Лекция № 2. Точка зрения моделирования, модель и диаграммы, иерархия метаописаний.

В этой лекции рассматривается, что такое точка зрения (viewpoint) моделирования, показывается необходимость для успешной разработки программной системы создания множества моделей, выполненных с разных точек зрения. Результаты визуального моделирования разделяются на модель и ее представления (диаграммы), рассматриваются операции над моделью и диаграммами. Рассматривается также иерархия метаописаний, необходимая при создании, изучении и использовании формальных языков визуального моделирования.

Множество точек зрения на ПО. Сложность и невидимость ПО часто не дают возможности положить в его основание простые, понятные и наглядные концепции. ПО объективно сложно, поэтому в процессе разработки на него удобно смотреть то как на черный ящик, реализующий некоторый набор пользовательской функциональности (не важно как), то как на аппаратуру (различные сервера, рабочие станции, специальное оборудование, например, телекоммуникационное), то как на набор размещенных в run time на этой аппаратуре программных компонент, хранилищ данных, конфигурационных файлов, то как на набор классов и т.д. – см. рис. 2.1.

Разработчик

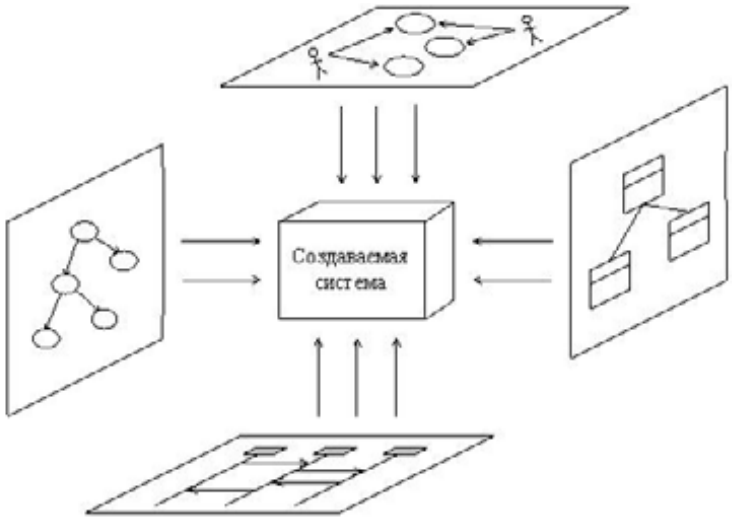

Рис. 2.1

С другой стороны, ПО является объектом бизнеса купли/продажи, адресуется пользователям-непрограммистам, взаимодействует с различными механическим, электронными, социальными системами. Поэтому в его разработку/использование вовлечено большое количество *очень разных* специалистов: программисты, инженеры, тестеры, технические писатели, менеджеры, заказчик, пользователи, продавцы-маркетологи и т.д. Для всех эти специалистов нужна разная информация о программной системе.

Рис. 2.2.

Итак, разные функции по разработке ПО и разные пользователи информации о ПО индуцируют создание разных моделей ПО, моделей, выполненных с разных точек зрения.

Из всего этого следует, что необходимы различные модели ПО, представляющие его с разных точек зрения, соответствующих тем работам, которые выполняются разными специалистами в разное время разработки.

Определение точки зрения моделирования. *Точка зрения моделирования* (viewpoint) – это определенный взгляд на систему, который осуществляется кем-либо из участников проекта для выполнения какой-то определенной задачи. Поскольку выполнение этой задачи может подразумевать общение с другими участниками проекта, а визуальные модели предназначаются именно для того, чтобы люди в процессе

выполнения задачи лучше понимали друг друга, то при создании модели нужно хорошо определить *аудиторию*, на которую рассчитывается эта модель. Итак, точка зрения моделирования определяется целью модели и ее аудиторией.

Определение точки зрения моделирования, на первый взгляд, нет ничего нового не привносит. Например, при создании различных инженерных объектов активно используется эта же концепция – принципиальная схема, монтажная схема, генеральный план, различные проекции и разрезы деталей, зданий и пр. – это все различные модели системы, выполненные с разных точек зрения на нее. Однако в обычных инженерных областях есть стандартные, зафиксированные точки зрения на систему, и им соответствуют стандартные же модели. Например, электрик при создании электропроекта жилого дома не изобретает различные виды чертежей и описаний, а руководствуется существующими нормативами (в России это свод документов, называемый ПУЭ – Правила Устройств Электроустановок). В случае с визуальным моделированием нет таких стандартизированных видов моделей. Существуют, конечно же, виды диаграмм в UML, но какие из них и когда использовать, какую часть системы с их помощью прорисовывать – решать самими разработчикам. Один из классиков объектно-ориентированного анализа и проектирования Грэди Буч многократно подчеркивал в своих книгах, что его метод – это не поварская книга готовых рецептов. Поэтому создание полезных визуальных моделей является более сложным делом, чем создание чертежей в других инженерных областях. И ясно сформулированная точка зрения на систему, которой придерживаются все время при создании модели – это один из основных критериев того, что модель действительно принесет пользу.

Например, диаграммы объектов UML предназначены для моделирования фрагментов системы, и сразу появляется вопрос – каких именно фрагментов. Далее, существует очень много разных стратегий по созданию диаграмм случаев использования (use case diagrams): одни авторы считают, что нужно создавать не много случаев использования, даже для крупных систем, другие предпочитают строить огромные полотна, одни считают, что не нужно подробно изображать окружение системы на этих диаграммах (только тех актеров, которые непосредственно взаимодействуют с системой), другие считают это важным и т.д.

Концепция точки зрения моделирования появилась при самом зарождении использования графовых нотаций для проектирования ПО, в конце 60-х годов, в составе подхода SADT (Structured Analysis and Design Technique) [4]. Однако в SADT использовалась единственная графическая нотация – просто различных моделей системы могло быть много. Тем не менее авторы подхода, будучи серьезно озабочены эффективностью моделирования, разработали подробные рекомендации относительно того, как определять фокус моделирования, а также как его удерживать при создании моделей. Позднее, при дальнейшем развитии структурного анализа (70-80-е годы), появились разные виды диаграмм (сущность-связь, потоков данных, состояний и переходов и т.д.) [3], и идея использовать все это многообразие при разработке ПО никого не смутила. Однако лишь впоследствии, в 1995 году, уже в рамках объектно-ориентированного подхода, Филиппом Кратченом [5] была в явном виде обоснована идея разных, равноправных точек зрения на систему при ее разработке, принципиальной несводимых друг к другу. В дальнейшем эти идеи легли в основу UML, который был создан как множество нотаций, с помощью которых можно представить систему с разных точек зрения (эта концепция в явном виде присутствовала в первых версиях стандарта). Однако в последнее время делаются последовательные попытки повысить целостность UML, максимально связав исходно разные подмножества языка. По всей видимости, истина лежит в балансе между целостностью языка (и создаваемых на его основе моделях) и возможностью отражать разные аспекты системы по-разному, с помощью различных выразительных средств. Однако «поймать» такой баланс не просто.. ..

Разъяснения и примеры. Концепция точки зрения, не смотря на свою простоту и актуальность, часто не освоена на практике теми, кто создает визуальные модели при разработке конкретного ПО.

Например, часто создаются описание ПО вообще. В этом случае точка зрения есть, ее не может быть, но она не осознается. И часто вообще потеряны те люди, которым предназначена модель. В результате диаграммы оказываются никому не нужными.

Другой пример. Аналитик основывается на собственном, очень специфическом видении системы, и прямо таки навязывает его всем остальным участникам проекта. Если он обладает влиянием в проекте, а также большой энергией, то с ним оказывается очень трудно работать, а его диаграммы опять таки никому не понять, и пользы они проекту не приносят. Он кипятится, отсылает нерадивых разработчиков на курсы по UML, никто на эти курсы, разумеется не идет (работать надо, а не на курсы ходить). В общем, на лицо скрытый или явный конфликт.

Подобных сюжетов на практике происходит множество. И тут важно понимать, что цель модели – это не какая-то гипотетические задача типа «описания архитектуры вообще, потому что так нужно, так правильно», а аудитория – это не абстракция типа «люди, желающие познакомиться с ПО». И то и другое – что-то очень конкретное, реально существующее в проекте. Ведь разработчики ПО не могут позволить себе за деньги заказчика создавать нечто на все века и народы. И цель моделирования и аудитория, которая будет работать с диаграммами, всегда существуют, важно лишь ясно понимать, какие они…

Нужно также отметить, что ясное понимание точки зрения на систему при использовании визуального моделирования означает, в частности, что мы осознаем *зачем* мы создаем эту модель. Потому что так правильно и все проблемы (даже те, о которых мы не знаем) волшебным образом исчезнут, развеются? Очень часто, например, при создании модели случаев присутствует именно такая цель моделирования. А потом оказывается, что никакие проблемы не «вылечились», а, наоборот, возникли новые (например, созданные нами диаграммы никто не понимает и не принимает). Да и сам аналитик чувствует, что диаграммы получились какие-то странные….

А может быть все совсем не так: аналитик действительно задался целью выявить требования к системе (не навязать свое собственное видение, а именно выявить, смоделировать), ему важно, чтобы будущие пользователи системы могли участвовать в этом процессе, диаграммы рисуются для них, они понятны и не избыточны в смысле формальных средств. И эти же диаграммы структурируют и проясняют информацию для самого аналитика (то есть они предназначаются не только для работы с пользователями, но и для аналитика). Это – совсем другой сюжет, здесь другая точка зрения моделирования. Важно, что концепция точки зрения моделирования существенно глубже, чем несколько строк текста с определением цели и аудитории создаваемой модели системы. Она захватывает еще отношение аналитика к делу, к людям, с которыми он общается.

Модель и диаграммы. Пусть мы выбрали точку зрения и создаем визуальную модель нашего ПО. И пусть модель получается большой и подробной. Она может содержать несколько (или даже много) диаграмм одного или нескольких видов. Мы активно обсуждаем эту модель с разными специалистами, меняем отдельные ее детали, доделываем и улучшаем ее.

В настоящее время вся эта работа выполняется с помощью программных инструментов типа IBM Rational Rose, Borland Together Control Center и т.д. Эти средства – не просто специализированные графические редакторы, поддерживающие библиотеки соответствующих графических символов и облегчающие процесс создания диаграмм и внесение в них изменений. Эти пакеты позволяют создавать целостную модель, представление которой разбито по разным диаграммам – см. рис. 2.3.

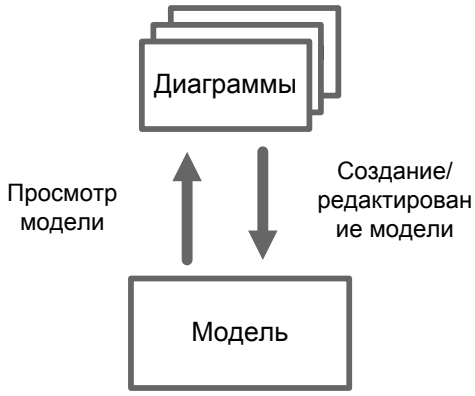

Рис. 2.3.

Итак, будем различать модель и представляющие ее диаграммы. Наличие единой модели существенно упрощает процесс редактирования и внесений изменений в диаграммы, если мы хотим, чтобы они при этом оставались согласованными. Рассмотрим пример.

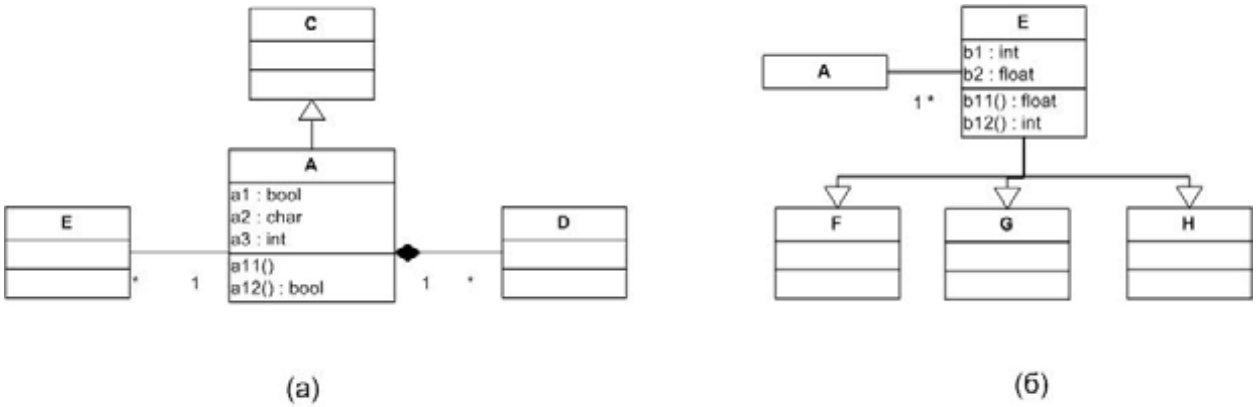

Рис. 2.4.

На рис. 2.4 (а) показана диаграмма классов, где приведена полная спецификация класса A – показаны все его атрибуты, операции, все его предки в иерархии наследования, а также все другие классы, которые связаны с ним какими-либо отношениями. На рис. 2.4 (б) представлена диаграмма классов, где аналогично определяется класс E. Классы A и E связаны друг с другом ассоциацией, поэтому будут присутствовать на обоих диаграммах. Пересечение двух этих диаграмм очевидно. А теперь предположим, что мы изменили имя класса E на рис. 2.4 (б). Очевидно, что на диаграмме с рис. 2.4 (а) это имя тоже должно измениться. Поскольку обе диаграммы изображают одну и ту же модель, то при первом переименовании второе произойдет автоматически.

И это мы рассмотрели еще постой пример. А диаграммы могут принадлежать разным типам и все равно быть связанными по информации. Например, мы нарисуем диаграмму объектов и там будет объект класса E. При измени имени данного класса данная диаграмма должна также измениться. Наконец, есть модельная информация, которая вовсе не отображается на диаграммах, однако нужна. Например, диаграммы могут образовывать иерархию – быть сгруппированы в пакеты, принадлежать отдельным модельным сущностям (например, набор диаграмм состояний и переходов может определять поведение одной компоненты). Для того, чтобы хранить всю эту информацию, связывающую разные модели в единое целое в тех местах, где они пересекаются по информации, и используется единая модель системы.

По сути, модель является собранными вместе, в один граф, всех модельных сущностей и их связей, «склеенных» вместе с устранением дублирования информации, а также присоединением дополнительной информации, отсутствующей на диаграммах.

Например, мы можем представить на одной диаграмме классов модель, составленную из двух диаграмм классов рис. 2.4 (а) и (б), как показано на рис. 2.5.

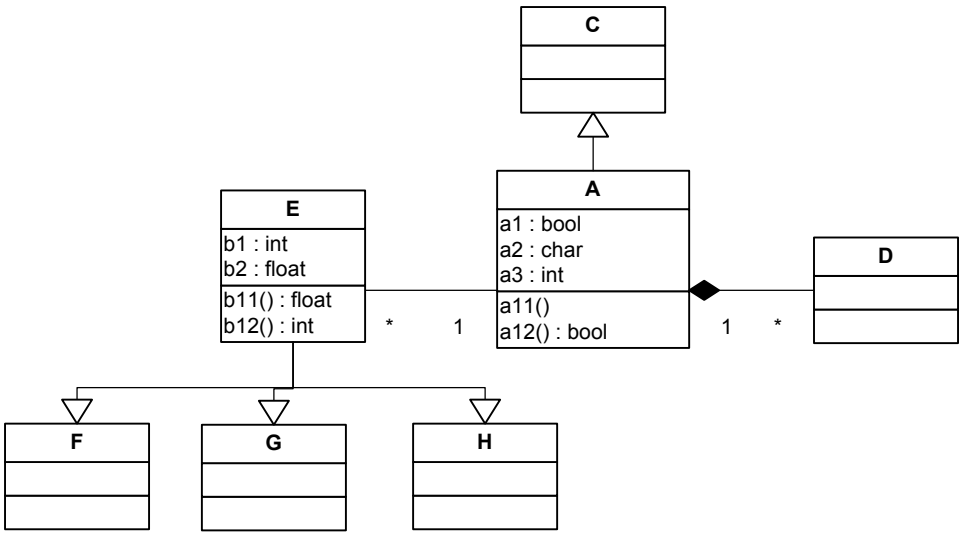

Рис. 2.5.

Диаграммы помогают создавать модель, а также просматривать и изучать ее. Модель же является хранилищем всей созданной информации, причем хранилищем «умным». Что это значит? А именно, что модель не есть склад «диаграмм». В таком случае класс А с рис. 2.4 (а) и тот же класс с рис. 2.4 (б) были бы разными сущностями. Модель хранит граф сущностей и связей, фрагменты которого отображаются на диаграммах. Если диаграммы сопоставить с исходными тестами программ, то модель – это проанализированные тексты, превращенные в синтаксический граф. Подобный анализ происходит при компиляции программ в исполняемый код, а в случае визуальных моделей он происходит раньше – при сохранении диаграмм. Взаимосвязь диаграмм и модели показана на рис. 2.3.

Об операциях над моделью и диаграммами. Теперь скажем несколько слов об операциях над моделью и диаграммами. Если бы в нашей модели, изображенной на рис. 2.5, еще не было класса А, то добавление его на любую диаграмму, относящуюся к этой модели, возможно только в режиме «добавить в модель». Но если такой класс уже существует в модели, а мы хотим его только отобразить на очередной диаграмме, выполняется операция «загрузить на диаграмму». То есть если мы, например, создал класс А на диаграмме с рис. 2.4 (а), подробно описали все его атрибуты, а потом приступили к созданию диаграммы на рис. 2.4 (б), то на эту последнюю диаграмму класс А «загружается». При желании мы можем загрузить также все его атрибуты и методы, а также другие классы, которые с ним связаны. Разница между добавлением в модель и загрузкой на диаграмму должна быть очевидна : в обоих случаях мы добавляем элемент на диаграмму, но в первом случае добавляем еще и в модель, а во втором случае – нет. Во втором случае, наоборот, из модели берется вся необходимая информация о данном классе и отображается на диаграмме.

К двум этим операциям есть пара двойственных им – «удалить из модели» и «выгрузить с диаграммы». Их смысл очевиден.

Все перечисленные выше операции выполнялись через диаграммы. Но, как правило, можно удалить/добавить элемент в модель и помимо диаграмм. Это можно сделать, например, программно, то есть через скрипт или приложение, которое обращается к хранилищу модели через программный интерфейс. В таком случае, если мы

удалили элемент из модели, то он должен исчезнуть со всех диаграмм. При добавлении в модель, вообще говоря, элемент не обязан появляться на какой-либо диаграмме.

Модель, метамодель, метаметамодель. Использование формальных языков для описания объектов действительности породило концепцию метауровней описаний. Каждый уровень – это некоторая классификация предыдущего уровня, по сути, язык для его описания. Первый уровень является исходным и представляет реальную действительность, классифицируемую последующими уровнями метаописаний.

Фактически, конструкции языка описаний, предоставляемые следующим уровнем, типизируют сущности предыдущего уровня, создавая для них метапонятия (см. рис. 2.6.). Например, система классификации видов является тем языком, с помощью которого биологи описывают живые существа нашей планеты. Таким образом многообразие объектов предыдущего уровня структурируется и классифицируется объектами следующего уровня. В нашем примере объектами следующего уровня являются виды, подвиды, отряды и т.д.

Иерархия метауровней возникает из-за того, что для описания одной предметной области появляется много разных языков. Вследствие этого возникает потребность в надъязыке, упорядочивающим их структуру и предоставляющим средства для создания новых подобных языков. Так де-факто стандартом для описания синтаксиса языков программирования является грамматика в форме Бэкуса-Науора. Общая предметная область является обязательным условием для множественности метауровней, большей, чем 2, поскольку вряд ли целесообразно использовать, например, одинаковые средства для описания естественных языков, языков программирования и системы видов в биологии.

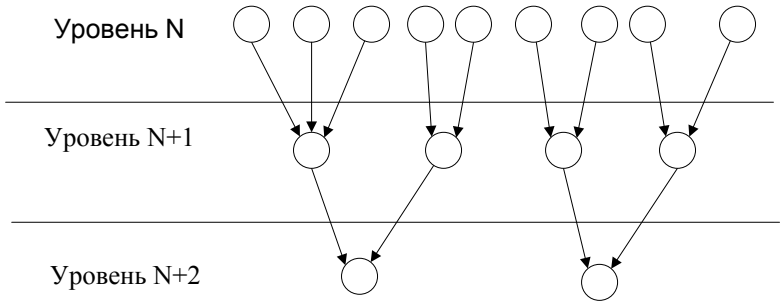

Рис. 2.6.

Рассматривая язык UML, целесообразно выделить четыре метауровня уровня (см. рис. 2.7). Рассмотрим эти уровни детально.

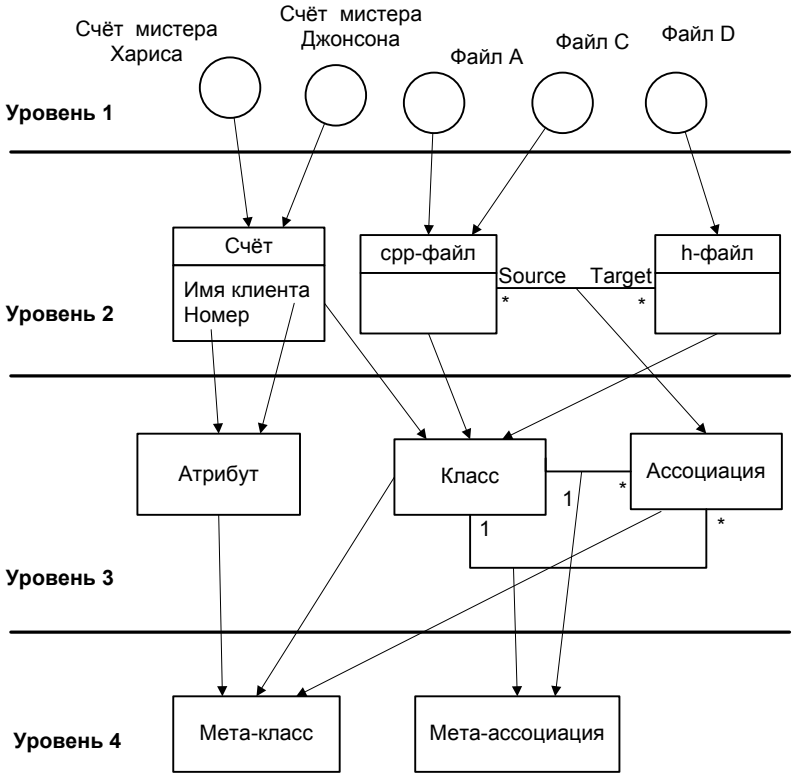

Рис. 2.7.

1. Предметная область (user objects), автоматизируемая с помощью создаваемой программной системы, например, бизнес некоторого банка. В качестве отдельных элементов там могут выступать, например, счета конкретных клиентов банка – мистера Джонсана, мистера Хариса, – факты обращения различных клиентов в банк и т.д. В качестве предметной области UML-моделирования может выступать и само ПО – например, при реинжиниринге.

2. Модель (model) – это язык описания предметной области. Например, для счетов мистера Хариса и мистера Джонсона и пр. введено понятие Счет с определенным набором атрибутов. Целью создания UML-моделей является создание классификационной схемы объектов предметной области, позволяющей их упорядочить и описать в единообразном стиле. Поскольку можно составить бесконечное множество разных классификаций (моделей) одной и той же предметной области, при моделировании необходимо определить, как указывалось при изучении SADT, цель и точку обзора.

3. Метамодель (meta-model) – это язык построения моделей. Стремление создать единый язык вместо различных модельных техник как раз и привело к созданию UML. При этом его авторы подчеркивали, что стандартизуется, в первую очередь, именно язык – способы его применения сильно зависят от различных индивидуальных особенностей конкретных проектов, предметных областей, личных пристрастий разработчиков и аналитиков и т.д.

4. Мета-метамодель (meta-metamodel) – это язык описания метамодели. Дело в том, что даже в рамках UML существуют различные виды диаграмм, то есть он неоднороден. При создании UML, объединяя разные типы диаграмм, пришлось использовать стандартный подход к их описанию – фрагмент его самого (точнее, модель классов). Таким способом можно описать и другие языки визуального моделирования.

В данном случае четырех уровней достаточно, поскольку мета-метамодель UML оказывается фрагментом самого UML: подмножеством диаграмм классов, которое

стандартизировано OMG как язык описания метамоделей – MOF (Meta Object Facility). С его помощью описываются такие стандарты OMG как Common Warehouse Metamodel (CWM), CORBA Component Model (CCM) и др.

Однако, MOF является фрагментом UML скорее де-факто, чем де-юре. Несмотря на фактически полное вложение в UML, MOF имеет собственное описание и большое количество косметических отличий от последнего. Такое положение сложилось в связи с тем, что MOF был стандартизирован OMG раньше, чем UML.

Основная литература

1. Г.Буч, А.Якобсон, Дж. Рамбо. UML. / Пер. с англ. – Изд. 2-ое. Питер 2006. – 735 с.
2. UML 2.0 Infrastructure Specification, September, 2004. http://www.omg.org/
3. Д.В. Кознов. Языки визуального моделирования: проектирование и визуализация программного обеспечения. Учебное пособие. – Изд-во СПбГУ, 2004. – 143 с.

Дополнительная литература

4. D.A.Marca, C.L.McGowan, SADT Structured Analysis and Design Technique. – McGraw-Hill, 1988.
5. P.Kruchten. The 4+1 View Model of Architecture. – IEEE Software, 1995, 12(6). – P. 42-50.