных моделей, которые:

- единообразно понимаются всеми разработчиками, вовлеченными в проект и
- являются средством коммуникации в рамках проекта.

Унифицированный Язык Моделирования (UML):

- не зависит от объектно-ориентированных (ОО) языков программирования,
- не зависит от используемой методологии разработки проекта,
- может поддерживать любой ОО язык программирования.

UML является *открытым* и обладает средствами расширения базового ядра. На UML можно содержательно описывать классы, объекты и компоненты в различных предметных областях, часто сильно отличающихся друг от друга.

Как создавался UML

В середине 90-х существовало более 50 различных объектно-ориентированных языков моделирования. И разработчики, и заказчики испытывали беспокойство при выборе метода проектирования ИС, который, как правило, включал в себя и собственную нотацию. В это время стали появляться новые версии таких распространенных методов как: Booch'93, OMT-2 (Object Modelling Technique), Fusion, OOSE (Object-Oriented Software Engineering). Возникла насущная потребность в стандартизации и унификации.

Разработка UML была начата в октябре 1994 Грэди Бучем (Grady Booch) и Джимом Рамбо (Jim Rumbaugh) в Rational Software Corporation как унификация двух методов: Booch'93 и OMT. Первая версия Унифицированного Метода (Unified Method 0.8) была опубликована в октябре 1995. Осенью 1995 к работе присоединился Айвер Якобсон (Ivar Jacobson), включив в процесс унификации свой метод OOSE. Таким образом, на первом концептуальном этапе UML имел трех авторов: Буча, Рамбо и Якобсона, каждый их которых являлся идеологом своего ОО метода визуального моделирования.*

В октябре 1996 года была выпущена редакция UML 0.91, в которой были отражены многочисленные пожелания, которые «три друга» получили в течение 1996 года. В это же время выяснилось, что ряд влиятельных организаций, связанных с компьютерным бизнесом, стал рассматривать UML как стратегический элемент своей деятельности. Катализатором объединения усилий по унификации UML стал выпуск консорциумом OMG (Object Management Group) «запроса на предложения» по UML (RFP - Request for Proposal). Известно, что выпуск RFP является первым шагом процедуры принятия OMG того или иного стандарта. После этого Rational Software под своей эгидой создала организацию

«Консорциум UML партнеров» («UML Partners consortium») для выработки формального определения UML 1.0 как стандарта. В работе консорциума приняли участие представители таких известных компаний как: **Digital Equipment Corp., Hewlett-Packard, i-Logix, IntelliCorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI, Unisys.**

В результате в январе 1997 г. в OMG был представлен вариант UML 1.0. как первый RFP-отклик. В это же время второй RFP-отклик независимо от «UML Partners consortium» представили такие организации как: **ObjecTime; Platinum Technology; Ptech; Taskon & Reich Technologies и Softeam. Для объединения предложений по двум представленным проектам** UML эти компании также присоединились к «UML Partners consortium», и в результате был подготовлен вариант UML 1.1. Именно этот вариант в ноябре 1997 года был утвержден как стандарт. Формальное описание UML 1.1 в виде семи pdf-файлов можно найти, например, на сайтах OMG, Rational Software, Platinum Technology. По заголовкам можно судить о составе официальной документации по UML:

1. UML Summary,
2. UML Notation Guide,
3. UML Semantics,
4. UML OCL (Object Constraint Language Specification),
5. UML Objectory (UML Extension for Objectory Process for Software Engineering),
6. UML Business (UML Extension for Business Modeling),
7. UML Metamodel_Diagram.

Принципы моделирования

Использование языка UML основывается на следующих общих принципах моделирования:

- **абстрагирование** - в модель следует включать только те элементы проектируемой системы, которые имеют непосредственное отношение к выполнению ей своих функций или своего целевого предназначения. Другие элементы опускаются, чтобы не усложнять процесс анализа и исследования модели;
- **многомодельность** - никакая единственная модель не может с достаточной степенью точности описать различные аспекты системы. Допускается описывать систему некоторым числом взаимосвязанных представлений, каждое из которых отражает определенный аспект её поведения или структуры;
- **иерархическое построение** - при описании системы используются различные уровни абстрагирования и детализации в рамках фиксированных представлений. При этом первое представление системы описывает её в наиболее общих чертах и является представлением концептуального уровня, а последующие уровни раскрывают различные аспекты системы с возрастающей степенью детализации вплоть до физического уровня. Модель физического уровня в языке UML отражает компонентный состав проектируемой системы с точки зрения ее реализации на аппаратурной и программной платформах конкретных производителей.

Сущности в UML

В UML определены четыре типа сущностей: **структурные**, **поведенческие**, **группирующие** и **аннотационные**. Сущности являются основными объектно-ориентированными элементами языка, с помощью которых создаются модели.

Структурные сущности - это имена существительные в моделях на языке UML. Как правило, они представляют статические части модели, соответствующие концептуальным или физическим элементам системы. Примерами структурных сущностей являются «класс», «интерфейс», «кооперация», «прецедент», «компонент», «узел», «актер».

Поведенческие сущности являются динамическими составляющими модели UML. Это глаголы, которые описывают поведение модели во времени и в пространстве. Существует два основных типа поведенческих сущностей:

- взаимодействие - это поведение, суть которого заключается в обмене сообщениями между объектами в рамках конкретного контекста для достижения определенной цели;
- автомат - алгоритм поведения, определяющий последовательность состояний, через которые объект или взаимодействие проходят в ответ на различные события.

Группирующие сущности являются организующими частями модели UML. Это блоки, на которые можно разложить модель. Такая первичная сущность имеется в единственном экземпляре - это пакет.

Пакеты представляют собой универсальный механизм организации элементов в группы. В пакет можно поместить структурные, поведенческие и другие группирующие сущности. В отличие от компонентов, которые реально существуют во время работы программы, пакеты носят чисто концептуальный характер, то есть существуют только в процессе разработки.

Аннотационные сущности - это пояснительные части модели UML: комментарии для дополнительного описания, разъяснения или замечания к любому элементу модели. Имеется только один базовый тип аннотационных элементов - примечание. Примечание используют, чтобы снабдить диаграммы комментариями или ограничениями, выраженными в виде неформального или формального текста.

Отношения в UML

В языке UML определены следующие типы отношений: **зависимость**, **ассоциация**, **обобщение** и **реализация**. Эти отношения являются основными связующими конструкциями UML и также как сущности применяются для построения моделей.

Зависимость (dependency) - это семантическое отношение между двумя сущностями, при котором изменение одной из них, независимой, может повлиять на семантику другой, зависимой.

Ассоциация (association) - структурное отношение, описывающее совокупность смысловых или логических связей между объектами.

Обобщение (generalization) - это отношение, при котором объект специализированного элемента (потомок) может быть подставлен вместо объекта обобщенного элемента (предка). При этом, в соответствии с принципами объектно-ориентированного программирования, потомок (child) наследует структуру и поведение своего предка (parent).

Реализация (realization) является семантическим отношением между классификаторами, при котором один классификатор определяет обязательство, а другой гарантирует его выполнение. Отношения реализации встречаются в двух случаях:

- между интерфейсами и реализующими их классами или компонентами;
- между прецедентами и реализующими их кооперациями.

Общие механизмы UML

Для точного описания системы в UML используются, так называемые, **общие механизмы**:

- **спецификации** (specifications);
- **дополнения** (adornments);
- **деления** (common divisions);
- **расширения** (extensibility mechanisms).

UML является не только графическим языком. За каждым графическим элементом его нотации стоит **спецификация**, содержащая текстовое представление соответствующей конструкции языка. Например, пиктограмме класса соответствует спецификация, которая описывает его атрибуты, операции и поведение, хотя визуально, на диаграмме, пиктограмма часто отражает только малую часть этой информации. Более того, в модели может присутствовать другое представление этого класса, отражающее совершенно иные его аспекты, но, тем не менее, соответствующее спецификации. Таким образом, графическая нотация UML используются для визуализации системы, а с помощью спецификаций описывают ее детали.

Практически каждый элемент UML имеет уникальное графическое изображение, которое дает визуальное представление самых важных его характеристик. Нотация сущности «класс» содержит его имя, атрибуты и операции. Спецификация класса может содержать и другие детали, например, видимость атрибутов и операций, комментарии или указание на то, что класс является абстрактным. Многие из этих деталей можно визуализировать в виде графических или текстовых **дополнений** к стандартному прямоугольнику, который изображает класс.

При моделировании объектно-ориентированных систем существует определенное **деление** представляемых сущностей.

Во-первых, существует деление на классы и объекты. Класс - это абстракция, а объект - конкретное воплощение этой абстракции. В связи с этим, практически все конструкции языка характеризуются двойственностью «класс/объект». Так, имеются прецеденты и экземпляры прецедентов, компоненты и экземпляры компонентов, узлы и экземпляры узлов. В графическом представлении для объекта принято использовать тот же символ, что и для класса, а название подчеркивать.

Во-вторых, существует деление на интерфейс и его реализацию. Интерфейс декларирует обязательства, а реализация представляет конкретное воплощение этих обязательств и обеспечивает точное следование объявленной семантике. В связи с этим, почти все конструкции UML характеризуются двойственностью «интерфейс/реализация». Например, прецеденты реализуются кооперациями, а операции - методами.

UML является открытым языком, то есть допускает контролируемые **расширения**, чтобы отразить особенности моделей предметных областей. **Механизмы расширения** UML включают:

- **стереотипы** (stereotype) - расширяют словарь UML, позволяя на основе существующих элементов языка создавать новые, ориентированные для решения конкретной проблемы;
- **помеченные значения** (tagged value) - расширяют свойства основных конструкций UML, позволяя включать дополнительную информацию в спецификацию элемента;
- **ограничения** (constraints) - расширяют семантику конструкций UML, позволяя создавать новые и отменять существующие правила.

Совместно эти три механизма расширения языка позволяют модифицировать его в соответствии с потребностями проекта или особенностями технологии разработки.

Виды диаграмм UML

Графические изображения моделей системы в UML называются **диаграммами**. В терминах языка UML определены следующие их виды:

- **диаграмма вариантов использования или прецедентов** (use case diagram)
- **диаграмма классов** (class diagram)
- **диаграммы поведения** (behavior diagrams)
- **диаграмма состояний** (statechart diagram)
- **диаграмма деятельности** (activity diagram)
- **диаграммы взаимодействия** (interaction diagrams)
- **диаграмма последовательности** (sequence diagram)
- **диаграмма кооперации** (collaboration diagram)
- **диаграммы реализации** (implementation diagrams)
- **диаграмма компонентов** (component diagram)
- **диаграмма развертывания** (deployment diagram)

Каждая из этих диаграмм конкретизирует различные представления о модели системы. При этом, диаграмма вариантов использования представляет концептуальную модель системы, которая является исходной для построения всех остальных диаграмм. Диаграмма классов является логической моделью, отражающей статические аспекты структурного построения системы, а диаграммы поведения, также являющиеся разновидностями логической модели, отражают динамические аспекты её функционирования. Диаграммы реализации служат для представления компонентов системы и относятся к ее физической модели.

Из перечисленных выше диаграмм некоторые служат для обозначения двух и более подвидов. В качестве же самостоятельных представлений используются следующие диаграммы: **вариантов использования**, **классов**, **состояний**, **деятельности**, **последовательности**, **кооперации**, **компонентов** и **развертывания**.

Для диаграмм языка UML существуют три типа визуальных обозначений, которые важны с точки зрения заключенной в них информации:

- **связи**, которые представляются различными линиями на плоскости;
- **текст**, содержащийся внутри границ отдельных геометрических фигур;
- **графические символы**, изображаемые вблизи визуальных элементов диаграмм.

При графическом изображении диаграмм рекомендуется придерживаться следующих правил:

- каждая диаграмма должна быть законченным представлением некоторого фрагмента моделируемой предметной области;
- представленные на диаграмме сущности модели должны быть одного концептуального уровня;
- вся информация о сущностях должна быть явно представлена на диаграмме;

- диаграммы не должны содержать противоречивой информации;
- диаграммы не следует перегружать текстовой информацией;
- каждая диаграмма должна быть самодостаточной для правильной интерпретации всех ее элементов;
- количество типов диаграмм, необходимых для описания конкретной системы, не является строго фиксированным и определяется разработчиком;
- модели системы должны содержать только те элементы, которые определены.

Литература

1. http://www.interface.ru/public/990804/uml4b.htm
2. http://www.interface.ru