Rational Rose и UML

Проектирование объектно-ориентированных приложений

Методы проектирования ИС

Объектно-ориентированное проектирование

Методологии:

- UML

Структурное проектирование

Методологии:

- IDEF0
- DFD
- IDEF3

Проблемы метода структурного проектирования

- Функциональную точку зрения трудно развивать
- Реальные системы трудно охарактеризовать функционально
- Фокусирование на функциональности теряет из виду данные
- Функциональная ориентация производит код, менее пригодный для многократного использования

ИТОГ:

- Мейер Бертран
  - *«Нисходящее функциональное проектирование плохо адаптируется к разработке крупных программных систем. Нисходящее проектирование остается полезной парадигмой для малых программ и индивидуальных алгоритмов..., но оно практически не масштабируется на большие системы. Смысл не в том, что Вы не можете разрабатывать систему сверху вниз: можете. Но, выторговывая для себя краткосрочное удобство за длительную негибкость, Вы некорректно нагромождаете одну функцию над другой и (достаточно часто) функциональный интерфейс над более важными параметрами системы. Вы теряете из виду аспект данных, и Вы жертвуете возможностью многократного использования».*

Объектно-ориентированное проектирование

- Мейер:
  - *«Объектно-ориентированное проектирование - конструирование программных систем в виде структурированных коллекций, реализующих абстрактные типы данных".*

- Неформально он определяет это как
  - *“Метод, который ведет к архитектурам программ, основанным на объектах, используемых системой или подсистемой (предпочтительнее чем "функция", которую система, как предполагается, выполняет)".*

Принципы объектно-ориентированного проектирования

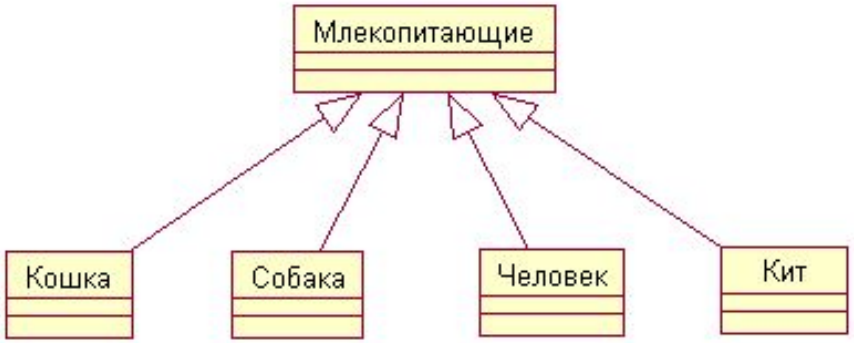
ИНКАПСУЛЯЦИЯ

- Инкапсуляция – подобна понятию сокрытия информации.
- Это возможность скрывать многочисленные детали объекта от внешнего мира.
- Внешним миром объекта является все то, что находится вне объекта, включая остальную часть системы.

Принципы объектно-ориентированного проектирования

НАСЛЕДОВАНИЕ

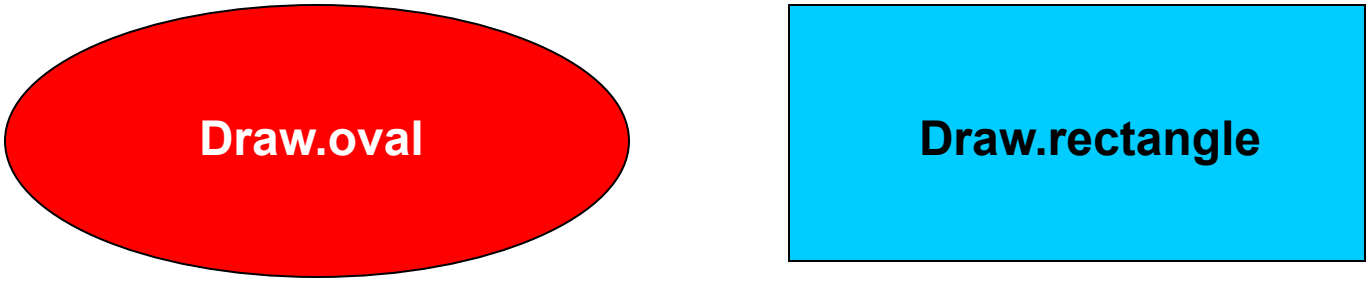
- Механизм, позволяющий создавать новые объекты, основываясь на уже существующих. Порождаемый или дочерний объект-потомок наследует свойства порождающего, родительского объекта.



Принципы объектно-ориентированного проектирования

ПОЛИМОРФИЗМ

- Полиморфизм означает наличие множества форм или реализаций конкретной функциональности.



Почему объектная ориентация работает???

- Объектная ориентация работает на более высоком уровне абстракции.
- Данные, на которых базируется система более стабильны, нежели функциональные возможности, которые эта система поддерживает.
- Объектно-ориентированное проектирование и программирование поддерживает многократное использование кода.

Уровни представления модели

- Уровень представления – концептуальный, логический и физический.

Обратно

Язык UML – унифицированный язык моделирования

- 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.

- Текущая спецификация UML 2.1.1

- Доступ можно получить на сайте группы Object Management Group по адресу http://www.omg.org/

Канонические диаграммы языка UML

- *Диаграмма (diagram)* — графическое представление совокупности элементов *модели* в форме связного графа, вершинам и ребрам (дугам) которого приписывается определенная семантика.

Диаграммы - представления

- Функциональное представление
  - Диаграммы вариантов использования
  - Диаграммы кооперации
  - Диаграммы последовательности

  Логическое представление
  - Диаграммы классов
- Представление компонентов
  - Диаграммы компонентов
- Представление размещения
  - Диаграммы размещения

Rational Rose

Enterprise Edition 2007

Rational Rose

- Rational Rose – это мощный инструмент анализа и проектирования объектно-ориентированных программных систем, позволяющий моделировать системы до написания кода, так чтобы с самого начала быть уверенными в адекватности их архитектуры.

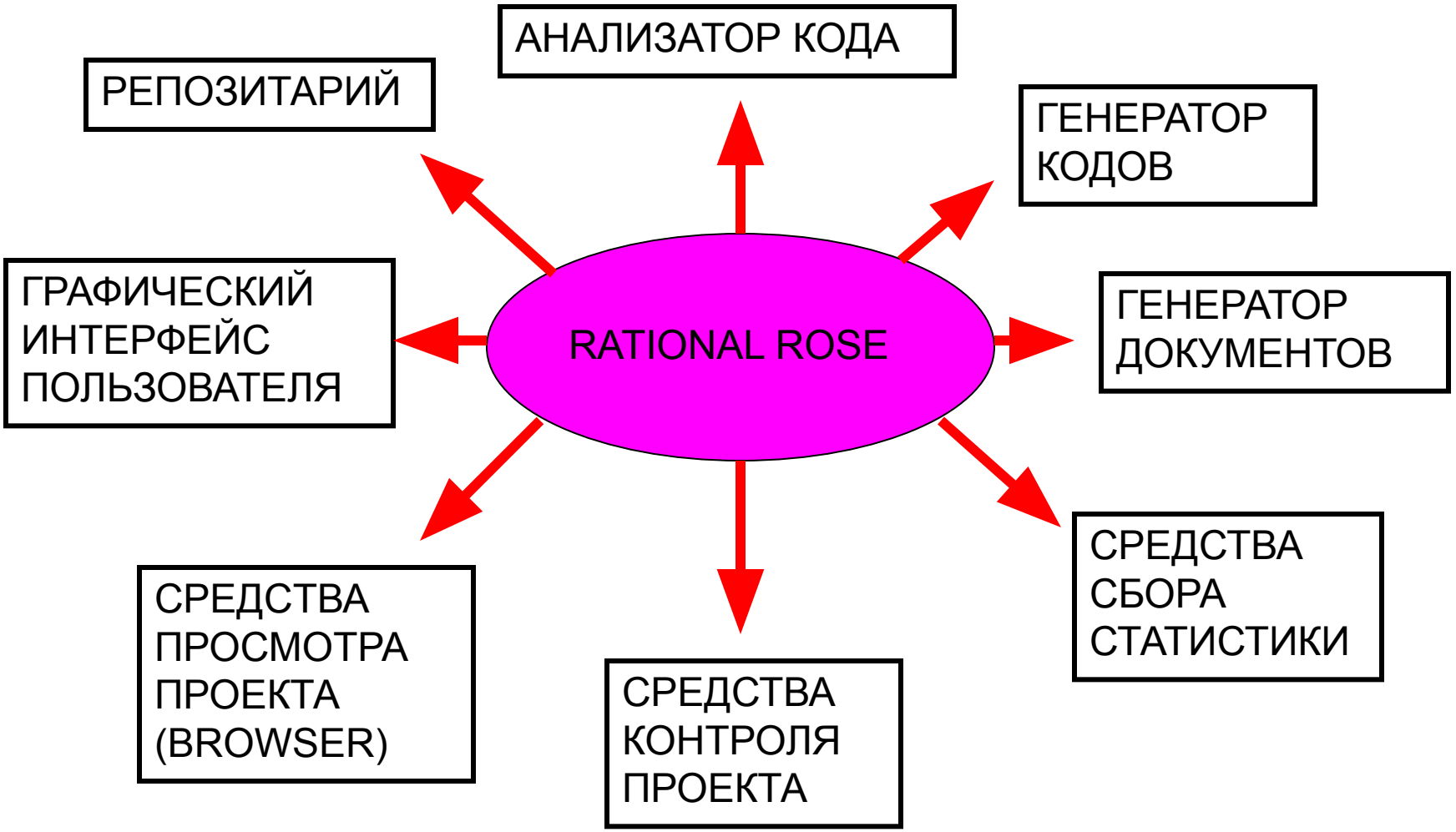
- Модель Rose – это картина системы. Она содержит все диаграммы UML, действующих лиц, варианты использования, объекты, классы, компоненты и узлы системы.

- Rose – это средство, которое может быть использовано всеми участниками проекта.

Продукты IMB Rational Rose

- Rational Rose Developer for Java – это комплексная среда визуального моделирования на основе языка Unified Modeling Language (UML). Эта среда поддерживает генерацию кода для моделей Java и J2EE.
- Rational Rose Developer for UNIX - Средство разработки на основе моделей, занимающее лидирующее положение в отрасли.
- Rational Rose Enterprise – это один из наиболее комплексных продуктов семейства Rational Rose.
- Rational Rose Technical Developer - Основанное на использовании моделей решение по разработке ПО, обеспечивающее автоматизацию генерации кода программ на языках Java, C и C++.

Структура Rational Rose Enterprise

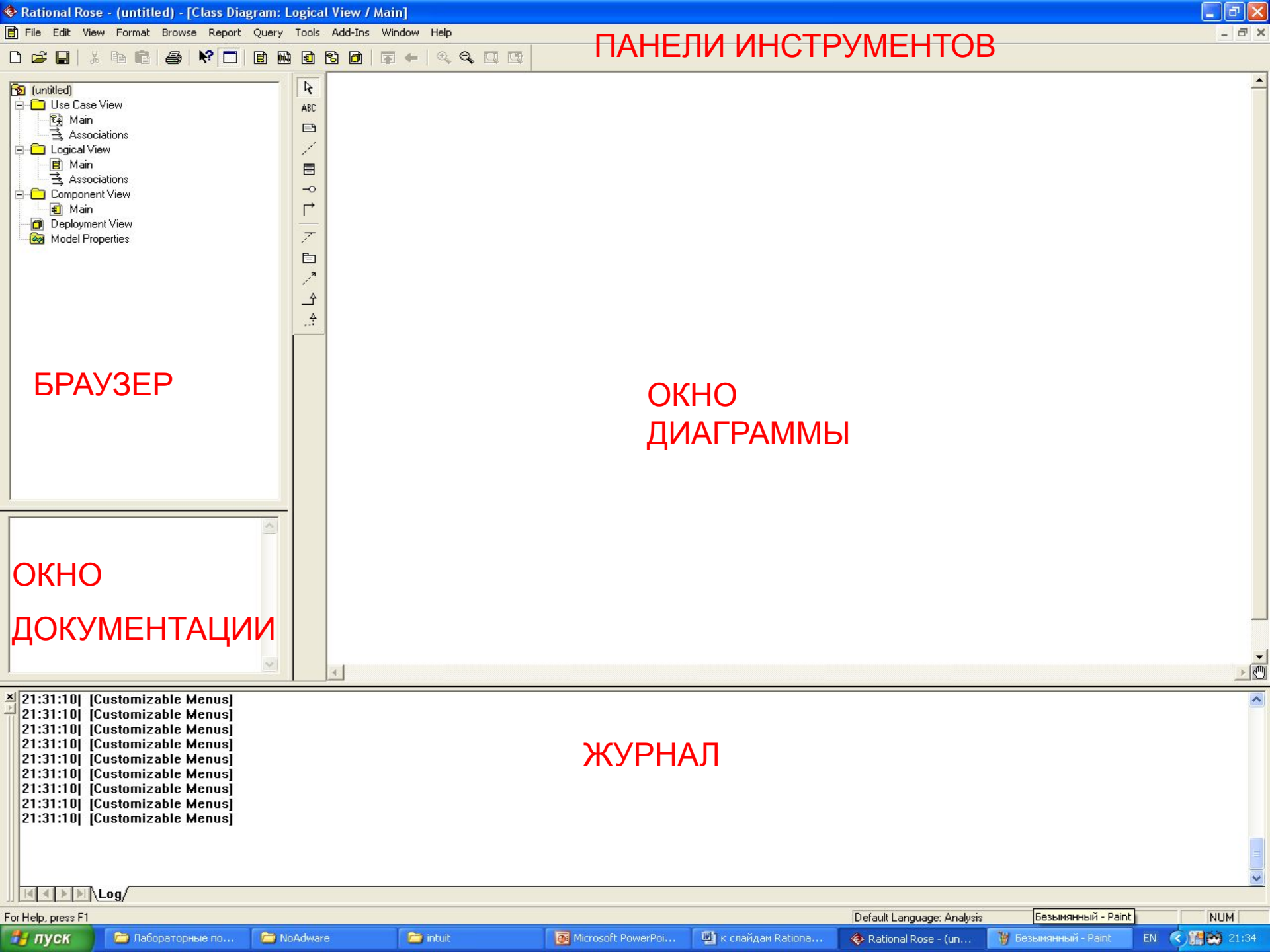


Дополнительные возможности Rational Rose Enterprise

- Новая возможность: поддержка прямого и обратного конструирования для наиболее распространенных конструкций Java 1.5.
- Генерация кода на языках Ada, ANSI C++, C++, CORBA, Java и Visual Basic с настраиваемой синхронизацией моделей и кода.
- Возможности анализа качества кода.
- Дополнительное встраиваемое средство Web-моделирования для визуализации, моделирования и разработки Web-приложений.
- UML-моделирование для разработки баз данных с возможностью представления интеграции данных и требований приложений на логической или физической основе.
- Возможность создания описаний типа документа (DTD) на языке XML для использования в приложениях.
- Интеграция с другими средствами разработки жизненного цикла IBM Rational.
- Возможность публикации моделей и отчетов в Интернете для облегчения процесса взаимодействия в распределенных группах разработчиков.

Изучение пакета Rational Rose Enterprise 2007

на примере создания модели системы управления банкоматом


Rational Rose - (untitled) - [Class Diagram: Logical View / Main]
File Edit View Format Browse Report Query Tools Add-Ins Window Help
ПАНЕЛИ ИНСТРУМЕНТОВ
(untitled)
Use Case View
Main
Associations
Logical View
Main
Associations
Component View
Main
Deployment View
Model Properties
БРАУЗЕР
ОКНО ДОКУМЕНТАЦИИ
ОКНО ДИАГРАММЫ
21:31:10| [Customizable Menus]
21:31:10| [Customizable Menus]
21:31:10| [Customizable Menus]
21:31:10| [Customizable Menus]
21:31:10| [Customizable Menus]
21:31:10| [Customizable Menus]
21:31:10| [Customizable Menus]
21:31:10| [Customizable Menus]
21:31:10| [Customizable Menus]
Log
ЖУРНАЛ
For Help, press F1
Default Language: Analysis
NUM
пуск
Лабораторные по...
NoAdware
intuit
Microsoft PowerPoi...
к слайдам Rationa...
Rational Rose - (un...
Безымянный - Paint
EN
21:34

Использование Rational Rose на начальной стадии проектирования системы

- Свойства системы исследуются на высоком уровне
- Некоторые задачи начальной фазы включают в себя определение вариантов использования и действующих лиц.
- Rose можно применять для документирования этих вариантов использования и действующих лиц, а также для создания диаграмм, показывающих связи между ними.
- Полученные диаграммы Вариантов использования можно показать пользователям, чтобы убедиться, что они дают достаточно полное представление о свойствах системы.

Использование Rational Rose в фазе уточнения

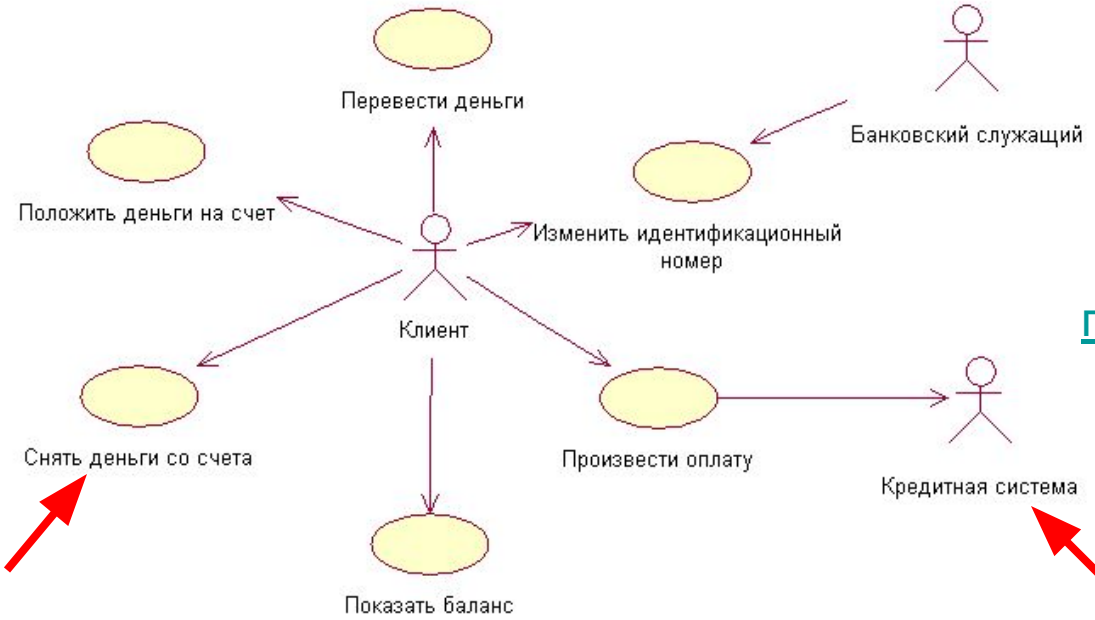
- Уточнение – это детализация требований к системе.
- Обновление модели Вариантов Использования
- Диаграммы Последовательности и Кооперации помогают проиллюстрировать поток обработки данных при его детализации. С их помощью можно спроектировать требуемые для системы объекты.
- Уточнение предполагает подготовку проекта к сдаче разработчикам, которые начнут ее конструирование
- В среде Rose это может быть выполнено путем создания диаграмм Классов и диаграмм Состояний

Использование Rational Rose в фазе конструирования

- В фазе конструирования пишется большая часть кода проекта.
- Чтобы показать зависимости между компонентами на этапе компиляции, создаются диаграммы Компонентов.
- После выбора языка программирования можно осуществить генерацию скелетного кода для каждого компонента.
- По завершении работы над кодом модель можно привести в соответствие с ним с помощью обратного проектирования.

Диаграмма вариантов использования (Use Case)

- Диаграммы использования описывают функциональность ИС, которая будет видна пользователям системы.
- Отображает взаимодействие между вариантами использования, представляющими функции системы, и действующими лицами, представляющими людей или системы, получающие или передающие информацию в данную систему.



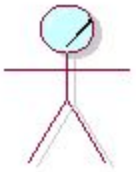
представление

Вариант использования

Актер = действующее лицо

Основные концепции моделирования вариантов использования

- Действующие лица – все , кто взаимодействует с разрабатываемой системой и находятся вне границ действия системы.

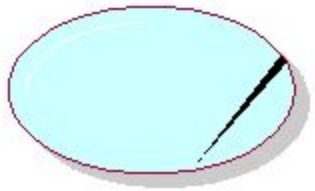


Клиент банкомата

- Типы действующих лиц:
  - Пользователи системы
  - Другие системы, взаимодействующие с данной
  - Время

Основные концепции моделирования вариантов использования

- Варианты использования – описание на высоком уровне функций, предоставляемых системой, т.е. они иллюстрируют, как можно использовать систему.

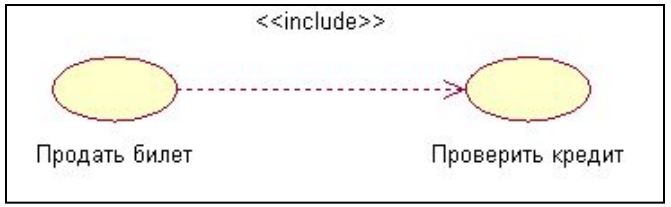


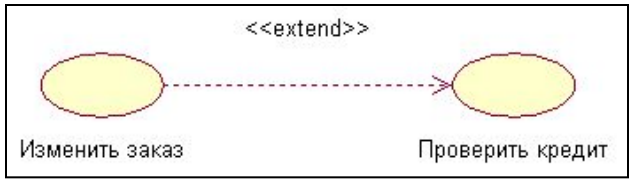
- Варианты использования не зависят от реализации

- Варианты использования дают высокоуровневую картину системы

Основные концепции моделирования вариантов использования

- Отношения
  - Между вариантом использования и действующим лицом
    - ассоциативные отношения
  - Между вариантами использования:
    - Включающие
    - Расширяющие и
    - Обобщенные
  - Между действующими лицами
    - обобщенные отношения





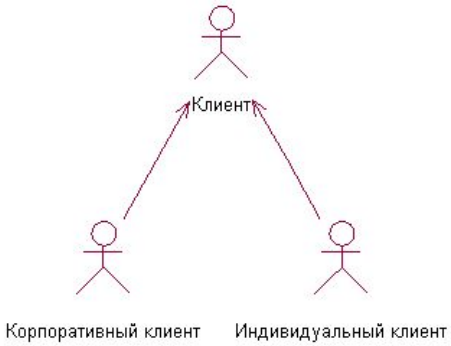
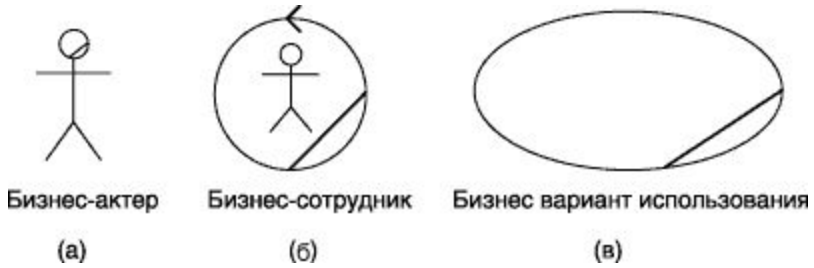


Диаграмма вариантов использования (Use Case)

Механизмы расширения



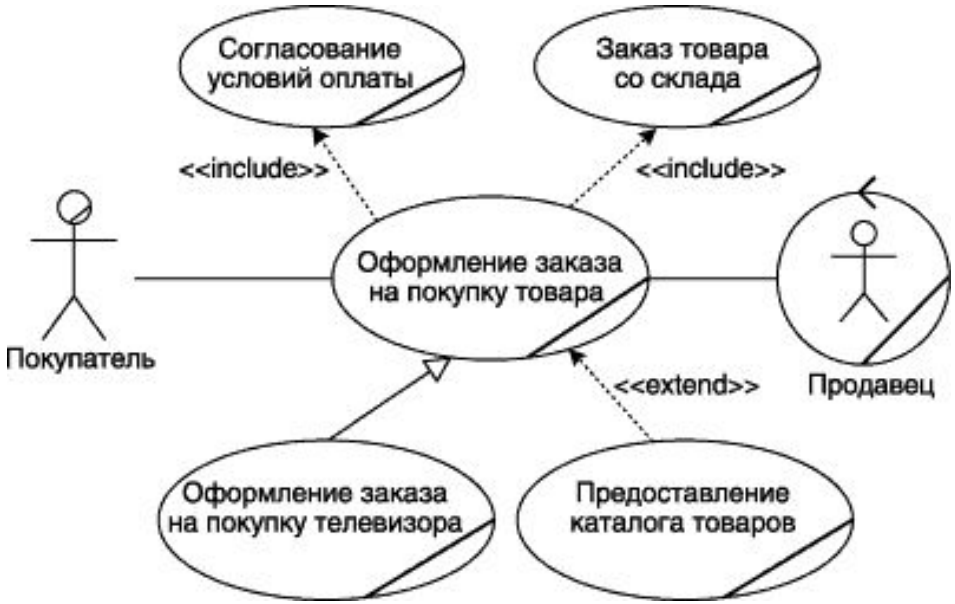
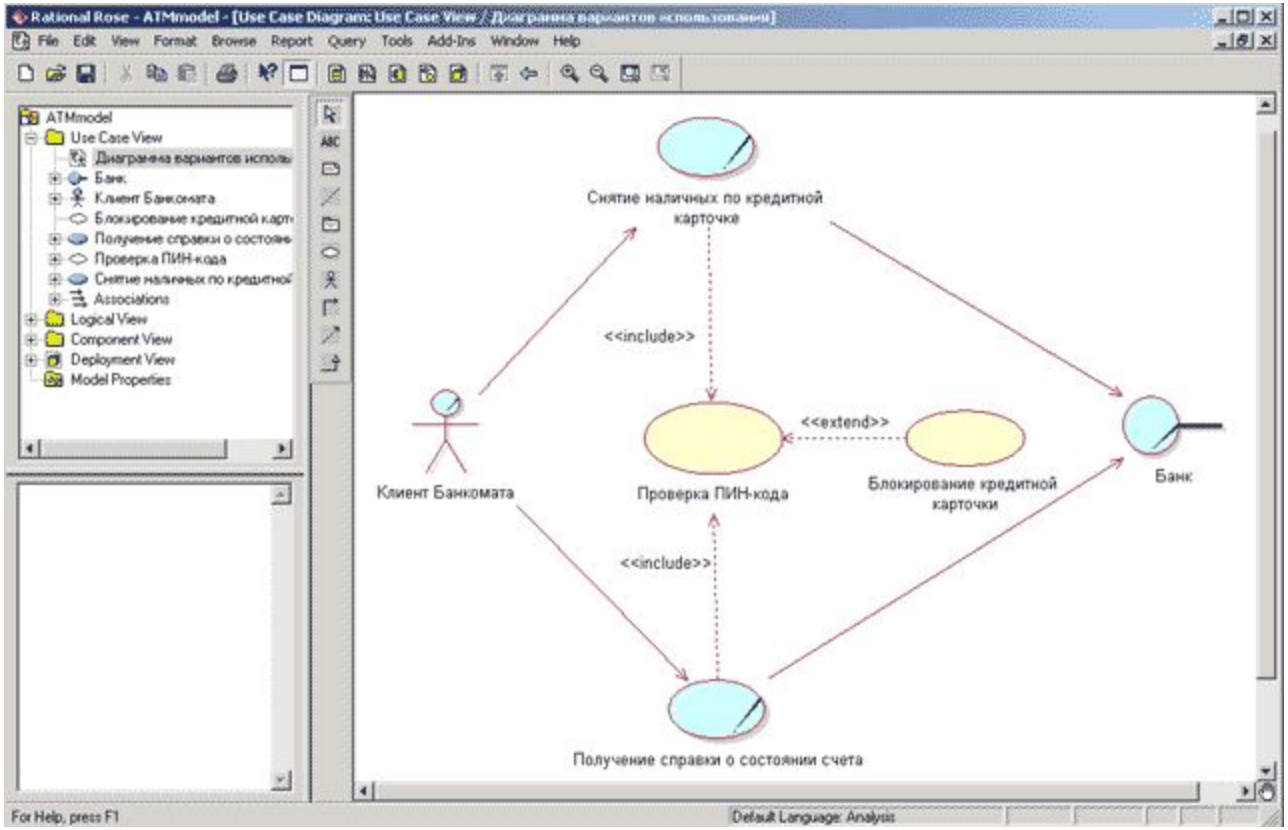


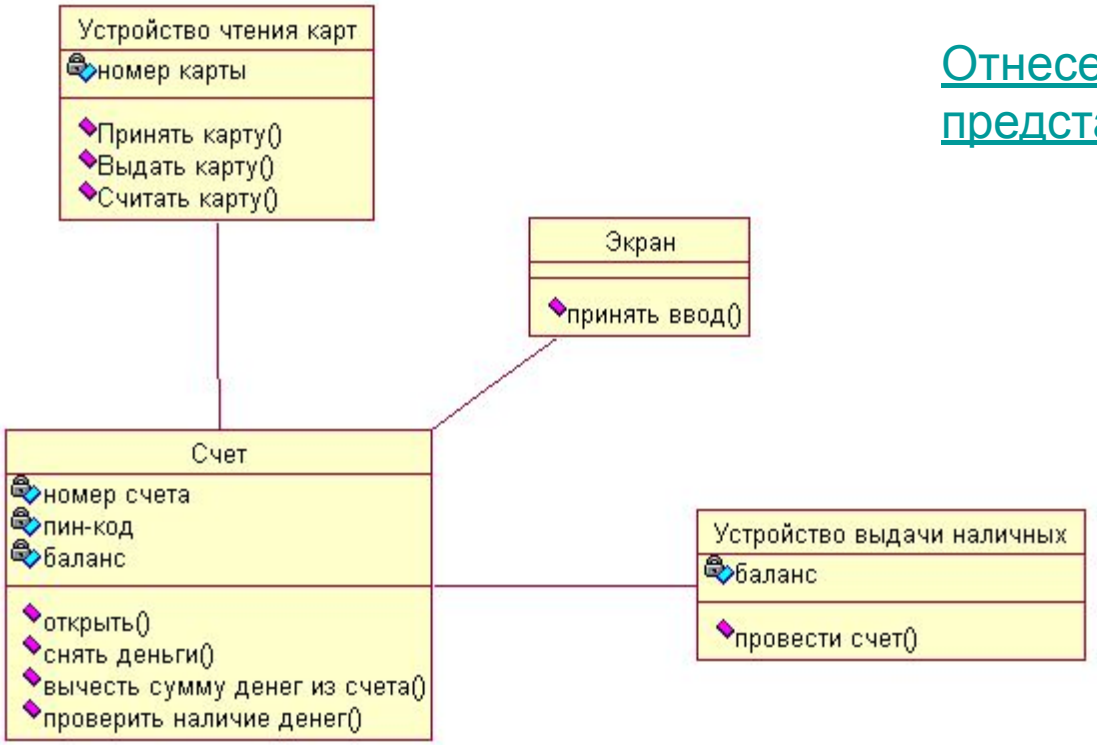
Диаграмма вариантов использования (Use Case)

- Самостоятельная работа:



Диаграммы классов

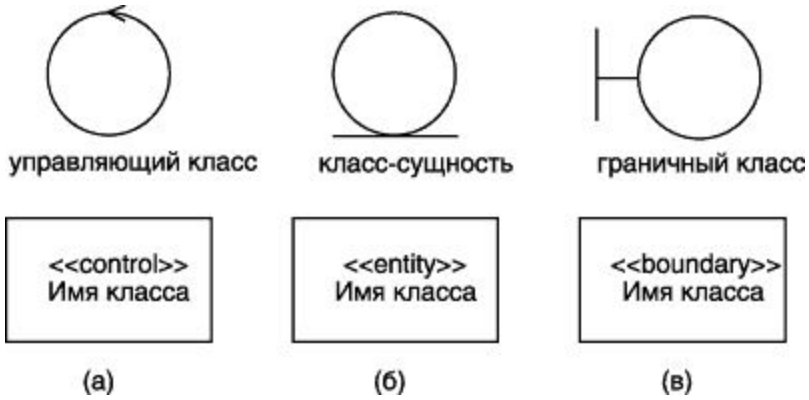
- Диаграмма *классов* (class diagram) — диаграмма языка UML, на которой представлена совокупность декларативных или статических элементов модели, таких как *классы* с *атрибутами* и *операциями*, а также связывающие их отношения.
- Разработчики используют диаграммы классов для реальной разработки классов. Rational Rose генерирует основу кода классов, которую программисты заполняют деталями на выбранном ими языке.
- Аналитики могут показать детали системы, а архитекторы – понять ее дизайн.



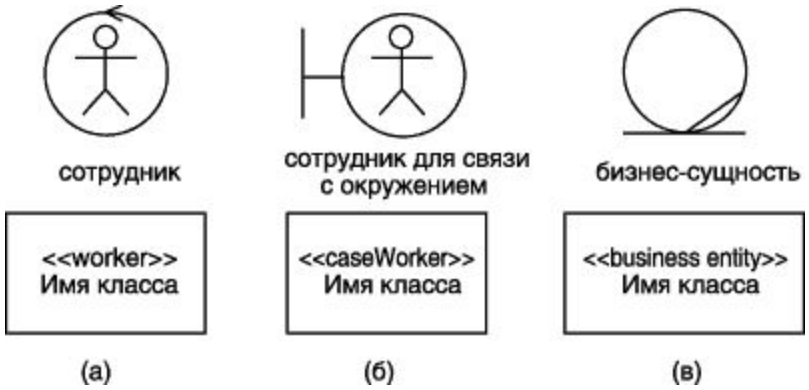
Отнесение к представлению

Механизмы расширения

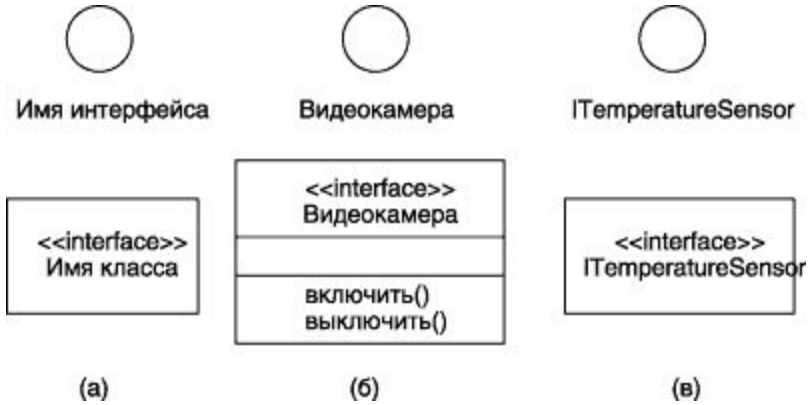
1. профиль для процесса разработки программного обеспечения



2. профиль для бизнес-моделирования

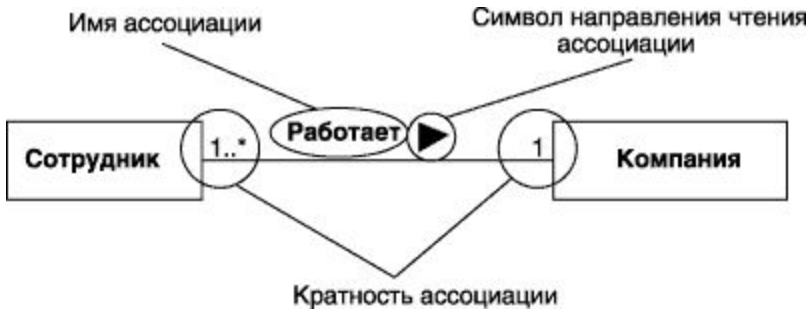


Интерфейс



Отношения на диаграмме классов

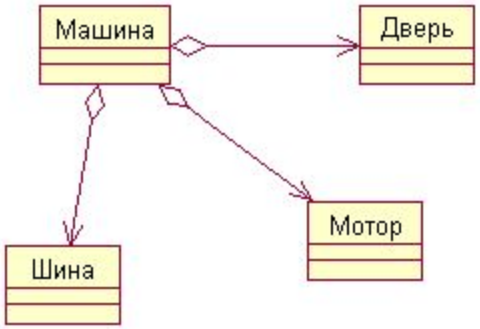
1. Отношение ассоциации



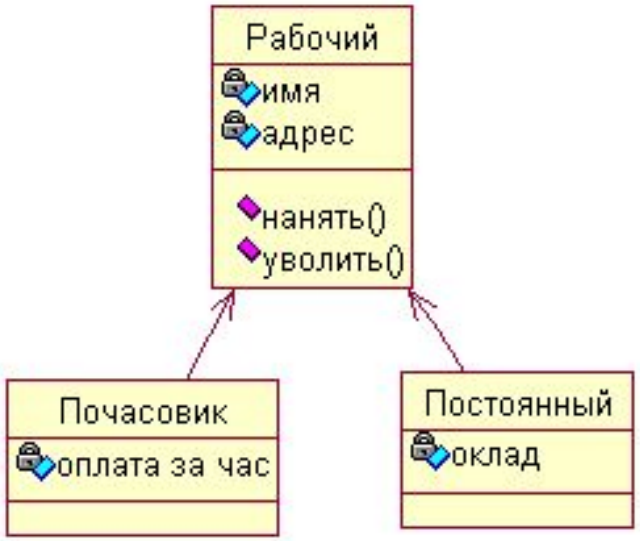
2. Отношение зависимости – показывают, что один класс ссылается на другой. Таким образом, изменения во втором классе повлияют на первый. При генерации кода для классов, между которыми установлены зависимости, Rose не добавляет к ним никаких новых атрибутов.

Отношения на диаграмме классов

3. Отношение агрегации – отношение между целым и его частями.

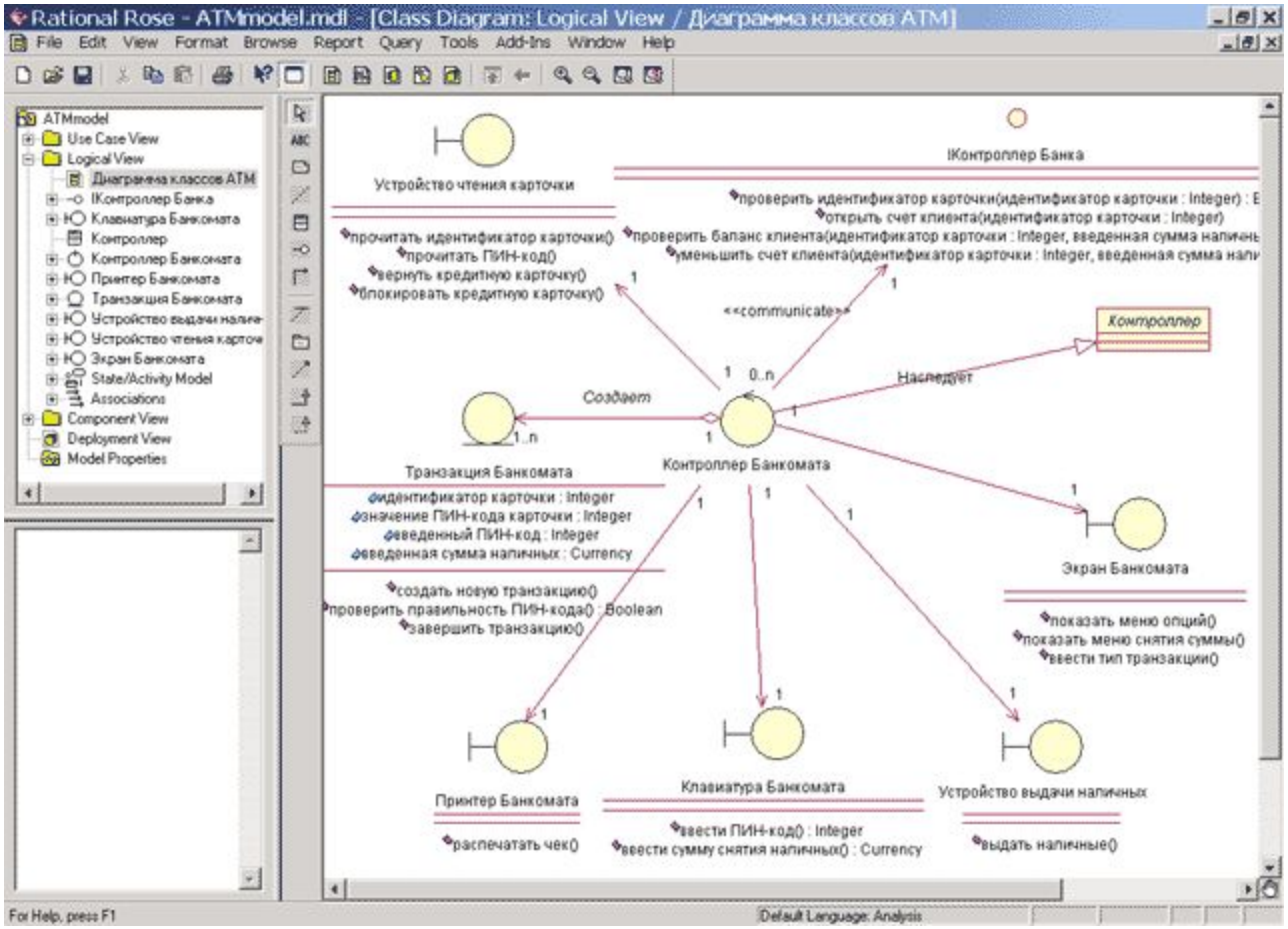


4. Отношение обобщения



Диаграммы классов

- Самостоятельная работа:



Диаграммы кооперации (взаимодействия)

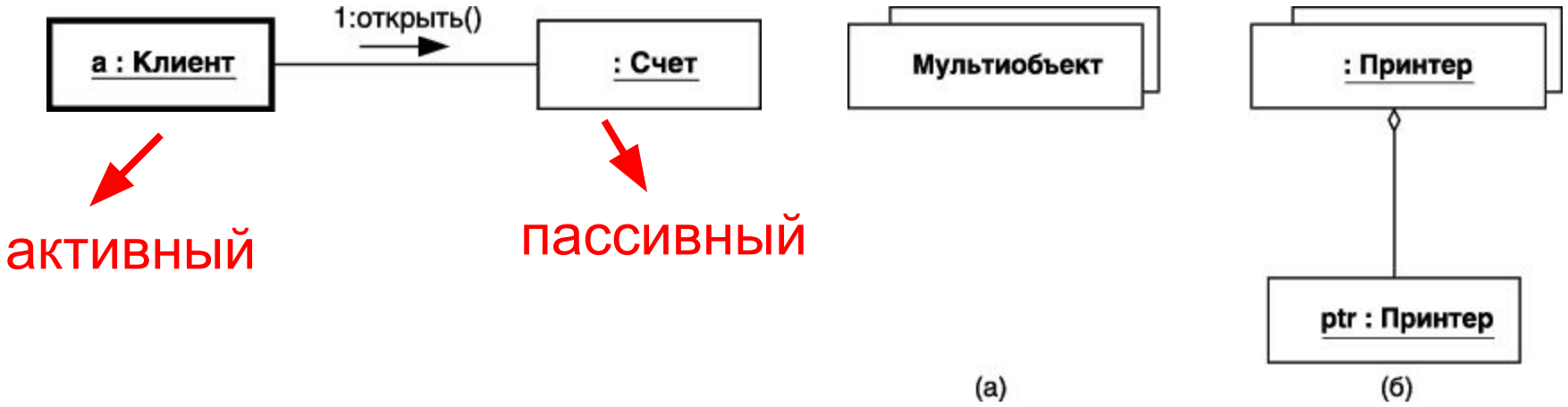
<собственное имя объекта >:<Имя класса >

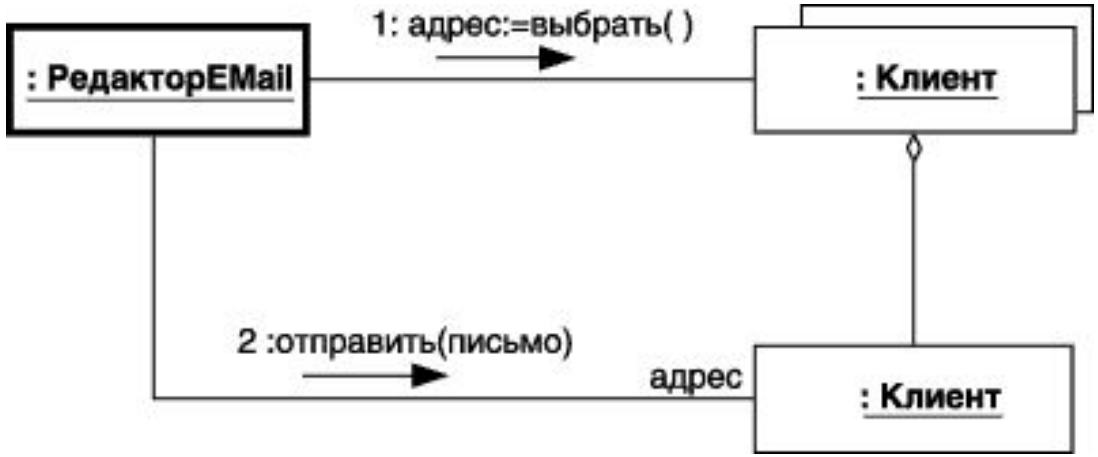
<собственное имя объекта >:<Имя класса >

- o : C– *объект* с собственным именем o, экземпляр класса C.
- : C– анонимный *объект*, экземпляр класса C.
- o :(или просто o) — *объект*-сирота с собственным именем o.

Диаграммы кооперации (взаимодействия)

Типы объектов

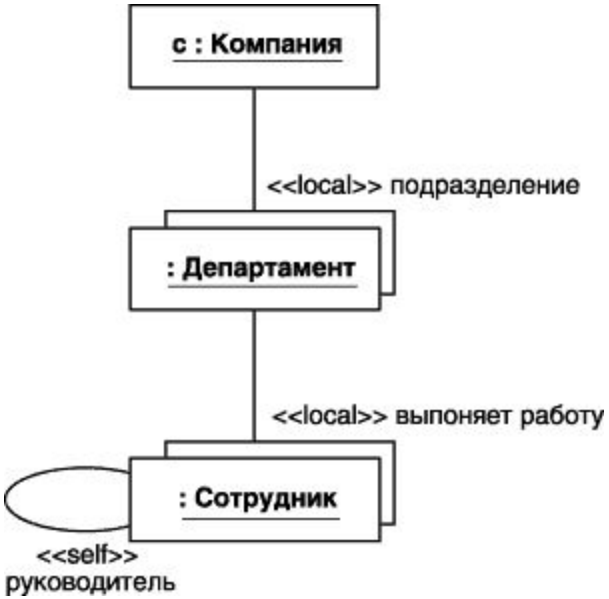




Диаграммы кооперации (взаимодействия)

СВЯЗИ

Связь(link) — любое семантическое отношение между некоторой совокупностью *объектов*.



Диаграммы кооперации (взаимодействия)

СООБЩЕНИЯ

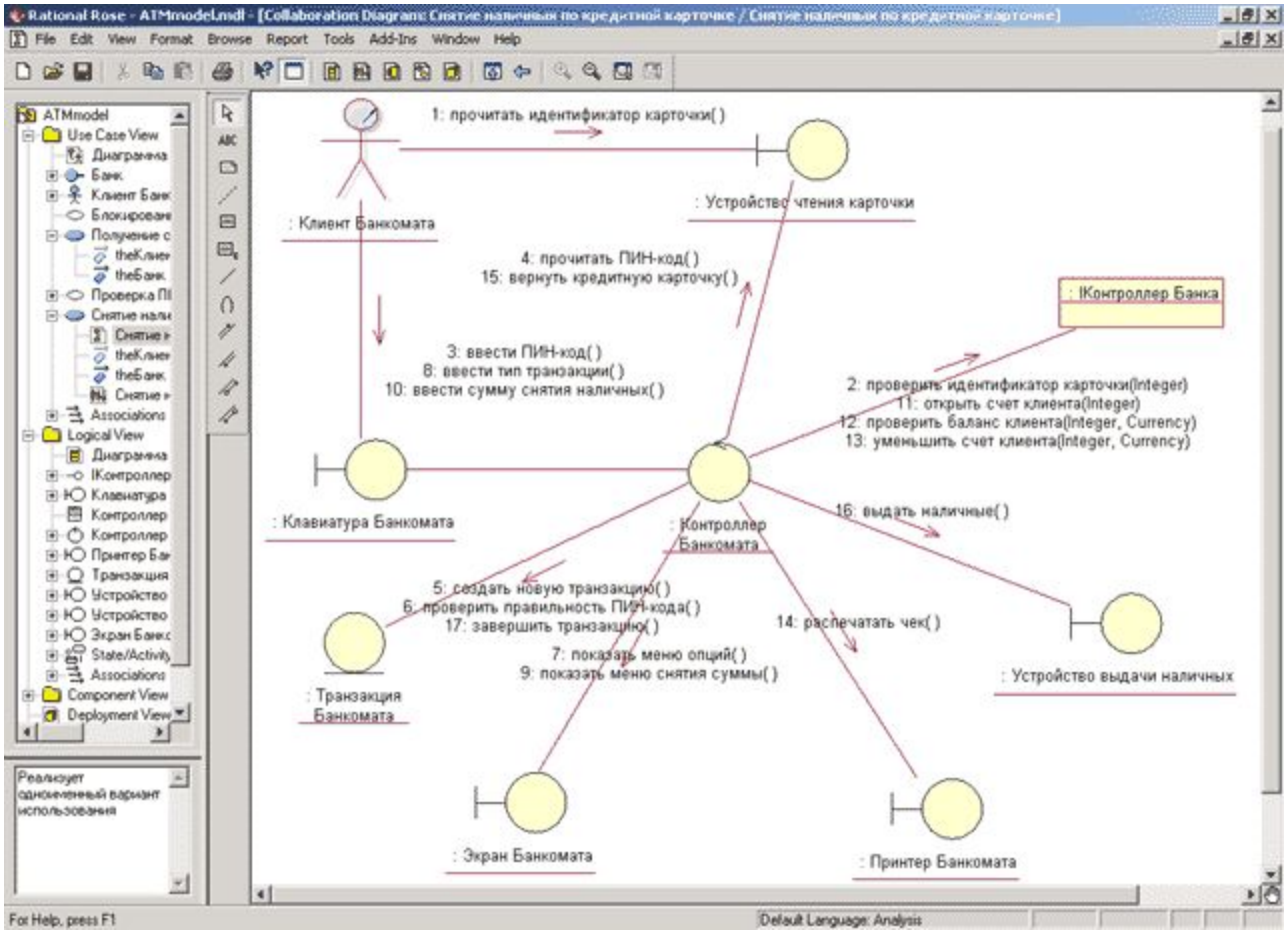
Simple (Простое)		Данное *сообщение* выполняется в одном потоке управления. Это *свойство* задается добавляемому на диаграмму *сообщению* по умолчанию
Synchronous (Синхронное)		После передачи данного *сообщения* клиент ожидает ответа от объекта-приемника о результате выполнения соответствующей операции
Balking (С отказом)		После передачи данного *сообщения* объект-приемник отказывает клиенту в выполнении соответствующей операции, если он занят выполнением других операций
Timeout (С ожиданием)		После передачи данного *сообщения* объект-приемник может поместить данное *сообщение* в очередь с ограниченным временем ожидания, если он занят выполнением других операций
Procedure Call (Выз процедуры)		Клиент посылает данное *сообщение* объекту-приемнику и, чтобы продолжить свою работу ожидает, пока вся дальнейшая вложенная последовательность *сообщений* не будет обработана приемником
Asynchronous (Асинхронное)		Клиент посылает данное *сообщение* и продолжает свою работу, не ожидая подтверждения от объекта-приемника о получении этого *сообщения*. При этом соответствующая операция может быть как выполнена, так и не выполнена
Return (Возврат)		Данное *сообщение* посылается клиенту после окончания выполнения вызова процедуры

Стереотипы сообщений

- <<call>>(вызвать) – *сообщение*, требующее вызова операции или процедуры объекта-получателя.
- <<return>>(возвратить) – *сообщение*, возвращающее значение выполненной операции или процедуры вызвавшему ее *объекту*.
- <<create>>(создать) – *сообщение*, требующее создания другого *объекта* для выполнения определенных действий.
- <<destroy>>(уничтожить) – *сообщение* с явным требованием уничтожить соответствующий *объект*.

Диаграммы кооперации (взаимодействия)

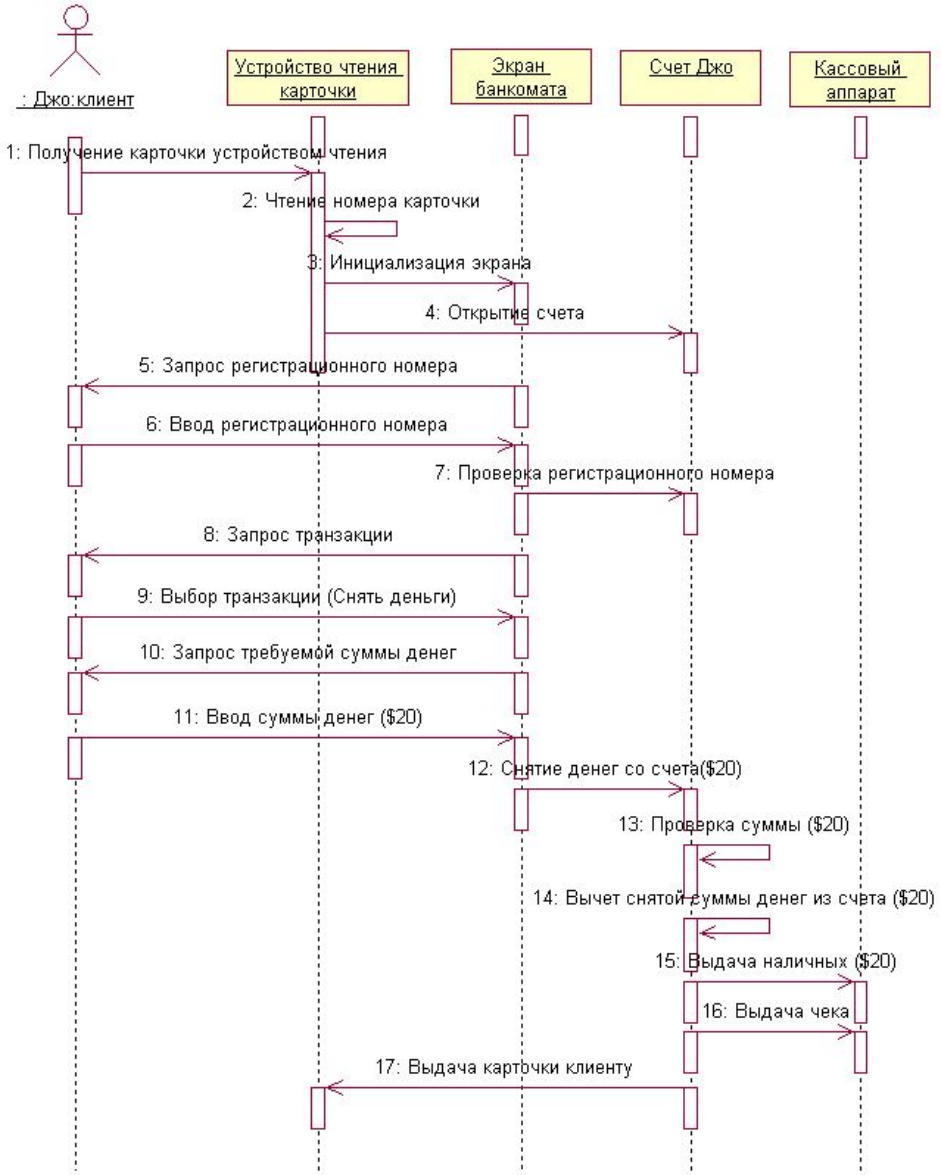
- Самостоятельная работа:



Диаграммы последовательности

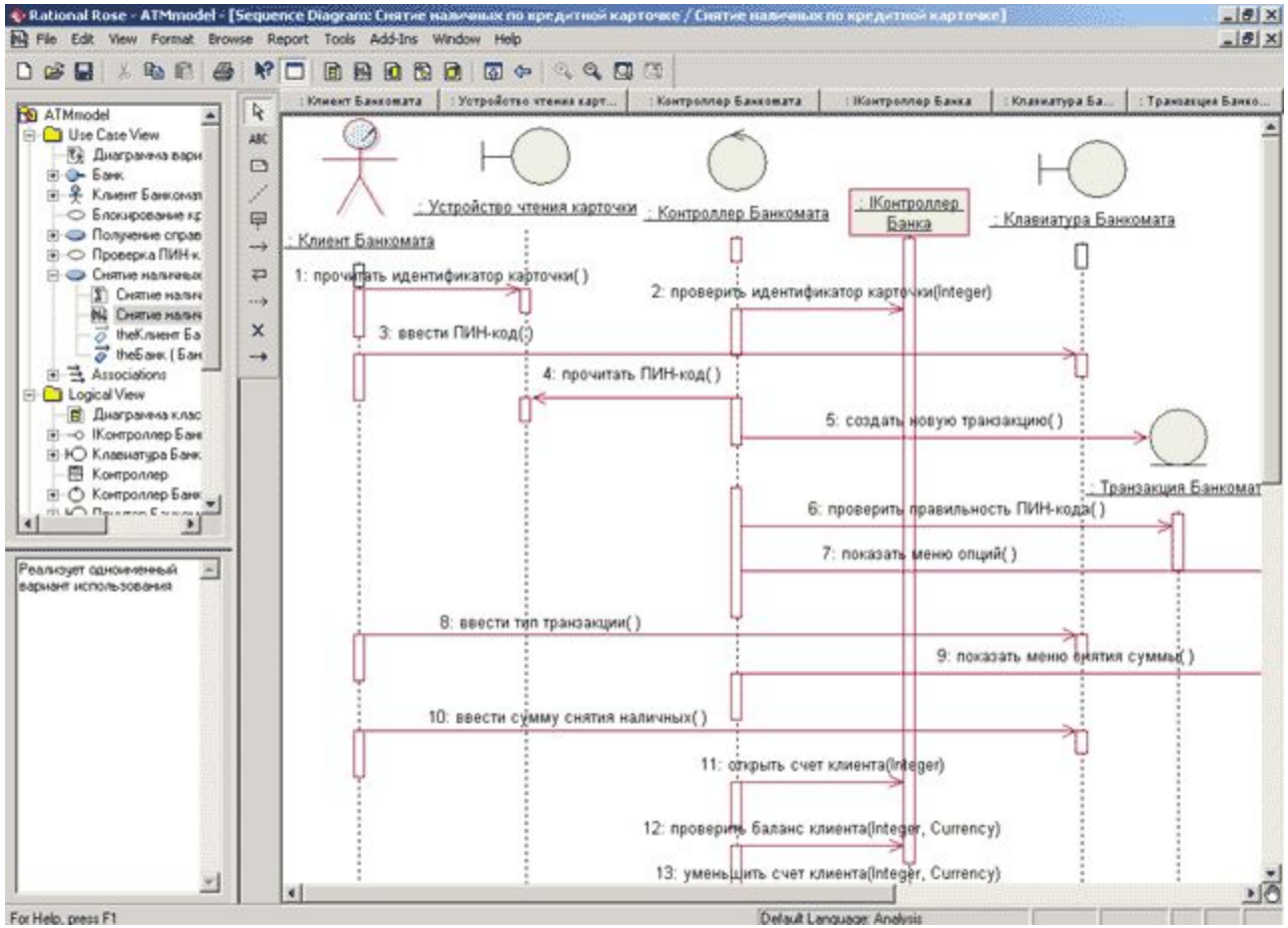
- ***Диаграмма последовательности (sequence diagram)*** - диаграмма, на которой показаны взаимодействия объектов, упорядоченные по времени их проявления.

- Глядя на эту диаграмму пользователи знакомятся со спецификой своей работы.
- Аналитики видят последовательность действий, разработчики – объекты, которые надо создать, и их операции.
- Специалисты по контролю качества поймут детали процесса и смогут разработать тесты для их проверки.



Диаграммы последовательности

- Самостоятельная работа:



Представление компонентов

Диаграмма компонентов

- Компонент – физический модуль кода.

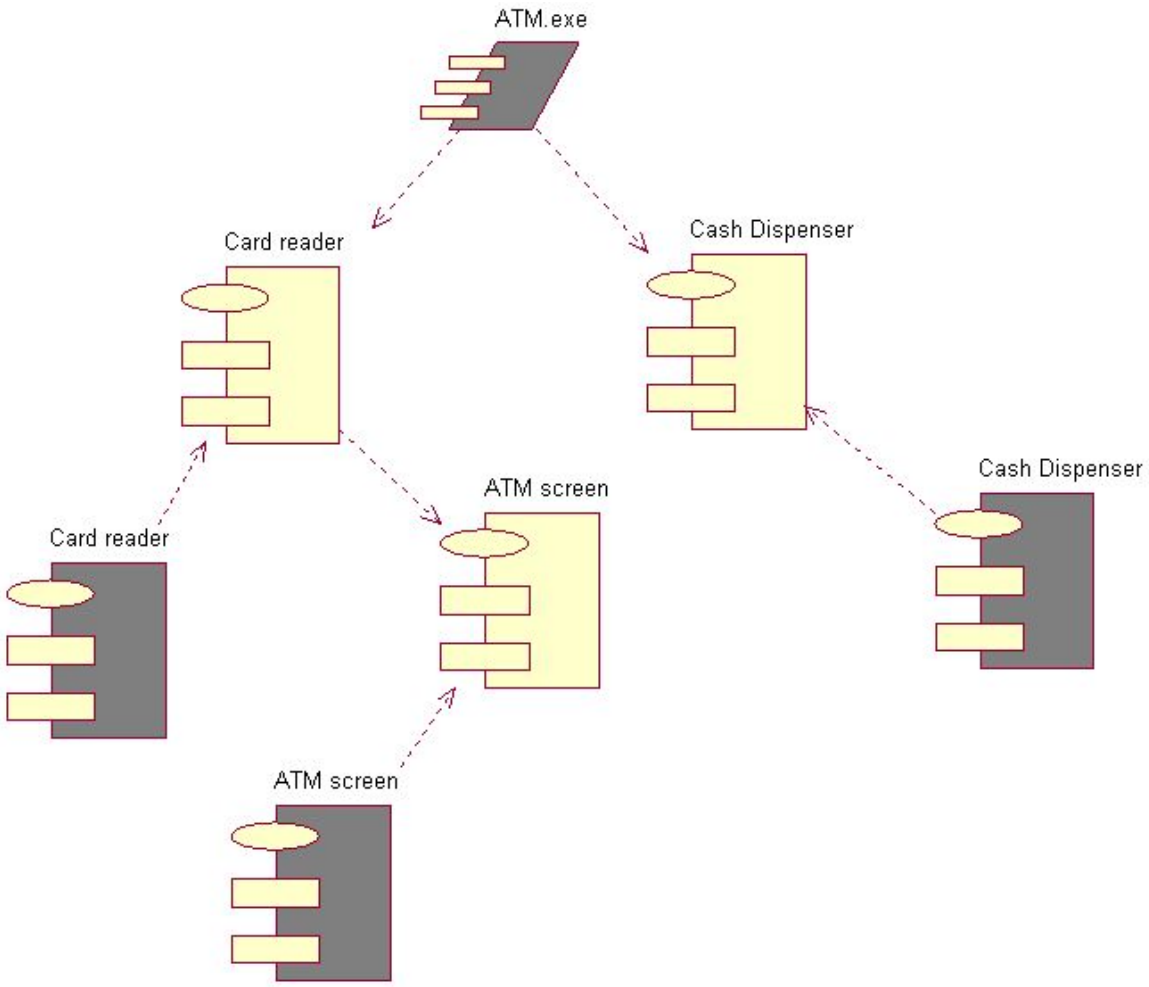
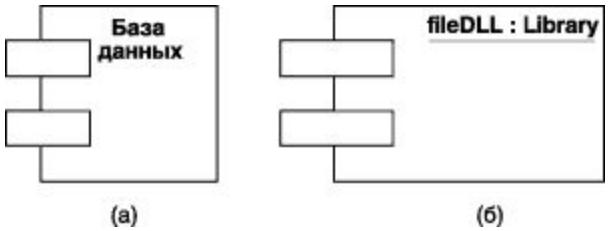
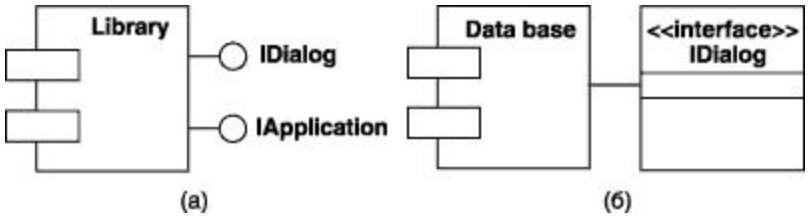


Диаграмма компонентов

Типы компонентов:





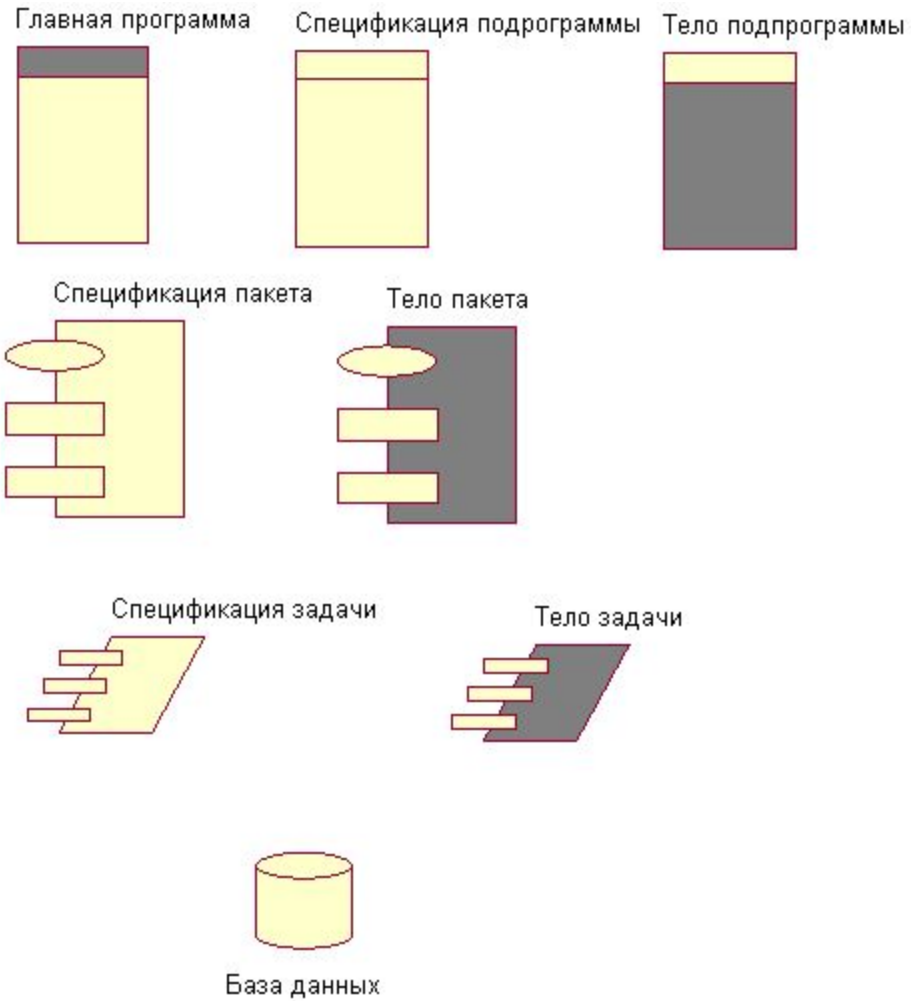
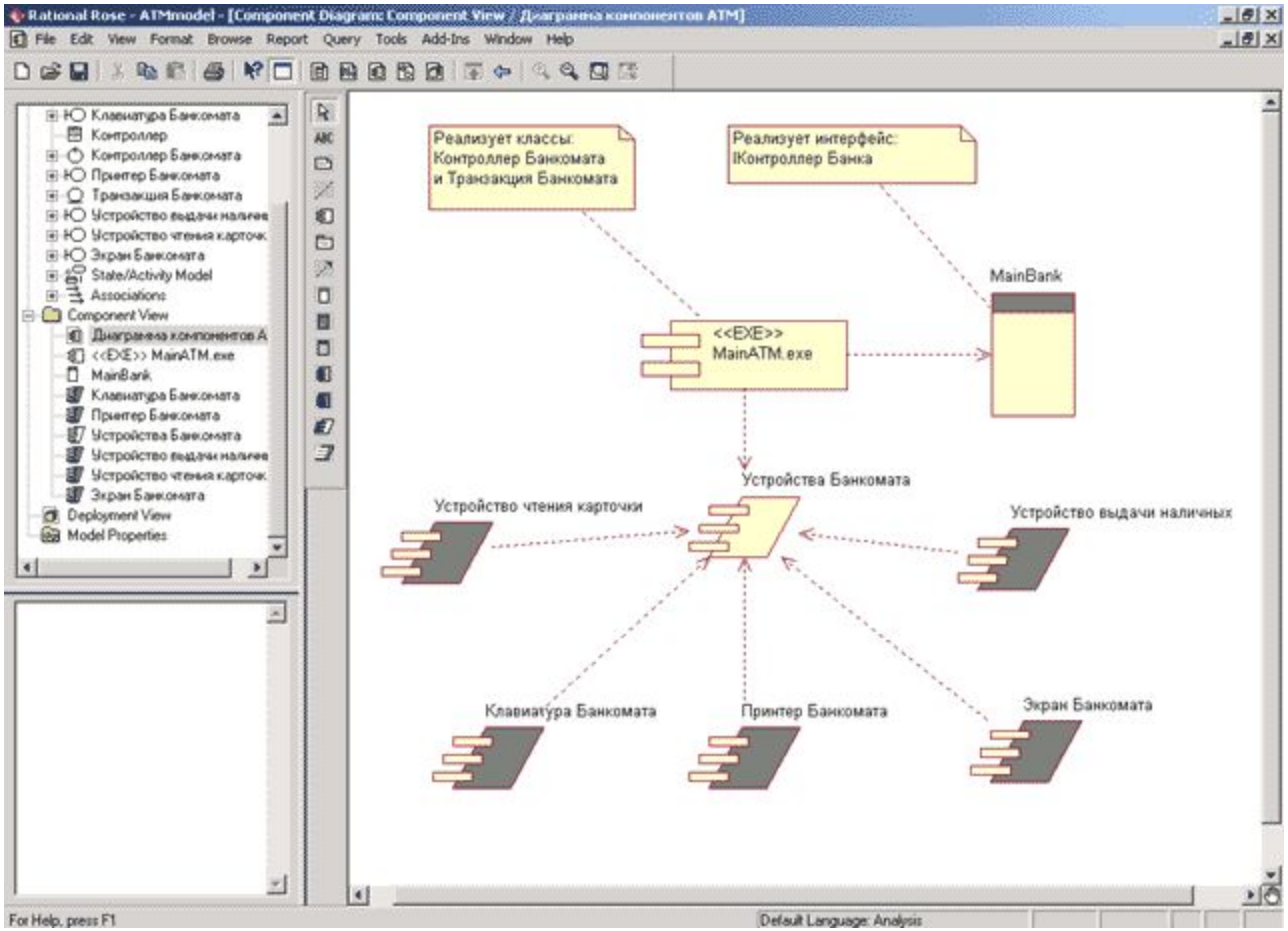


Диаграмма компонентов

- Самостоятельная работа:



Генерация кода

ANSI C++

Этапы:

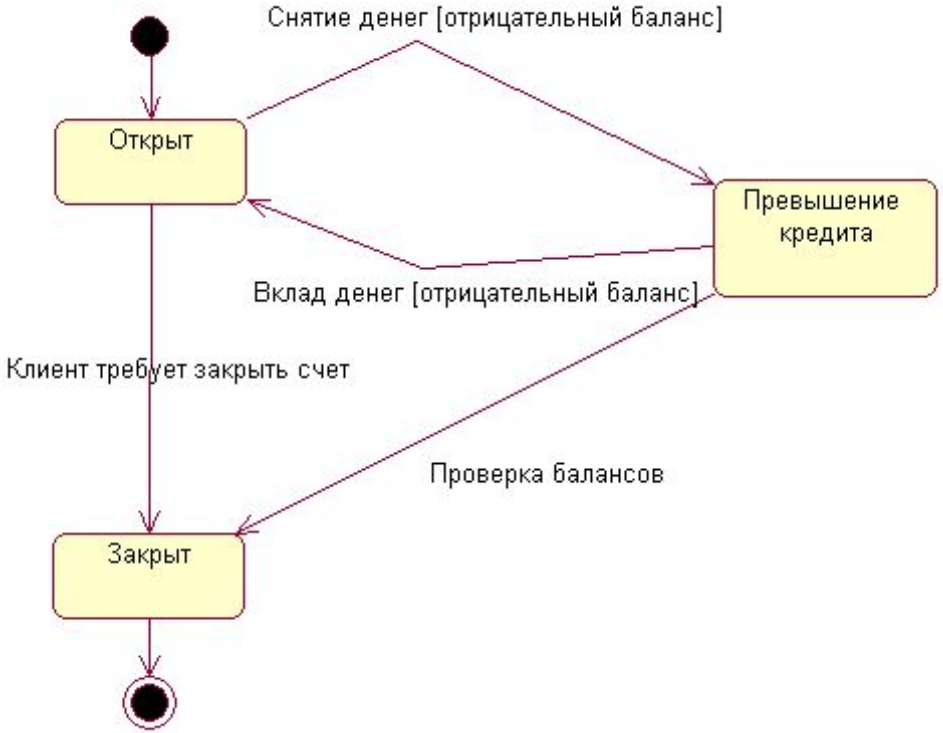
- Проверка модели на отсутствие ошибок.
- Создание *компонентов* для реализации *классов*.
- Отображение *классов* на *компоненты*.
- Выбор *языка программирования* для генерации текста программного кода.
- Установка свойств генерации программного кода.
- Выбор *класса*, *компонента* или пакета.
- Генерация программного кода.

Для генерации кода на C++ необходимо:

1. Создать компоненты
2. Определить компоненты для классов
3. Установить свойства генерации программного кода
4. Выбрать класс или компонент для генерации на диаграмме Классов или Компонентов
5. Выбрать в меню Tools – ANSI C++ -- Code Generation
6. Выбрать в меню Tools – ANSI C++ -- Browse Header или Browse Body для просмотра сгенерированного заголовка или программного кода

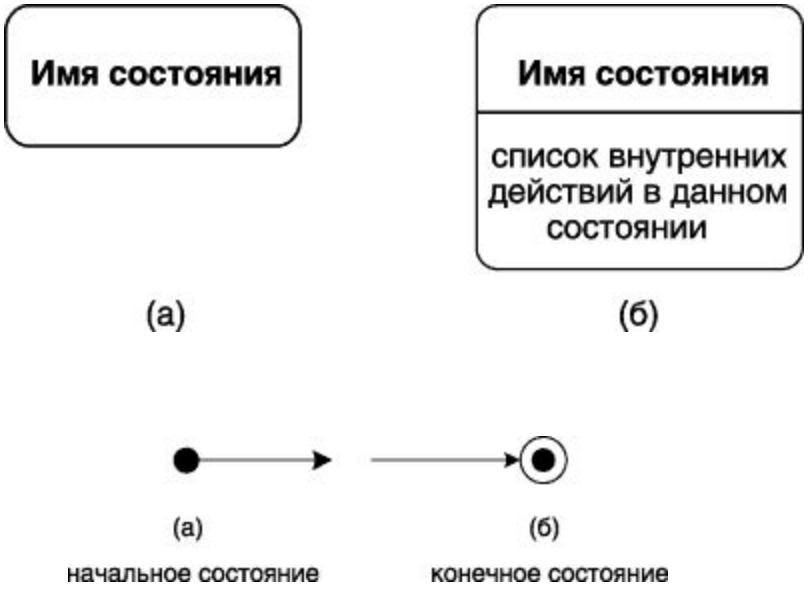
Диаграммы состояний

- На диаграмме состояний отображают жизненный цикл одного объекта, начиная с момента его создания и заканчивая разрушением.
- С помощью таких диаграмм удобно моделировать динамику поведения класса.



Диаграммы состояний

- *Состояние (state)* - **условие или ситуация в ходе жизненного цикла объекта, в течение которого он удовлетворяет логическому условию, выполняет определенную деятельность или ожидает события.**

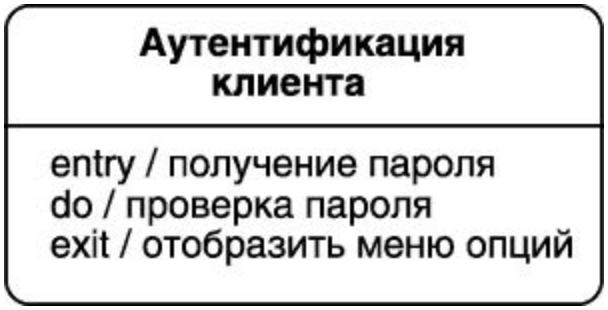


Диаграммы состояний

- Действие (action) - спецификация выполнимого утверждения, которая образует абстракцию вычислительной процедуры.

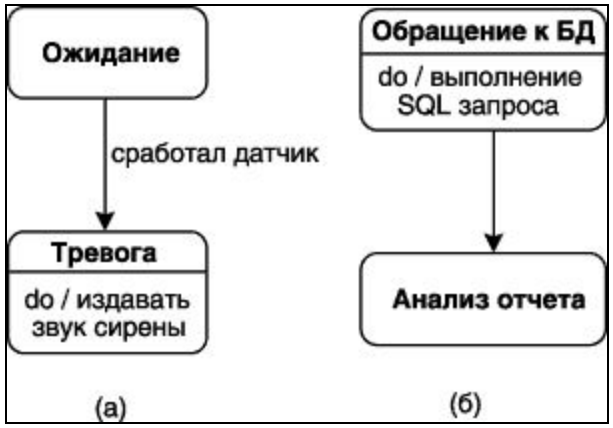
- Каждое действие записывается в виде отдельной строки и имеет следующий формат:
  - **<метка действия '/ ' выражение действия>**

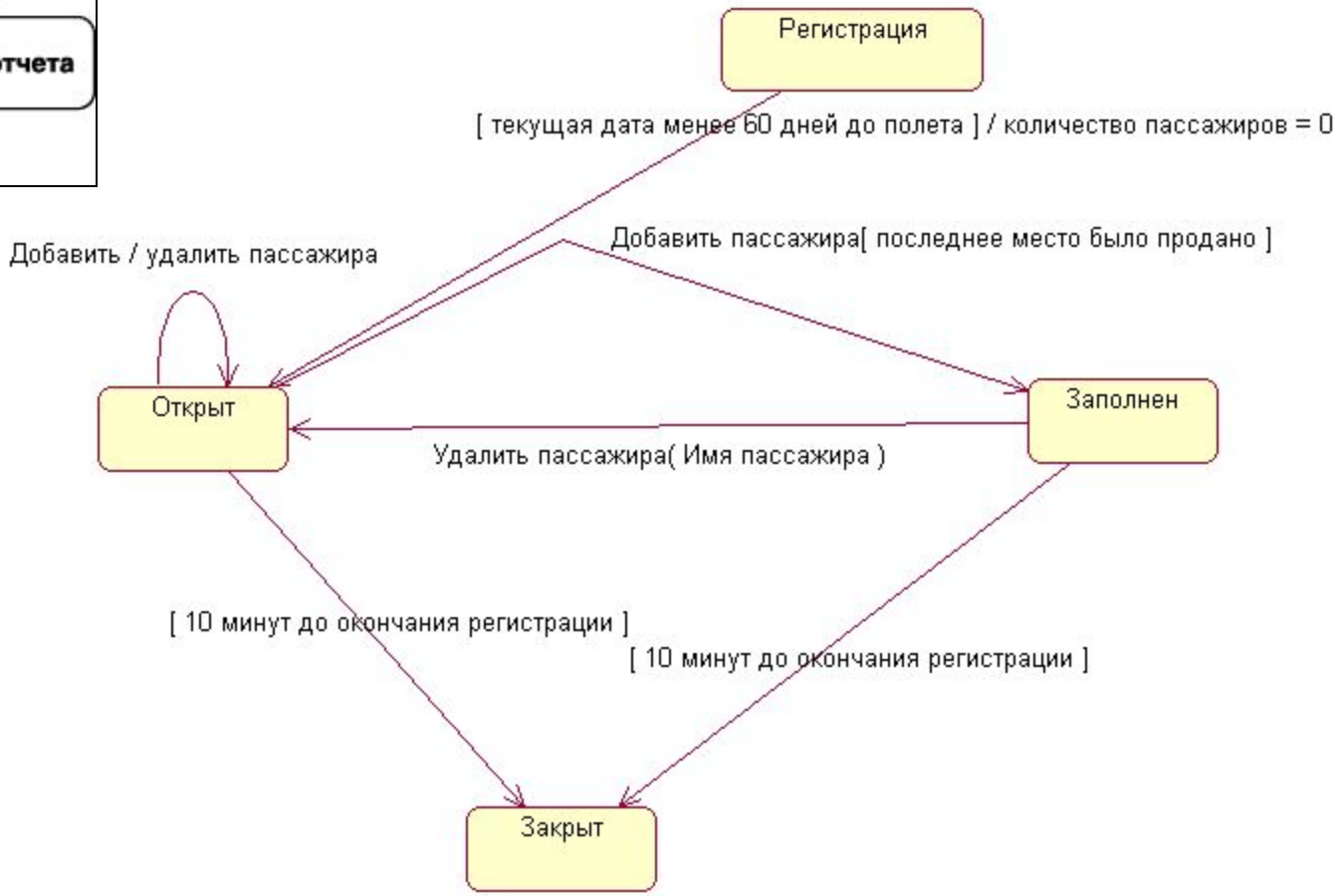
- Метки действия:
  - Входное действие
  - Действие выхода
  - Деятельность



Диаграммы состояний

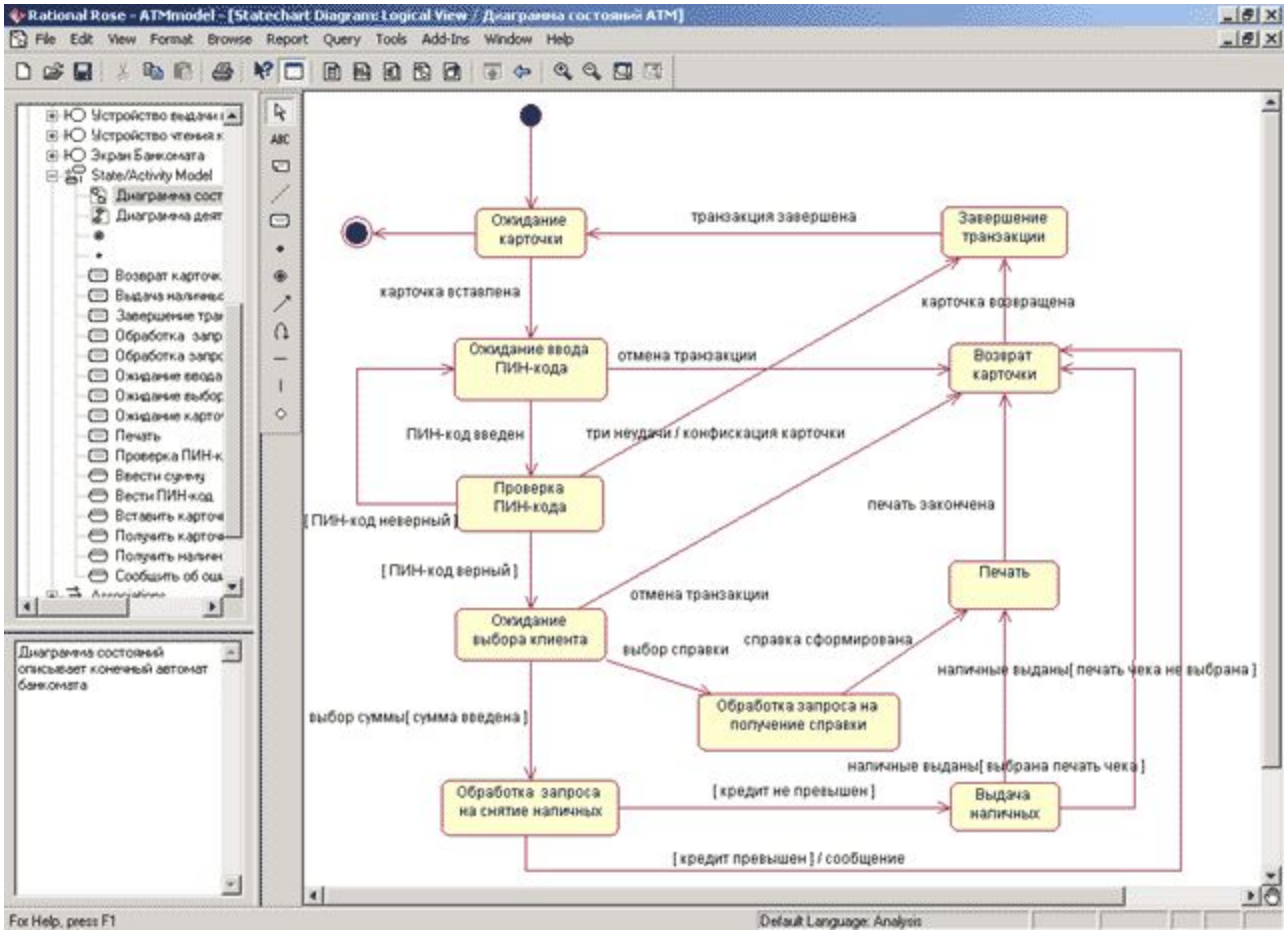
ПЕРЕХОДЫ





Диаграммы состояний

- Самостоятельная работа:

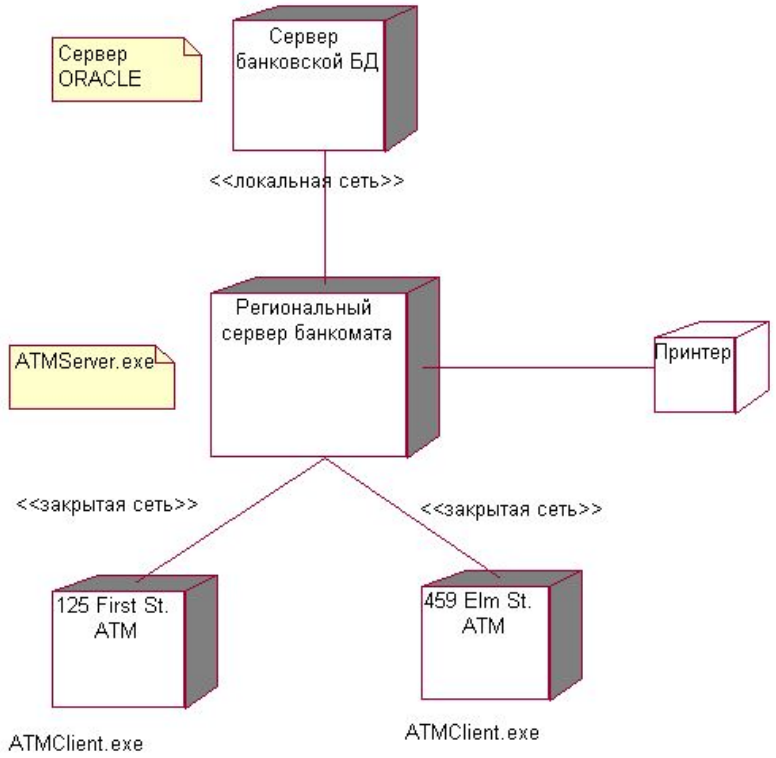


Представление размещения

Deployment View

Диаграммы размещения

- Показывает физическое расположение различных компонентов системы в сети.



Механизмы расширения базовой семантики объектов модели

- ***Стереотип (stereotype)*** — новый тип элемента *модели*, который расширяет семантику метамодели. *Стереотипы* должны основываться на уже существующих и описанных в метамодели языка UML типах или классах.
- *Стереотипы* предназначены для расширения именно семантики, но не структуры уже описанных типов или классов. Некоторые *стереотипы* предопределены в языке UML, другие могут быть указаны разработчиком. На *диаграммах* изображаются в форме текста, заключенного в угловые кавычки. Предопределенные *стереотипы* являются ключевыми словами языка UML, которые используются на *канонических диаграммах* на языке оригинала без их перевода.
- ***Помеченное значение (tagged value)*** — явное определение свойства как пары "имя – значение". В *помеченном значении* само имя называют тегом (tag).
- *Помеченные значения* на *диаграммах* изображаются в форме строки текста специального формата, заключенного в фигурные скобки. При этом используется следующий формат записи: {тег = значение}. Теги встречаются в нотации языка UML, но их определение не является строгим, поэтому теги могут быть указаны самим разработчиком.
- ***Ограничение (constraint)*** — некоторое логическое условие, ограничивающее семантику выбранного элемента *модели*.
- Как правило, все *ограничения* специфицируются разработчиком. *Ограничения* на *диаграммах* изображаются в форме строки текста, заключенного в фигурные скобки. Для формальной записи *ограничений* предназначен специальный язык объектных *ограничений* (Object Constraint Language, OCL), который является составной частью языка UML.