# ЛЕКЦИЯ 4. RATIONAL ROSE И UML

4.1. Введение

В создании программ, как и в любом деле, важное место занимает творчество. Однако требования, предъявляемые к современным программным продуктам, заставляют ставить процесс их разработки на индустриальную основу. Необходимы инструменты, которые позволят уменьшать долю ручного труда при программировании и предоставят лучшие возможности для творчества, освобождая программиста от выполнения рутинных операций и от ошибок.

Таким инструментом является CASE (Computer-Aided Software/System Engineering) - средство Rational Rose Enterprise Edition. Его можно использовать как отдельно, так и вместе с другими продуктами компании Rational Software, что позволяет создавать сложные программные системы быстрее, качественнее и легче. Неслучайно Rational Rose включается во все конфигурации Rational Suite – инструмента для аналитиков, программистов и тестировщиков.

Пользуясь Rational Rose, проектировщик может создать не абстрактное словесное описание системы, а его конкретную модель, которая затем дополняется описанием классов на языке программирования. Rational Rose поддерживает проектирование, основанное на двух способах: прямом и обратном. В первом режиме разработчик строит диаграммы классов и их взаимодействия, а на выходе получает сгенерированный код. Во втором режиме возможно построение модели на базе имеющегося исходного кода. Отсюда следует главная возможность для разработчика: повторное проектирование. Программист описывает классы в Rational Rose, генерирует код, вносит изменения в модель и снова пропускает ее через Rational Rose для получения обновленного результата.

Моделирование позволяет значительно сократить время разработки, уложиться в бюджет и создать систему требуемого качества. Модели в виде UML (Unified Modeling Language) диаграмм, созданные в среде Rational Rose, имеют целый ряд замечательных особенностей. Они удобны для понимания алгоритмов работы, взаимосвязей между объектами системы и ее поведения в целом, а также позволяют непосредственно из проекта автоматически построить исходный текст программы на одном из языков, поддерживаемых Rational Rose.

Преимущества от применения Rational Rose:

* сокращение цикла разработки приложения “заказчик-программист-заказчик”;
* увеличение продуктивности работы программиста;
* улучшение потребительских качеств создаваемых программ за счет ориентации на пользователей и бизнес;
* способность вести большие проекты и группы проектов;
* возможность повторного использования уже созданного программного обеспечения за счет упора на разбор их архитектуры и компонентов.

4.2. Case-средство Rational Rose

Rational Rose – это объектно-ориентированное средство автоматизированного проектирования программных систем. В его основе лежит CASE-технология, комплексный подход и использование единой унифицированной нотации на всех этапах жизненного цикла создания ПС.

Для работы с Rational Rose необходим UML – графический язык описания архитектуры системы. Программы на UML представляются в виде диаграмм, состоящих из объектов и связей между ними или из этапов процесса проектирования. Графические возможности продукта позволяют решать задачи, связанные с проектированием, на различных уровнях абстракции: от общей модели процессов предприятия до конкретной модели класса в создаваемом программном обеспечении. В среде Rational Rose проектировщик и программист работают в тандеме. Первый создает логическую модель системы, а второй дополняет ее моделями классов на конкретном языке программирования. В настоящее время Rational Rose обеспечивает генерацию кода по модели на ряде языков программирования: Microsoft Visual С++, Ada, Java, Visual Basic, CORBA, XML, COM, Oracle. Кроме того, есть специальные мосты к не входящим в стандартную поставку языкам, например к Delphi, Assembler.

4.3. Работа в среде Rational Rose

После запуска системы открывается ее главное окно, показанное на рис. 4.4. В верхней части экрана находится меню и стандартная панель инструментов. Она видна всегда, и ее кнопки соответствуют командам, которые могут использоваться для работы с любой диаграммой. Слева расположено окно Browser, представляющее собой иерархическую структуру и предназначенное для выполнения ряда действий:

* просмотр и добавление элементов к модели;
* просмотр существующих отношений между элементами модели;
* перемещение и переименование элементов модели;
* добавление элементов модели к диаграмме;
* связывание элемента с файлом или адресом Интернета;
* группирование элементов в пакеты;
* работа с детализированной спецификацией элемента;
* открытие диаграммы.

Каждый объект в Rational Rose имеет свое контекстное меню, посредством которого изменяются свойства и выполняются действия над объектом.

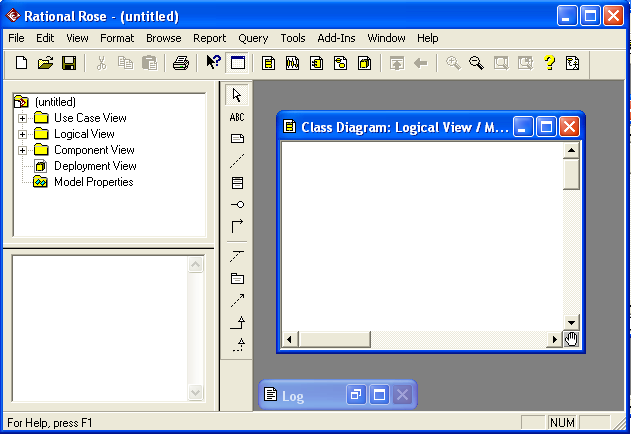


Рис. 4.1 Главное окно Rational Rose

Под Browser находится окно Documentation, предназначенное для документирования элементов модели Rational Rose. При описании класса вся информация из этого окна появится затем как комментарий в сгенерированном коде и в отчетах, создаваемых в среде Rational Rose. В случае смены активного элемента содержание Documentation автоматически обновляется.

В правой части экрана, называемой рабочим столом Rational Rose, находятся открытые в данный момент диаграммы. При создании новой модели на рабочем столе открывается Class Diagram (диаграмма классов). Окна Browser и Diagram разделены строкой инструментов, которая изменяется в зависимости от типа активной диаграммы. Внизу рабочего стола видно свернутое окно Log (протокол), в котором фиксируются все действия, выполненные над диаграммами, также туда попадают сообщения об ошибках, произошедших в течение работы. Информация заносится в окно Log независимо от того, свернуто оно или вообще закрыто.

При изменении диаграмм в области рабочего стола Rational Rose автоматически обновит структуру Browser. Аналогично при внесении изменений в элемент с помощью Browser Rational Rose автоматически обновит соответствующие диаграммы. Это помогает поддерживать модель в непротиворечивом состоянии.

4.4 Меню и диаграммы Rational Rose

Рассмотрим состав и назначение пунктов меню главного окна Rational Rose, приведенного на рис. 4.4.

* *File* используется для сохранения, загрузки, обновления проекта, печати диаграмм и дополнительных настроек.
* *Edit* предназначен для копирования и восстановления данных, а также для редактирования свойств и стилей объектов.
* *View* применяется для настройки представления окон меню и строк инструментов.
* *Format* позволяет изменять параметры отображения объектов, такие как шрифт, заливку, формат линий и т.д.
* *Browse* предназначен для навигации между диаграммами и спецификациями диаграмм, представленных в модели.
* *Report* предназначен для получения различного вида справок и отчетов.
* *Query* предоставляет возможности контролировать, какие элементы модели будут показаны на текущей диаграмме, и выполнять различные манипуляции с объектами диаграмм: скрывать, добавлять, фильтровать.
* *Tools* предоставляет доступ к различным дополнительным инструментам и подключаемым модулям.
* *Add-Ins* предоставляет доступ к менеджеру подключаемых модулей.
* *Window* позволяет управлять окнами на рабочем столе.
* *Help* позволяет получать справочную информацию.

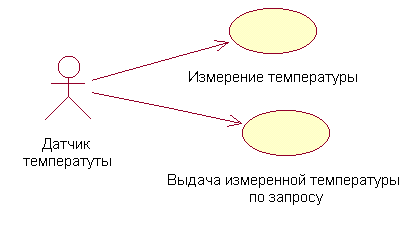
Все перечисленные возможности среды Rational Rose служат для моделирования прикладной программной системы в виде совокупности диаграмм на основе графических средств языка UML.

4.5. ВИДЫ ДИАГРАММ В RATIONAL ROSE

В распоряжение проектировщика системы Rational Rose предоставляет следующие типы диаграмм, последовательное создание которых позволяет получить полное представление о всей проектируемой системе и об отдельных ее компонентах :

* Use case diagram (диаграммы прецедентов);
* Deployment diagram (диаграммы топологии);
* Statechart diagram (диаграммы состояний);
* Activity diagram (диаграммы активности);
* Interaction diagram (диаграммы взаимодействия);
* Sequence diagram (диаграммы последовательностей действий);
* Collaboration diagram (диаграммы сотрудничества);
* Class diagram (диаграммы классов);
* Component diagram (диаграммы компонент).

## 4.5.1. Use case diagram (диаграммы прецедентов)

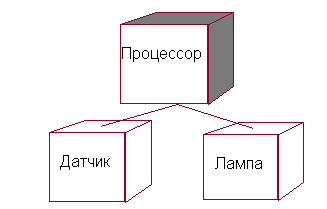


Этот вид диаграмм позволяет создать список операций, которые выполняет система. Часто этот вид диаграмм называют диаграммой функций, потому что на основе набора таких диаграмм создается список требований к системе и определяется множество выполняемых системой функций.

Каждая такая диаграмма или, как ее обычно называют, каждый Use case – это описание сценария поведения, которому следуют действующие лица (Actors).

Данный тип диаграмм используется при описании бизнес процессов автоматизируемой предметной области, определении требований к будущей программной системе. Отражает объекты как системы, так и предметной области и задачи, ими выполняемые.

## 4.5.2. Deployment diagram (диаграммы топологии)



Этот вид диаграмм предназначен для анализа аппаратной части системы, то есть «железа», а не программ. В прямом переводе с английского Deployment означает «развертывание», но термин «топология» точнее отражает сущность этого типа диаграмм.

Для каждой модели создается только одна такая диаграмма, отображающая процессоры (Processor), устройства (Device) и их соединения.

Обычно этот тип диаграмм используется в самом начале проектирования системы для анализа аппаратных средств, на которых она будет эксплуатироваться.

## 4.5.3. State Maсhine diagram (диаграммы состояний)

Каждый объект системы, обладающий определенным поведением, может находится в определенных состояниях, переходить из состояния в состояние, совершая определенные действия в процессе реализации сценария поведения объекта. Поведение большинства объектов реальных систем можно представить с точки зрения теории конечных автоматов, то есть поведение объекта отражается в его состояниях, и данный тип диаграмм позволяет отразить это графически. Для этого используется два вида диаграмм: Statechart diagram (диаграмма состояний) и Activity diagram (диаграмма активности)

4.5.3.1. Statechart diagram (диаграмма состояний)

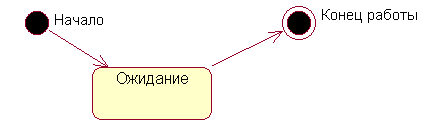
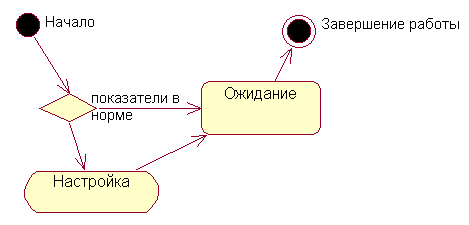


Диаграмма состояний (Statechart) предназначена для отображения состояний объектов системы, имеющих сложную модель поведения. Это одна из двух диаграмм State Machine, доступ к которой осуществляется из одного пункта меню.

4.5.3.2. Activity diagram (диаграммы активности)



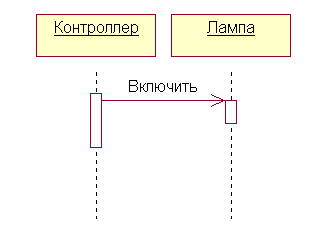
Это дальнейшее развитие диаграммы состояний. Фактически данный тип диаграмм может использоваться и для отражения состояний моделируемого объекта, однако, основное назначение Activity diagram в том, чтобы отражать бизнес-процессы объекта. Этот тип диаграмм позволяет показать не только последовательность процессов, но и ветвление и даже синхронизацию процессов.

Этот тип диаграмм позволяет проектировать алгоритмы поведения объектов любой сложности, в том числе может использоваться для составления блок-схем.

## 4.5.4. Interaction diagram (диаграммы взаимодействия)

Этот тип диаграмм включает в себя диаграммы Sequence diagram (диаграммы последовательностей действий) и Collaboration diagram (диаграммы сотрудничества). Эти диаграммы позволяют с разных точек зрения рассмотреть взаимодействие объектов в создаваемой системе.

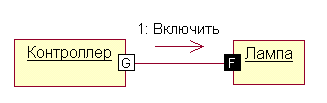
## 4.5.4.1. Sequence diagram (диаграммы последовательностей действий)

 Взаимодействие объектов в системе происходит посредством приема и передачи сообщений объектами-клиентами и обработки этих сообщений объектами-серверами. При этом в разных ситуациях одни и те же объекты могут выступать и в качестве клиентов, и в качестве серверов.

Данный тип диаграмм позволяет отразить последовательность передачи сообщений между объектами.

Этот тип диаграммы не акцентирует внимание на конкретном взаимодействии, главный акцент уделяется последовательности приема/передачи сообщений. Для того чтобы окинуть взглядом все взаимосвязи объектов, служит Collaboration diagram.

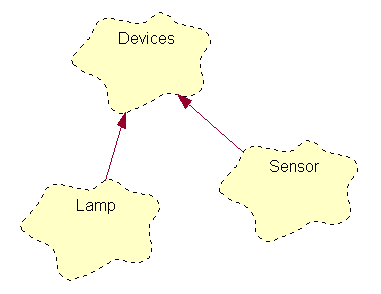
## 4.5.4.2. Collaboration diagram (диаграммы сотрудничества)



Этот тип диаграмм позволяет описать взаимодействия объектов, абстрагируясь от последовательности передачи сообщений. На этом типе диаграмм в компактном виде отражаются все принимаемые и передаваемые сообщения конкретного объекта и типы этих сообщений.

По причине того, что диаграммы Sequence и Collaboration являются разными взглядами на одни и те же процессы, Rational Rose позволяет создавать из Sequence диаграммы диаграмму Collaboration и наоборот, а также производит автоматическую синхронизацию этих диаграмм.

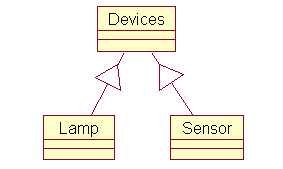
## 4.5.5. Class diagram (диаграммы классов)



Этот тип диаграмм позволяет создавать логическое представление системы, на основе которого создается исходный код описанных классов.

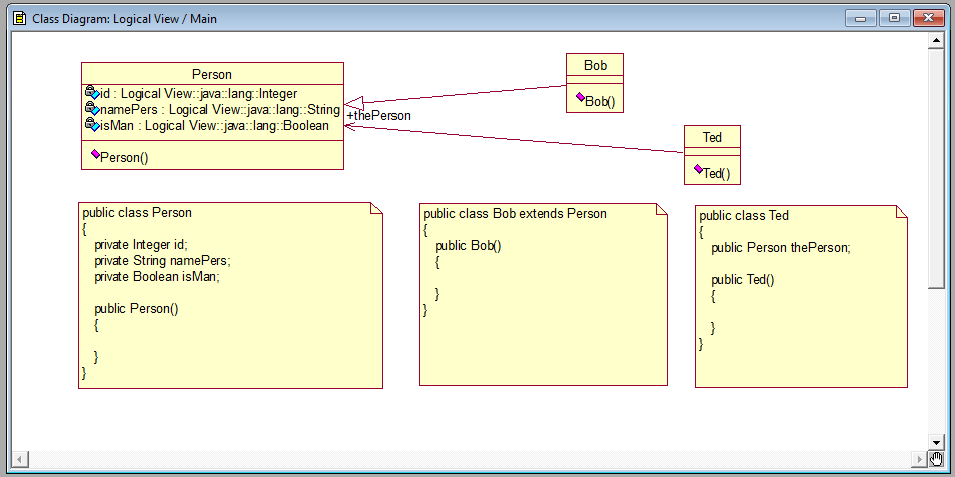
Значки диаграммы позволяют отображать сложную иерархию систем, взаимосвязи классов (Classes) и интерфейсов (Interfaces). Данный тип диаграмм противоположен по содержанию диаграмме Collaboration, на котором отображаются объекты системы.

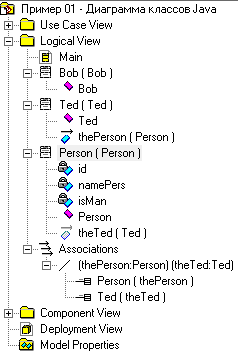
Rational Rose позволяет создавать классы при помощи данного типа диаграмм в различных нотациях.

В нотации, предложенной Г. Бучем, которая так и называется Booch, классы изображаются в виде чего-то нечеткого, похожего на облако. Таким образом Г.Буч пытается показать, что класс – это лишь шаблон, по которому в дальнейшем будет создан конкретный объект.

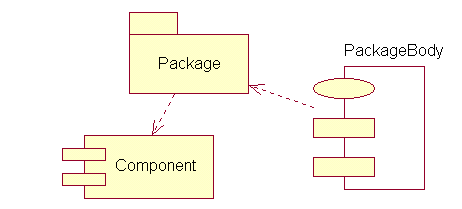
Нотация OMT, более строга. И конечно же, Rational Rose позволяет создавать диаграмму классов в унифицированной нотации.

## 4.5.5.1 Пример диаграммы классов для Java





## 4.5.6. Component diagram (диаграммы компонентов)

 Этот тип диаграмм предназначен для распределения классов и объектов по компонентам при физическом проектировании системы. Часто данный тип диаграмм называют диаграммами модулей.

При проектировании больших систем может оказаться, что система должна быть разложена на несколько сотен или даже тысяч компонентов, и этот тип диаграмм позволяет не потеряться в обилии модулей и их связей.