## Тема 4.2. Иерархия мета описаний. Моделирование. Граф модели и диаграммы

4.2.1. [Предметная область](http://www.intuit.ru/department/se/vismodtp/2/#sect1) и [множество моделей ПО](http://www.intuit.ru/department/se/vismodtp/2/2.html#sect3)

4.2.2.[Граф модели и диаграммы](http://www.intuit.ru/department/se/vismodtp/2/3.html#sect4) и операции над ними

4.2.3. [Контрольные вопросы](http://www.intuit.ru/department/se/vismodtp/2/3.html#sect6) по теме «Иерархия мета описаний. Моделирование. Граф

модели и диаграммы»

4.2.4. Тестовые задания по теме «Иерархия мета описаний. Моделирование. Граф

модели и диаграммы»

### 4.2.1. Предметная область и множество моделей ПО

При визуальном моделировании программного обеспечения используются следующие уровни абстракции:

* предметная область;
* модель;
* метамодель;
* метаметамодель.

Для визуального моделирования в качестве предметной области обычно выступает:

* тот фрагмент действительности, куда создаваемое ПО будет встроено: бизнес-процессы компании, для которой создается информационная система, электромеханическая среда для встроенного ПО и т. д.; программистам необходимо тщательно изучить тот контекст, в котором их ПО будет работать, чтобы оно было там адекватно;
* архитектурные решения ПО, которые должны быть тщательно проработаны, обсуждены с разными специалистами и ими понятны; с этой целью они и подвергаются визуализации.

Модель – это упрощенное описание предметной области, созданное для удобства выполнения там действий, работы. Более простая модель дает возможность не рассматривать все бесконечное многообразие предметной области, а сосредоточиться лишь на некоторых ее свойствах. Например, для создания информационной системы автоматизации предприятия строится модель предприятия, которая фокусируется на бизнес-процессах, потоках данных, бизнес-ролях. В эту модель не входит следующая информация о предприятии: межличностные отношения сотрудников, детали планировки помещений офисов, расписание работы компании (начало работы, обеденный перерыв, выходные) и т. д.

При визуальном моделировании ПО обычно строятся следующие модели.

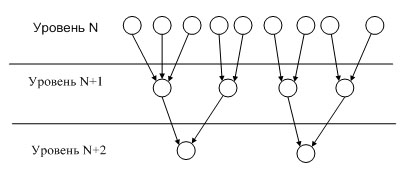
* Модели анализа (analysis models), формализующие результаты изучения программистами того контекста, где будет работать их будущее ПО; эти модели позволяют хорошо формализовать требования к ПО, согласовать их с будущими пользователями системы, заказчиком и др. заинтересованными лицами, тем самым, создав хорошую основу для дальнейшей разработки программной системы.
* Модели проектирования (design models), в которых фиксируются архитектурные решения будущего ПО - его структура, внешние и внутренние интерфейсы, принципиальные вопросы реализации с учетом средств разработки, платформ исполнения и т.д.

Модели анализа должны "плавно" переходить в модели проектирования, и это является одним из главных принципов модельно-ориентированного подхода к разработке ПО.

В индустриальном производстве создание той или иной модели - это не единичный прецедент. Например, люди, специализирующиеся на разработке информационных систем, создают много моделей разных компаний. Соответственно, у них возникает потребность в специальном языке, который существенно упростил бы разработку таких моделей. Этот язык должен содержать описание всех тех абстракций, которые обычно нужны при моделировании деятельности предприятий. Само множество этих моделей оказывается предметной областью для новой модели, которую поэтому естественно называть метамоделью (metamodel).

В рамках одной области деятельности может быть востребовано много разных модельных языков, и тогда необходим общий способ по их разработке и спецификации. В этом случае оказывается востребованным язык описания языков (метамоделей) - метаметамодель (meta-metamodel). Предметной областью для этой новой модели являются соответствующие метамодели.

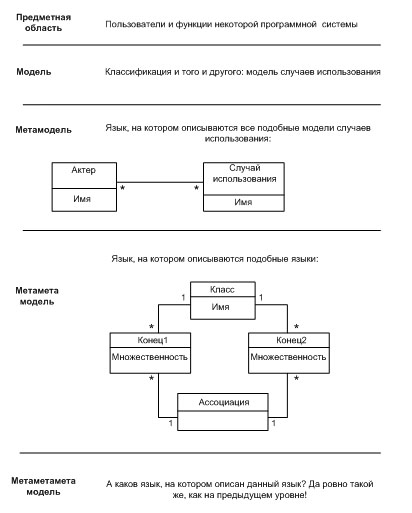
Теоретически, приведенную выше цепочку метауровней можно продолжать бесконечно. Каждый следующий уровень будет служить моделью для предыдущего, а предыдущий уровень оказывается для него предметной областью, как показано на [рис. 4.2.1](http://www.intuit.ru/department/se/vismodtp/2/#image.2.1).



**Рис. 4.2-1.**  Уровни моделирования

Переход на следующий метауровень целесообразен лишь тогда, когда на некотром уровне появляется много сходных объектов, нуждающихся в структурировании, а, значит, в метаописании. В какой-то момент будет достигнут предел по количеству объектов, требующих унификации и упорядочивания. На [рис. 4.2.2](http://www.intuit.ru/department/se/vismodtp/2/#image.2.2) приведен пример четырех метауровней с кратким обоснованием, почему пятый уровень и далее не нужны.

1. Предметная область - некоторая программная система, ее функции и пользователи. Пользователей у системы могут быть десятки, сотни и даже тысячи, функциональность может быть очень сложной. Очевидно, что необходима специальная модель, структурирующая все это разнообразие.
2. Модель в данном случае - это одна или несколько диаграмм случаев использования, классифицирующих и описывающих функции системы и ее пользователей. Пользователи сгруппированы по типам, функциональность - по случаям использования (см. следующую лекцию, где этот тип диаграмм будет описываться подробно). Очевидно, что разработчикам ПО приходится часто строить такие модели для разных систем.
3. Поэтому необходима метамодель, описывающая язык случаев использования. В данном случае, в упрощенном варианте она состоит из актера и случая использования, соединенных между собой связью "многие-ко-многим" - один актер может быть связан с несколькими случаями использования, несколько актеров могут быть связаны с одним случаем использования. Очевидно, что подобных метамоделей можно составить множество - для других визуальных языков.
4. Мета модель - это язык для создания метамоделей всех визуальных языков. В данном, упрощенном случае она состоит из класса и ассоциации.
5. Попытка построить метаметаметамодель приводит к забавному противоречию - получается ровно такая же диаграмма, как на предыдущем уровне (попробуйте - увидите сами!). Это происходит потому, что метаметамодель (п. 4) также описана с помощью некоторого визуального языка. А раз так, то этот новый язык тоже описываться средствами метаметамодели (см. определение метамодели в п. 4 - ведь она подходит для описания всех визуальных языков).



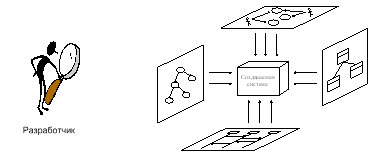
**Рис. 4.2-2.**  Пример четырех метауровней в визуальном моделировании

Есть и объективная предпосылка к тому, что пятый уровень оказывается вырожденным, безотносительно тому, какие средства используются для создания мета модели. Дело в том, что не существует большого множества средств для задания метамоделей визуальных языков (в следующих лекциях, посвященных DSM-подходу, будет продемонстрирована еще один подход - грамматики в форме Бэкуса-Науэра). И поэтому не возникает задачи по структурированию и упорядочиванию таких способов путем разработки для них общей модели. То есть мета модель не требуется…

Язык UML, являясь, очевидно, метамоделью, описан с помощью своего подмножества –диаграмм классов. Это подмножество стандартизовано в качестве стандартной мета модели как универсальное средство описания различных.

Выше уже говорилось, что модели ПО обычно бывают или моделями анализа, или моделями проектирования. На самом деле моделей оказывается значительно больше, правда, не все они визуализируются. Посмотрим, почему их оказывается много.

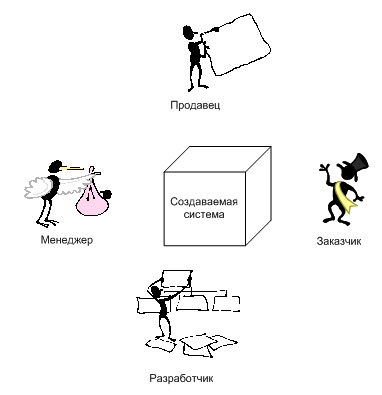
Прежде всего, модели в проекте «множатся» из-за разных видов деятельности процесса разработки ПО (см. [рис. 4.2-3](http://www.intuit.ru/department/se/vismodtp/2/2.html#image.2.3)).



**Рис. 4.2-3.**  Модели "множатся" из-за разных видов деятельности

При анализе на ПО смотрят как на то, что реализует определенную бизнес-функциональность, нужную заказчику. При этом несущественными оказываются принципы и детали реализации. При проектировании, наоборот, на первое место выходят принципы реализации ПО. А при тестировании детали реализации снова неважны - на ПО смотрят как на черный ящик, реализующий (не важно каким способом) некоторый набор пользовательской функциональности. При развертке у заказчика на ПО смотрят как на набор файлов, хранилищ данных и т. д.

Далее, в разработку/использование ПО вовлечено большое количество очень разных специалистов: программисты, инженеры, тестеры, технические писатели, менеджеры, заказчик, пользователи, продавцы-маркетологи и т. д. (см. [рис. 4.2-4](http://www.intuit.ru/department/se/vismodtp/2/2.html#image.2.4)). Для всех эти специалистов нужна разная информация о программной системе. Представьте, что произойдет, если, например, продавцу или заказчику-непрограммисту в ответ на просьбу получше ознакомиться с ПО вы дадите почитать программные коды…



**Рис. 4.2-4.**  Модели "множатся" из-за разного рода специалистов в проекте

Большое количество конфликтов и трудностей в проектах возникает просто из-за того, что одни специалисты не могут понять других. Например, частой является ситуация, когда инженерам по аппаратуре трудно понять программистов, которые создают ПО, взаимодействующее с этой аппаратурой. Программисты объясняют алгоритмы работы своих программ в терминах процедур, переменных, классов и т. д. И наоборот, инженеры "заваливают" программистов деталями реализации и функционирования своих устройств. Другой пример - очень часто технические писатели, создающие пользовательскую документацию для ПО, плохо разбираются в том программном обеспечении, которое они описывают. И документация получается никуда не годной, для галочки. Еще пример: менеджеры (особенно высокопоставленные, в больших проектах) часто не понимают реальных проблем проекта и склонны "расхлебывать" то, что уже произошло, а не реагировать на первые признаки неурядец. Подобные проблемы легче разрешаются, если в проекте существуют или могут быть созданы по требованию разные модели, предназначенные для различных специалистов, на которых в доступной форме и без лишних деталей представлена нужная информация.

Итак, разные виды деятельности при разработке ПО и разные категории специалистов, задействованные в программном проекте, - все это приводит к созданию и использованию различных моделей, выполненных с разных точек зрения.

Точка зрения моделирования (viewpoint) - это определенный взгляд на систему, который осуществляется для выполнения какой-то определенной задачи кем-либо из участников проекта. Далее будут рассматриваться только визуальные модели ПО, хотя многое из сказанного ниже справедливо также и для произвольных моделей.

На первый взгляд, введенное выше определение очевидно и 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. Все это является моделями создаваемой системы, выполненными с разных точек зрения. 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 Устройства Электроустановок). То же самое касается и проектировщиков зданий, конструкторов автомобилей, самолетов и т. д.

В случае с визуальным моделированием ПО таких стандартизированных видов моделей, к сожалению, не существует. Есть, конечно же, типы диаграмм в UML, но какие из них и когда использовать, какую часть системы с их помощью "прорисовывать" - решать самим разработчикам. Более того, само разбиение UML на разные типы диаграмм условно - диаграммы можно смешивать, как будет показано далее, в лекциях по UML.

Забегая вперед, замечу, что, например, диаграммы объектов UML предназначены для моделирования фрагментов системы, и сразу появляется вопрос - каких именно фрагментов? Решать приходится разработчикам, использующим эти диаграммы. 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, наоборот, считают это важным и т. п. Какой из этих способов избрать, или создать свой собственный, - опять-таки решать разработчикам конкретной системы.

Один из классиков визуального моделирования, Грэди Буч, многократно подчеркивал в своих книгах, что его метод - это не поварская книга готовых рецептов. Создание полезных визуальных моделей является более сложным делом, чем создание чертежей и спецификаций в других инженерных областях. И правильно выбранная и ясно сформулированная точка зрения на систему, которая не "плывет" при моделировании, - это один из основных критериев того, что модель действительно принесет пользу.

Важнейшими характеристиками точки зрения моделирования является цель (зачем создается модель) и целевая аудитория (то есть для кого она предназначается). Важным вопросом, на который нужно честно себе ответить в самом начале моделирования - это зачем вы используете UML. Это и есть определение цели моделирования. Потому что так создавать модели правильнее? И все проблемы (даже те, о которых ничего еще не известно) волшебным образом исчезнут, развеются? Очень часто, например, при создании модели случаев использования 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 все происходить совсем не так. Например, аналитик действительно задался целью выявить требования к системе - не навязать свое собственное видение другим, а выяснить нужную информацию, смоделировать и изложить ее доступно. Для этого он и использует диаграммы случаев использования. Ему важно, чтобы будущие пользователи системы могли участвовать в этом процессе, диаграммы рисуются для них, они понятны и не избыточны. И эти же диаграммы структурируют и проясняют информацию для самого аналитика.

Типична ситуация, когда UML используется, чтобы создавать модели ПО "вообще" - потому что так правильно, потому что люди недавно узнали, что такое UML и т.д. В этом случае какая-то точка зрения при моделировании все-таки есть, но она, как правило, не осознается авторами таких описаний. Цели моделирования расплывчаты и туманны, а люди, которым предназначены данные модели, вообще "потеряны". В результате такие диаграммы никому не нужны, а средства, затраченные на их создание, оказываются выброшенными на ветер.

Другой пример. Аналитик основывается на собственном, очень специфическом видении системы и прямо-таки навязывает его всем остальным участникам проекта, порождая с помощью UML многочисленные модели. Если он к тому же обладает влиянием в проекте, а также большой энергией, то от его UML-моделей не отмахнуться. В итоге с таким аналитиком оказывается очень трудно работать, в частности, его диаграммы никому не понятны и пользы проекту не приносят. Он кипятится, отсылает нерадивых разработчиков к литературе по UML, но никто, разумеется, эти книги не читает (работать надо!). В общем, налицо скрытый или явный конфликт.

Подобных сюжетов на практике происходит множество. Тут важно понимать, что цель модели - это не какая-то гипотетическая задача типа "описания архитектуры, потому, что создавать модели правильно?", а целевая аудитория - это не абстракция типа "люди, желающие познакомиться с ПО". И то и другое - что-то очень конкретное, реально существующее в проекте или рядом с ним. Ведь разработчики ПО не могут позволить себе за деньги заказчика создавать нечто на все века и для всех народов. И цель моделирования, и аудитория, которая будет работать с диаграммами, всегда существуют, важно лишь ясно понимать, какие они…

Вот полезный практический прием для фокусировки на целевую аудиторию, для которой предназначена создаваемая вами модель. Можно выбрать одного представителя такой аудитории - конкретного и известного вам человека - и создавать диаграммы, понятные именно ему. При этом важно не обсуждать чрезмерно с ним ваши модели, поскольку это может создать дополнительный контекст, которого другие пользователи моделей будут лишены. Полезно представлять воображать себе этого человека при работе над моделями - его реакции, вопросы, недоумения и пр. И, исходя из этого, корректировать, исправлять созданное. И, конечно же, полезно проверить свои предположения, показав ему, что получилось.

Кроме того, важно, чтобы точка зрения была "живая", а не выдумывалась аналитиком или бездумно копировалась из книжек и тренингов, посвященных UML. Незаметно для себя аналитик может придумать свой собственный проект, своих собственных пользователей системы, заказчика и т.д. Он может исподволь навязывать самому себе определенное восприятие реально существующих людей, задач, сильно искажая реальное положение дел. И именно в контексте этой воображаемой ситуации он будет создавать свои модели… Идеальный аналитик должен обладать гибкостью сознания, а также чуткостью и искренним стремлением к тому, чтобы сделать каждый конкретный проект, где он участвует, более гармоничным, более адекватным. И в любом случае иметь дело с реальной ситуацией, облегчая, распутывая и освобождая ее. Тут важно не путать:

* профессионализм с жесткостью и агрессивностью, приверженностью к какому-либо одному способу работы, жестко фиксированному набору приемов, техник;
* функции конструктивного лидера, созидателя, с навязыванием фантазий и бесплодным формотворчеством;
* и т.д.

И вовремя пресекать свои собственные нежелательные "увлечения", не бояться порой непростых поисков нужных выразительных форм, акцентов и точек зрения на ситуацию.

Концепция точки зрения моделирования появилась при самом зарождении использования графовых нотаций для проектирования ПО, в конце 1960-х годов, в составе подхода SADT [[2.4]](http://www.intuit.ru/department/se/vismodtp/popup.lit.html#17). 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 разработке моделей. Позднее, при дальнейшем развитии структурного анализа (1970 - 1980-е годы), появились разные виды диаграмм (сущность-связь, потоков данных, состояний и переходов и т. д.), 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 [[2.5]](http://www.intuit.ru/department/se/vismodtp/popup.lit.html#18) была в явном виде сформулирована идея использования разных точек зрения при объектно-ориентированном моделировании. 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 заключается в балансе между целостностью, единством языка (и создаваемых на его основе моделях) и возможностью отражать разные аспекты системы с помощью разных типов диаграмм. Однако "поймать" такой баланс непросто…

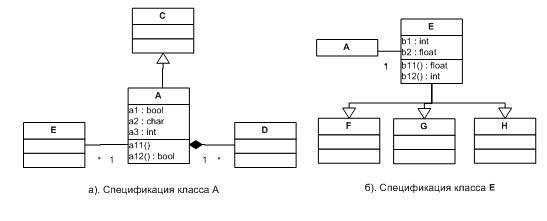
### 4.2.2. Граф модели, диаграммы и операции над ними

Визуальные модели создаются не с помощью карандаша и бумаги, а в специальных программных пакетах (например, CASE-пакетах). Это удобно, но, с другой стороны, усложняет структуру моделей и вводит новые правила работы с ними.

Визуальные спецификации обычно разделяют на граф модели и диаграммы. Граф модели - это набор сущностей визуальной модели, их атрибутов и связей. Диаграмма - это внешнее представление модели: геометрические размеры сущностей, их координаты, цвета, шрифты надписей, толщина линий и пр. Графические редакторы позволяют менять эти и многие другие параметры, делая диаграмму максимально удобной для работы. При этом граф модели остается неизменным.

Это разделение проходит красной нитью через средства визуального моделирования, отражаясь в строении визуальных языков, в интерфейсе и внутренней архитектуре программных инструментов и т. д.

Рассмотрим пример.

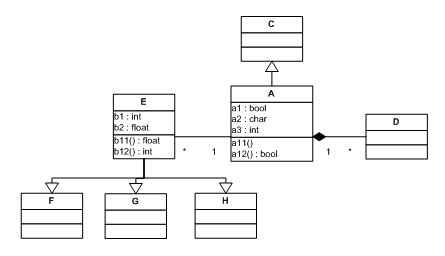


**Рис. 4.2-5.**  Пример двух разных, но "пересекающихся" по информации диаграмм

На [рис. 4.2-5](http://www.intuit.ru/department/se/vismodtp/2/3.html#image.2.5) а показана диаграмма классов, где приведена полная спецификация класса А - всех его атрибутов, операций, всех его предков в иерархии наследования, а также связей с другими классами. На [рис. 4.2-5](http://www.intuit.ru/department/se/vismodtp/2/3.html#image.2.5)б представлена диаграмма классов, где аналогично определяется класс E. Классы A и E связаны друг с другом ассоциацией, поэтому будут присутствовать на обеих диаграммах. Очевидно, что на этих диаграммах имеется общая информация. А теперь допустим, что изменилось имя класса E на [рис. 4.2-5](http://www.intuit.ru/department/se/vismodtp/2/3.html#image.2.5)б. Очевидно, что и на диаграмме с [рис. 4.2-5](http://www.intuit.ru/department/se/vismodtp/2/3.html#image.2.5) а это имя тоже должно измениться. Поскольку обе диаграммы представляют один и тот же граф модели, то при первом переименовании второе должно произойти автоматически.

И это еще простой пример. А диаграммы могут принадлежать разным типам и все равно быть связанными по информации. Например, в дополнение к диаграммам класса с [рис. 4.2-5](http://www.intuit.ru/department/se/vismodtp/2/3.html#image.2.5) можно создать диаграмму объектов, на которой будет присутствовать объект класса Е. При изменении имени класса Е диаграмма объектов должна также измениться, поскольку имя класса Е указано в имени объекта. Наконец, есть модельная информация, которая вовсе не отображается на диаграммах, но тем не менее 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 информацию, которая связывает разные диаграммы в единое целое, и используется граф модели.

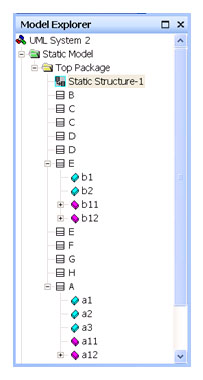
Полная модель для диаграмм с [рис. 4.2-5](http://www.intuit.ru/department/se/vismodtp/2/3.html#image.2.5) а и б представлен на [рис. 4.2-6](http://www.intuit.ru/department/se/vismodtp/2/3.html#image.2.6). Однако далеко не каждую модель удается полностью изобразить на одной диаграмме.



**Рис. 4.2-6.**  Полная модель для диаграмм с рис. 4.2-5

Диаграммы помогают создавать граф модели, а также просматривать и изменять его. Граф модели является хранилищем модельной информации, причем хранилищем "умным". Что это значит? Граф модели не есть склад "диаграмм". В таком случае класс А на [рис. 4.2-5](http://www.intuit.ru/department/se/vismodtp/2/3.html#image.2.5), а и тот же класс на [рис. 4.2-5](http://www.intuit.ru/department/se/vismodtp/2/3.html#image.2.5), б были бы разными сущностями. Граф модели хранит общий граф всех сущностей и связей, фрагменты которого отображаются на диаграммах. Если диаграммы модели сопоставить с файлами исходных текстов некоторой программы, то граф модели - это проанализированный компилятором единый текст этой программы, представленный в виде графа синтаксического разбора. Подобный анализ происходит при компиляции программ в исполняемый код, а в случае визуальных моделей он происходит раньше - при сохранении диаграмм в CASE-средстве.

Как правило, самым распространенным средством обзора графа модели является браузер модели. Такие браузеры есть в каждом CASE-пакете. Пример браузера модели для графа, представленного на [рис. 4.2-6](http://www.intuit.ru/department/se/vismodtp/2/3.html#image.2.6), показан на [рис..4.2-7](http://www.intuit.ru/department/se/vismodtp/2/3.html#image.2.7). На этом рисунке показаны все классы этой визуальной модели, а также пакеты, в которые они входят. Но в этом браузере не показаны отношения наследования и ассоциации, поскольку их неудобно представлять в таком виде - в дереве.



**Рис. 4.2-7.**  Пример браузера модели

В современных CASE-пакетах граф модели хранится в репозитории - едином хранилище модельной информации. В прежних CASE-пакетах репозиторий реализовывался как база данных. С ее помощью решались все вопросы с хранением графа модели, а также с доступом к нему. Несомненным плюсом такого подхода является решение вопросов многопользовательского и сетевого доступа[1)](http://www.intuit.ru/department/se/vismodtp/2/footnote.3.1.htm). Однако многопользовательский аспект не является в настоящее время ключевым, так как современные CASE-пакеты не являются средами разработки, как CASE-пакеты предыдущего поколения. Более важным оказываются вопросы быстродействия репозитория на больших моделях. Для решения этой задачи в современных CASE-пакетах часто применяются объектные базы в памяти, используются также специальные библиотеки для задания бизнес-объектов в памяти, например Eclipse/EMF. Долгосрочное хранение графа модели осуществляется в XML-формате.

Если бы в графе модели из представленного выше примера не было бы класса А, то его добавление на любую диаграмму возможно было бы только в режиме "добавить в граф модели". Но если такой класс уже существует в графе модели, а есть необходимость только отобразить его на очередной диаграмме, выполняется операция "загрузить на диаграмму". То есть если сначала был создан класс А на диаграмме с [рис. 4.2-5](http://www.intuit.ru/department/se/vismodtp/2/3.html#image.2.5), а подробно описаны все его атрибуты, а потом создается диаграмма на [рис. 4.2-5](http://www.intuit.ru/department/se/vismodtp/2/3.html#image.2.5), б, то на эту последнюю диаграмму класс А "загружается". При желании можно "загрузить" также все его атрибуты и методы, а также другие классы, которые с ним связаны. Разница между добавлением в граф модели и "загрузкой" на диаграмму должна быть очевидна: в обоих случаях элемент добавляется на диаграмму, но в первом случае он добавляется еще в граф модели, а во втором случае - нет. Во втором случае, наоборот, из модели берется вся необходимая информация о данном классе и отображается на диаграмме.

В CASE-пакетах операция "добавить в граф модели", доступная из диаграмм, совмещается с операцией "загрузить на диаграмму": при добавлении нового элемента на диаграмму он автоматически добавляется в граф модели.

Если элемент уже есть на диаграмме, его можно туда добавить еще раз, используя операцию "загрузить на диаграмму". Такая возможность часто используется для уменьшения количества пересечения связей на диаграммах.

К этим операциям есть пара двойственных им - "удалить из графа модели" и "выгрузить с диаграммы". Их смысл очевиден. На практике важно их не путать.

Все перечисленные выше операции выполнялись через диаграммы. Но, как правило, можно удалить/добавить элемент в граф модели и помимо диаграмм, в браузере модели. Это можно также делать программно, через скрипт или приложение, которое обращается к репозиторию через программный интерфейс. В таком случае, если удален элемент из графа модели, то CASE-пакет должен обеспечить его автоматическое "исчезновение" со всех диаграмм. При добавлении элемента в граф модели через браузер такой элемент, вообще говоря, не обязан появляться на какой-либо диаграмме.

### 4.2.3. Контрольные вопросы по теме «Иерархия мета описаний. Моделирование. Граф модели и диаграммы»

1. Что такое предметная область, модель, метамодель и мета модель?
2. Что является предметной областью для моделей ПО?
3. Что такое модели анализа?
4. Что такое модели проектирования, чем они отличаются от моделей анализа?
5. Что является предметной областью для метамоделей ПО? А для метаметамоделей?
6. Почему в случае визуального моделирования нам хватает четырех метауровней? Дайте два варианта ответа - принципиальный и следующий из способа описания метамоделей.
7. Вообразите и опишите ситуацию, когда здесь вам понадобился пятый уровень.
8. Приведите свой пример для четырех мета уровней. Сравните его с примером из лекций.
9. Чем является UML: (i) предметной областью, (ii) моделью (iii) метамоделью (iv) мета моделью? Ответ обоснуйте.
10. Что такое точка зрения моделирования? Расскажите подробно о важнейших составляющих в ее определении.
11. С чем связано использование множественности точек зрения при визуальном моделировании ПО?
12. Опишите точку зрения моделей анализа.
13. Опишите точку зрения моделей проектирования.
14. Как вы поняли практический прием по учету целевой аудитории моделирования. Собираетесь ли вы использовать его на практике?
15. Зачем для визуальных моделей выделять граф модели и диаграммы?
16. Что такое браузер модели и зачем он нужен?
17. Расскажите об операциях над графом модели.
18. Расскажите об операциях над диаграммами.
19. Расскажите о сочетании операций над диаграммами с операциями над графом модели.
20. Что такое репозиторий CASE-пакета? Расскажите о способах его реализации.
21. Расскажите об операциях над графом модели через браузер и средствами стороннего приложения (через открытый программный интерфейс). Что при этом происходит (должно происходить) с диаграммами?

### 4.2.4. Тестовые задания по теме «Иерархия мета описаний. Моделирование. Граф модели и диаграммы»