

UML

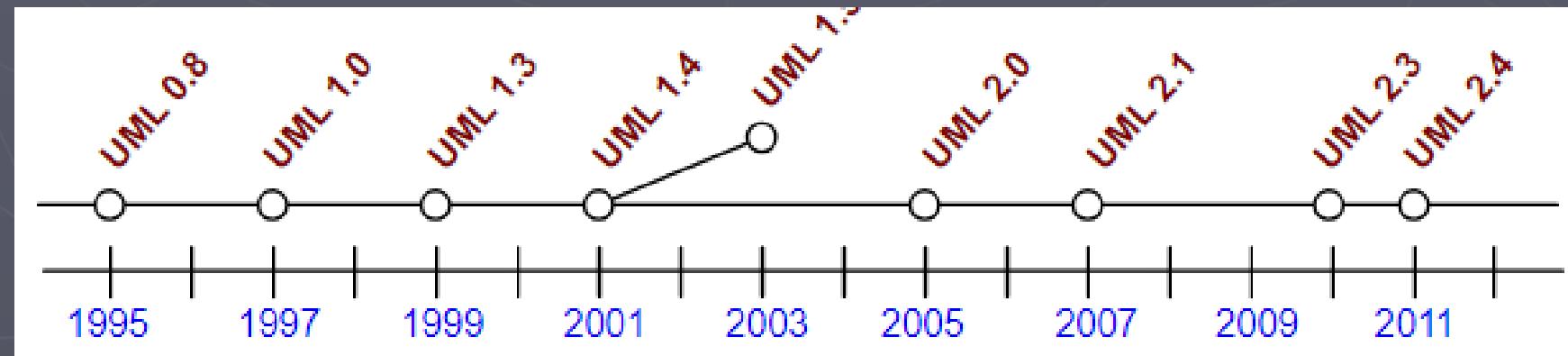
Unified Modeling Language

UML – Unified Modeling Language

- язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур

- 1995 - Гради Буч и Джеймс Рамбо (Rational Software) и Ивар Якосон

- 0.8 *Unified Method*
 - 2.5.1 - 2017

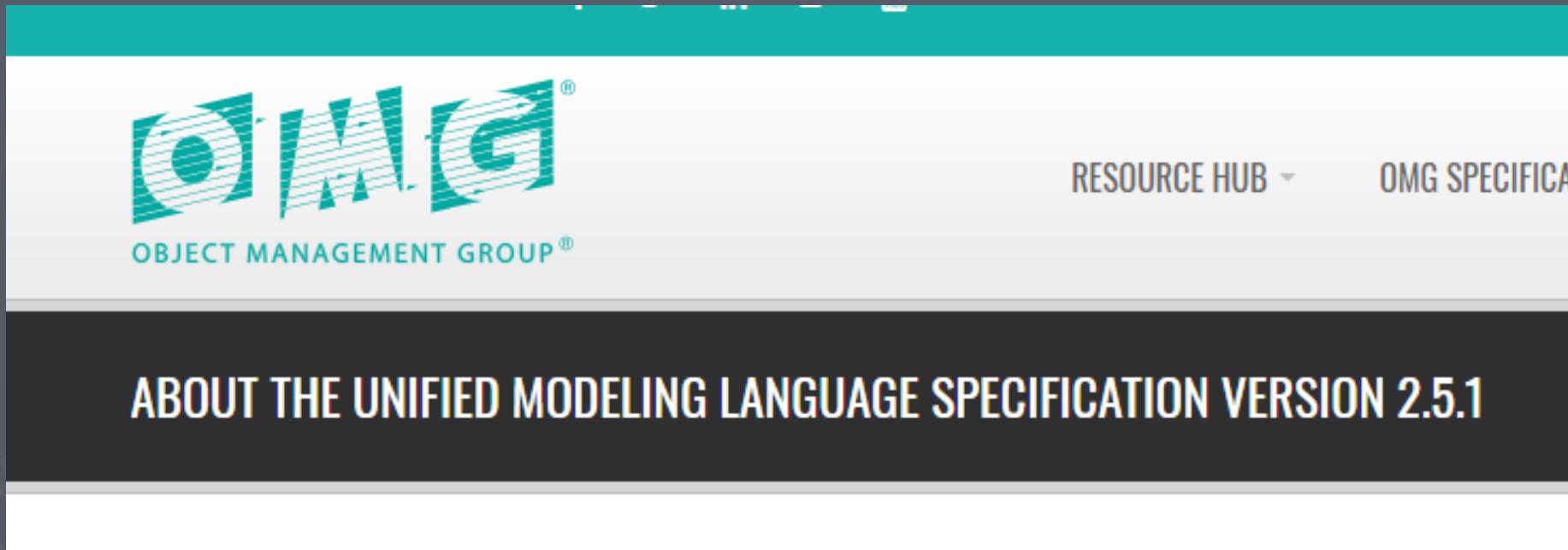


- UML 2.4.1 принят в качестве международного стандарта ISO/IEC 19505-1, 19505-2

- альтернативы

- [SysML](#)
 - [IDEF](#)
 - [DFD](#)
 - [ДРАКОН](#)

► <https://www.omg.org/spec/UML/About-UML/>



<http://book.uml3.ru/content>

Назначение UML

- ▶ Язык **UML** – это графический язык моделирования общего назначения, предназначенный для спецификации, визуализации, проектирования и документирования всех артефактов, создаваемых при разработке программных систем.

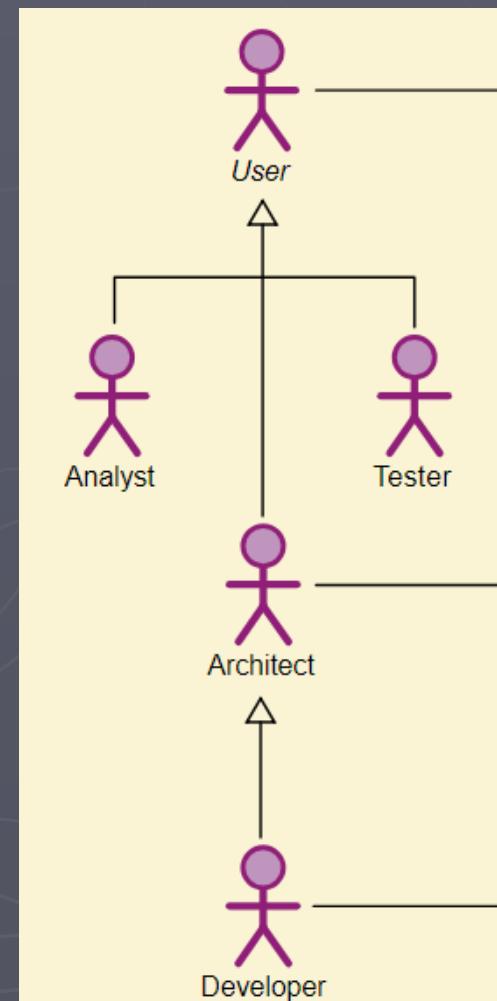
▶ Спецификация

- декларативное описание того, как нечто устроено или работает

▶ Визуализация

▶ Проектирование

▶ Документирование



Виды диаграммы

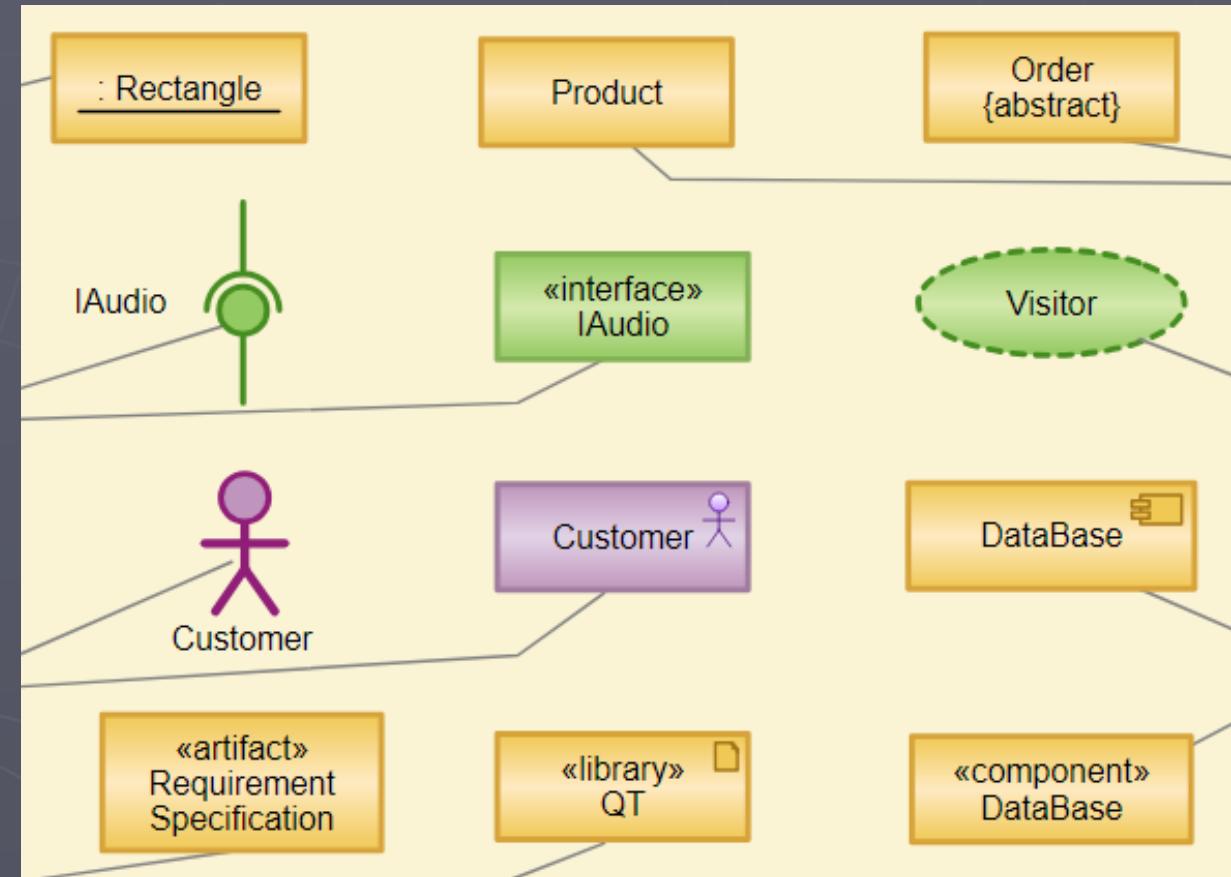
► **Диаграмма (diagram)** – это графическое представление некоторой части графа модели

► Сущности

- структурные;
- поведенческие;
- группирующие;
- аннотационные.

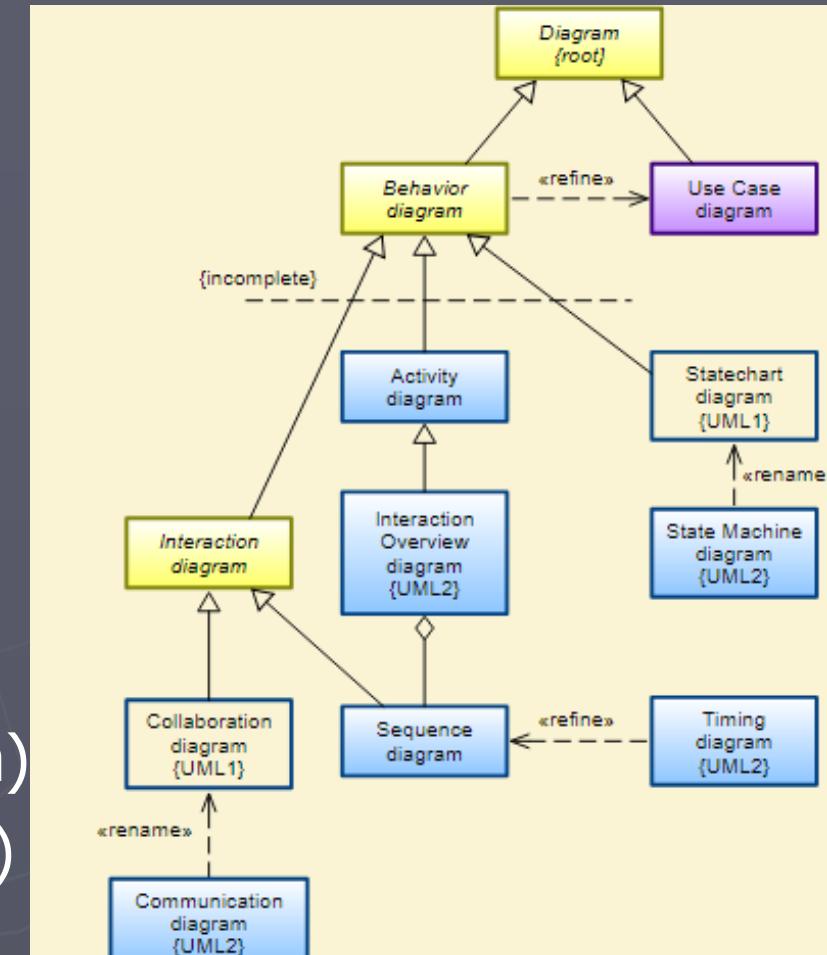
► Отношения

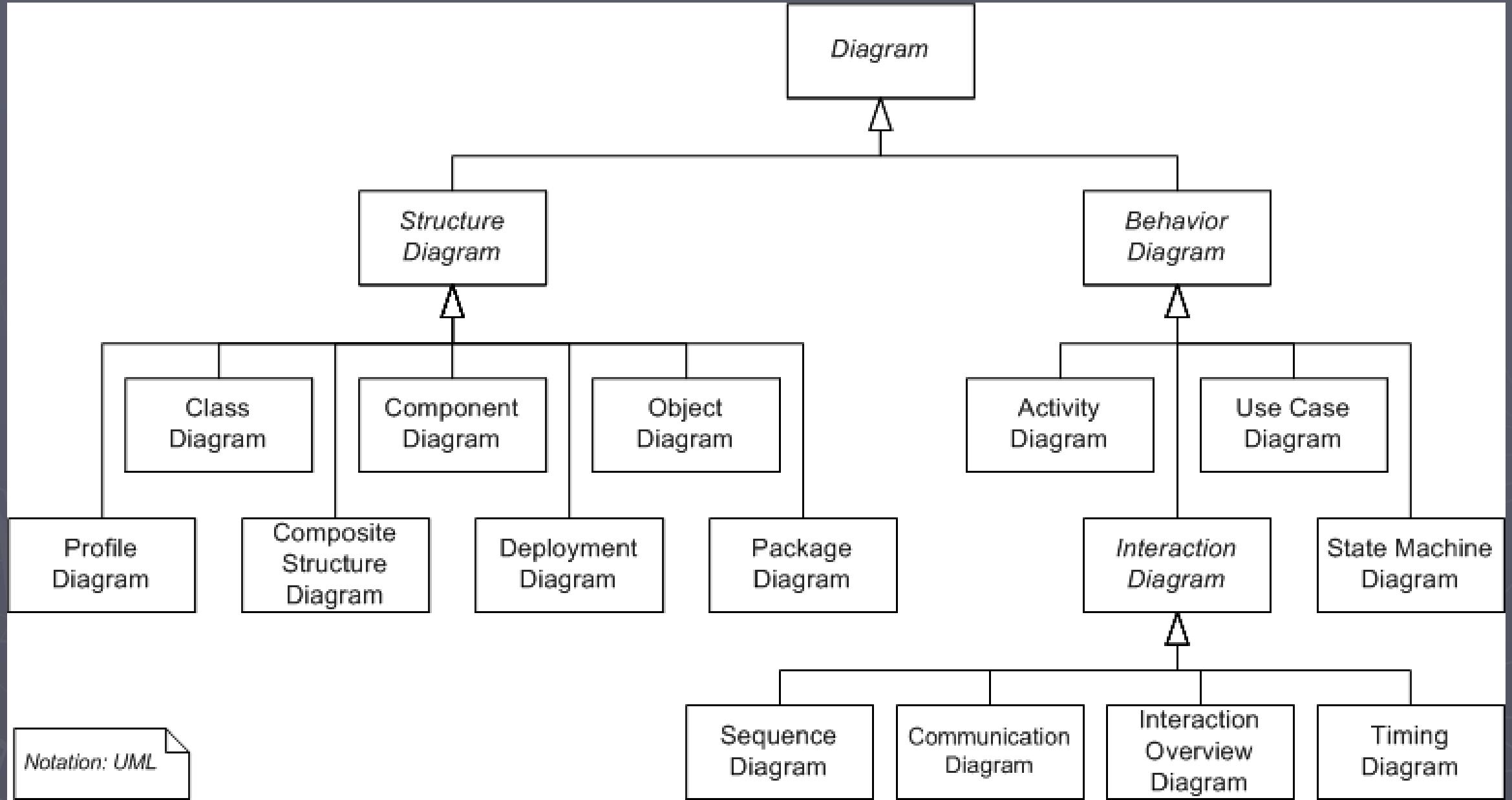
- зависимость (dependency);
- ассоциация (association);
- обобщение (generalization);
- реализация (realization).



Виды диаграммы

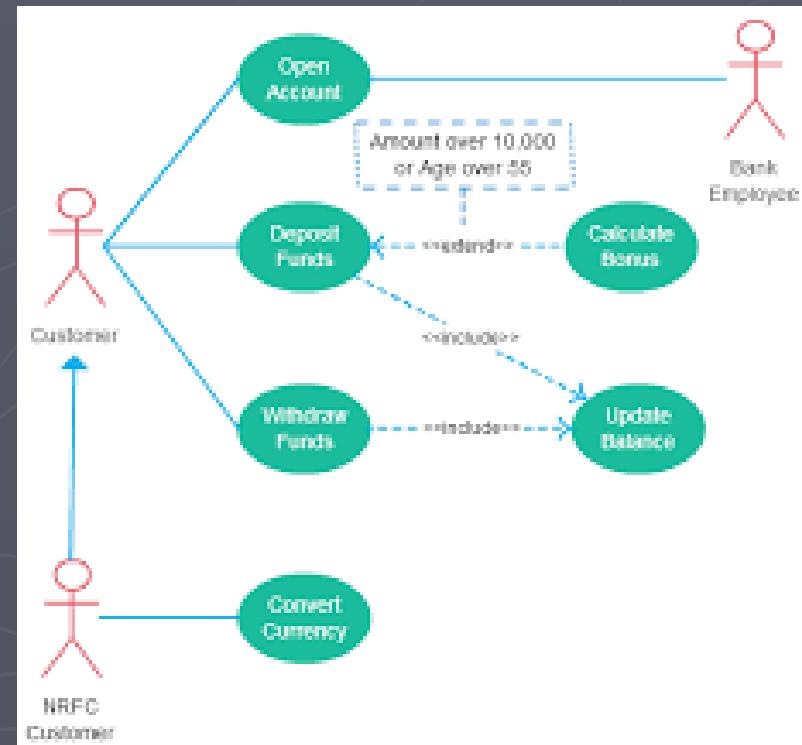
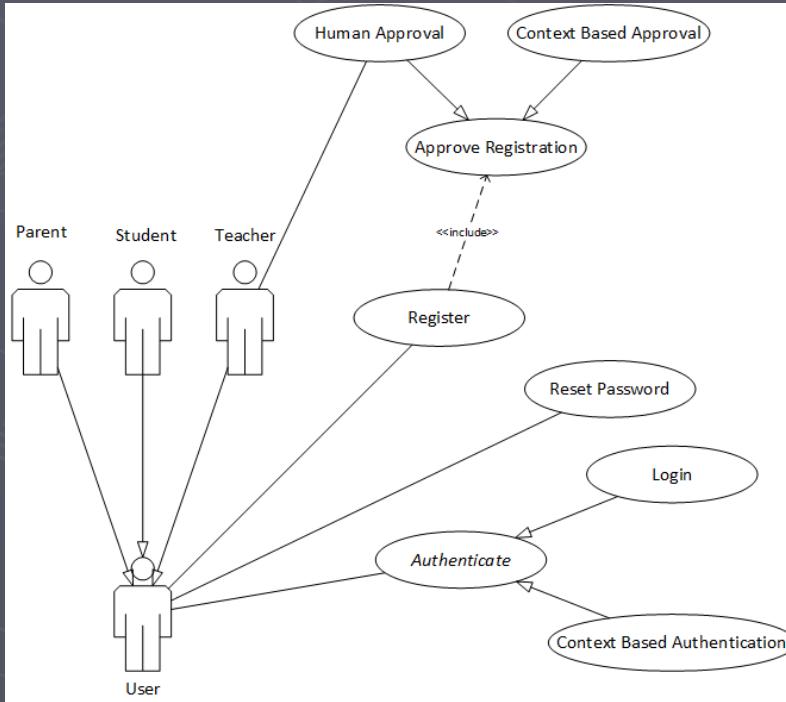
- ▶ Диаграмма использования (Use Case diagram)
- ▶ Диаграмма классов (Class diagram)
- ▶ Диаграмма объектов (Object diagram)
- ▶ Диаграмма автомата (State machine diagram)
- ▶ Диаграмма деятельности (Activity diagram)
- ▶ Диаграмма последовательности (Sequence diagram)
- ▶ Диаграмма коммуникации (Communication diagram)
- ▶ Диаграмма компонентов (Component diagram)
- ▶ Диаграмма размещения (Deployment diagram)
- ▶ Обзорная диаграмма взаимодействия (Interaction Overview diagram)
- ▶ Диаграмма синхронизации (Timing diagram)





Диаграммы использования

- ▶ use case diagrams
- ▶ общее представление функционального назначения системы.
- ▶ Типы сущностей: варианты использования и действующие лица ,



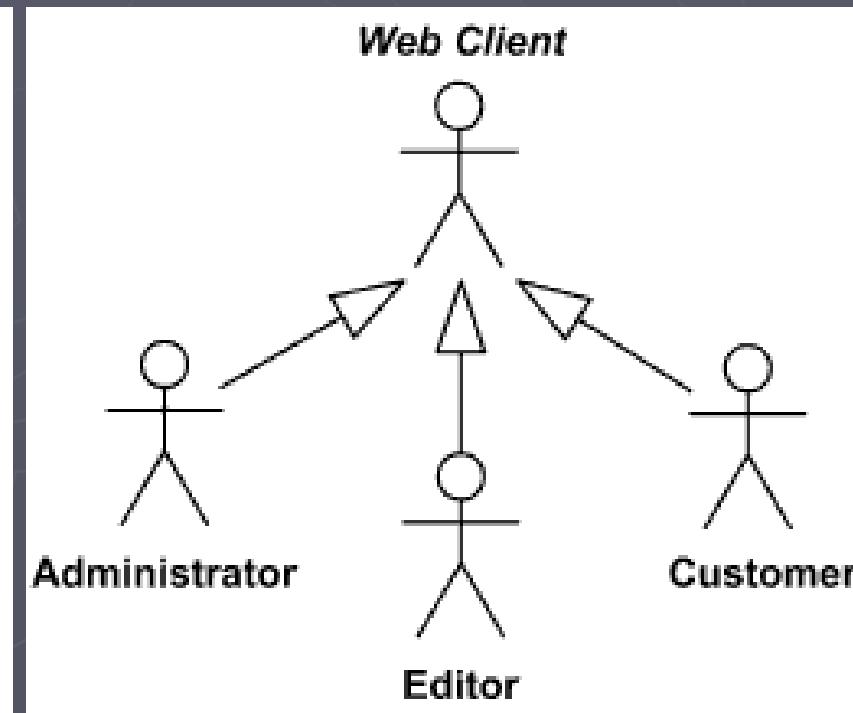
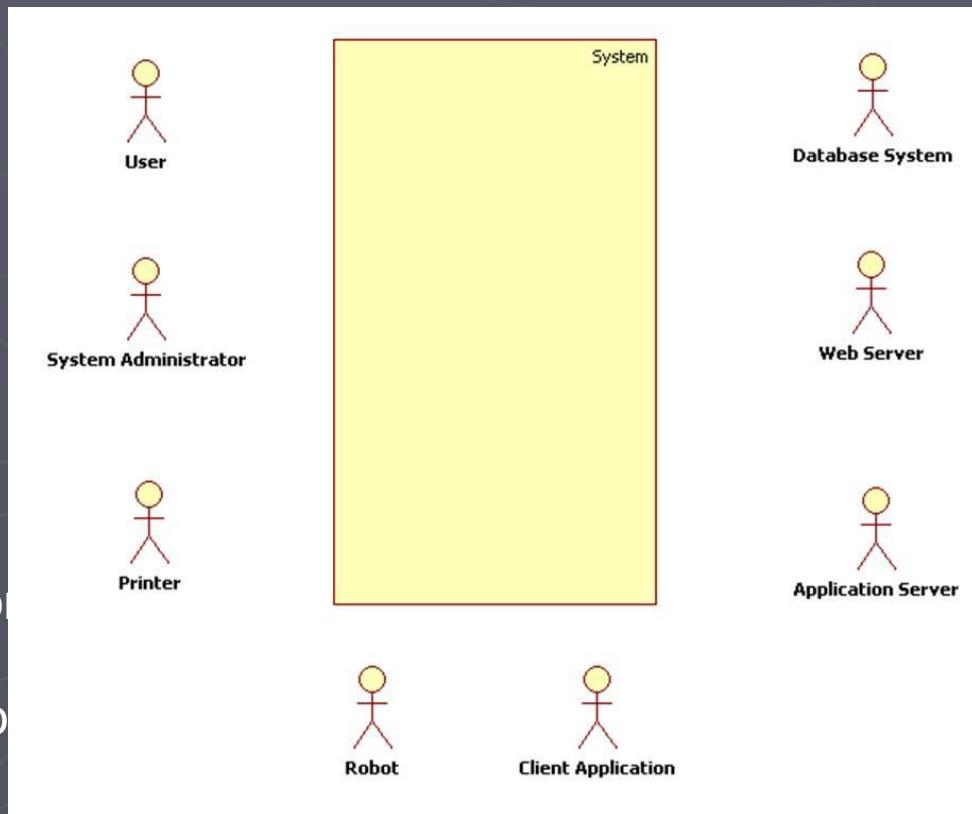
Действующее лицо

(actor) – это роль , которую пользователь играет по отношению к системе .

- ▶ пользователи системы ,
- ▶ другие системы , взаимодействующие с данной
- ▶ время

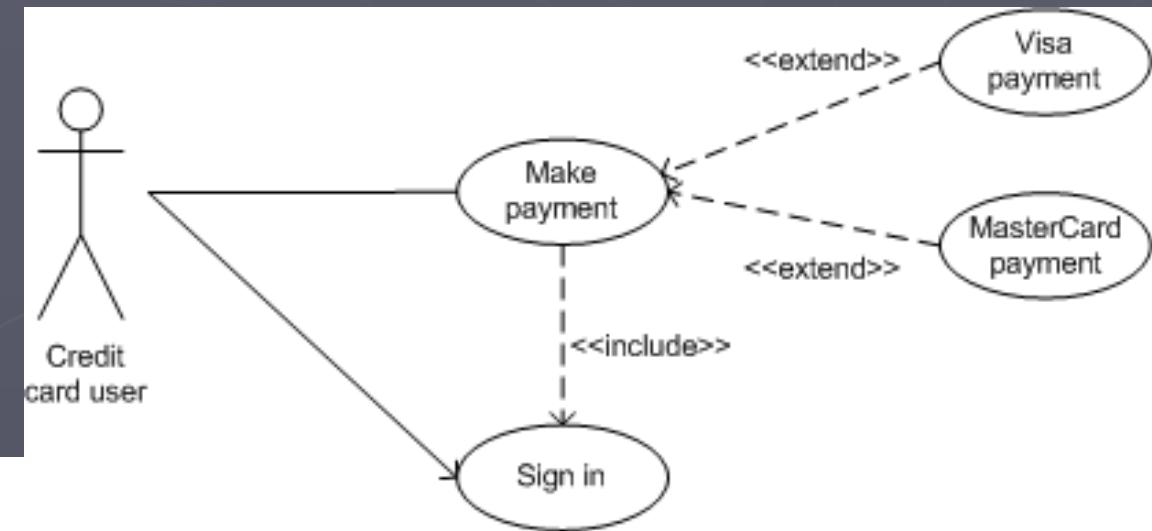
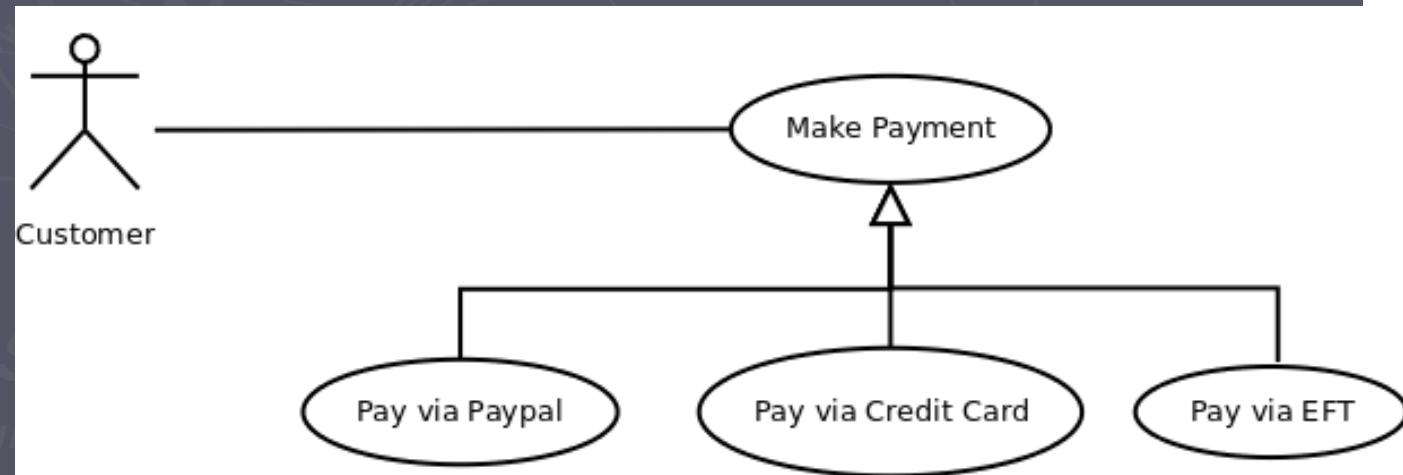
признаки:

- пользователи участвуют в разных (независимых) бизнес-процессах;
- пользователи имеют различные права на выполнение действий и доступ к информации;
- пользователи взаимодействуют с системой в разных режимах: от случая к случаю, регулярно постоянно.



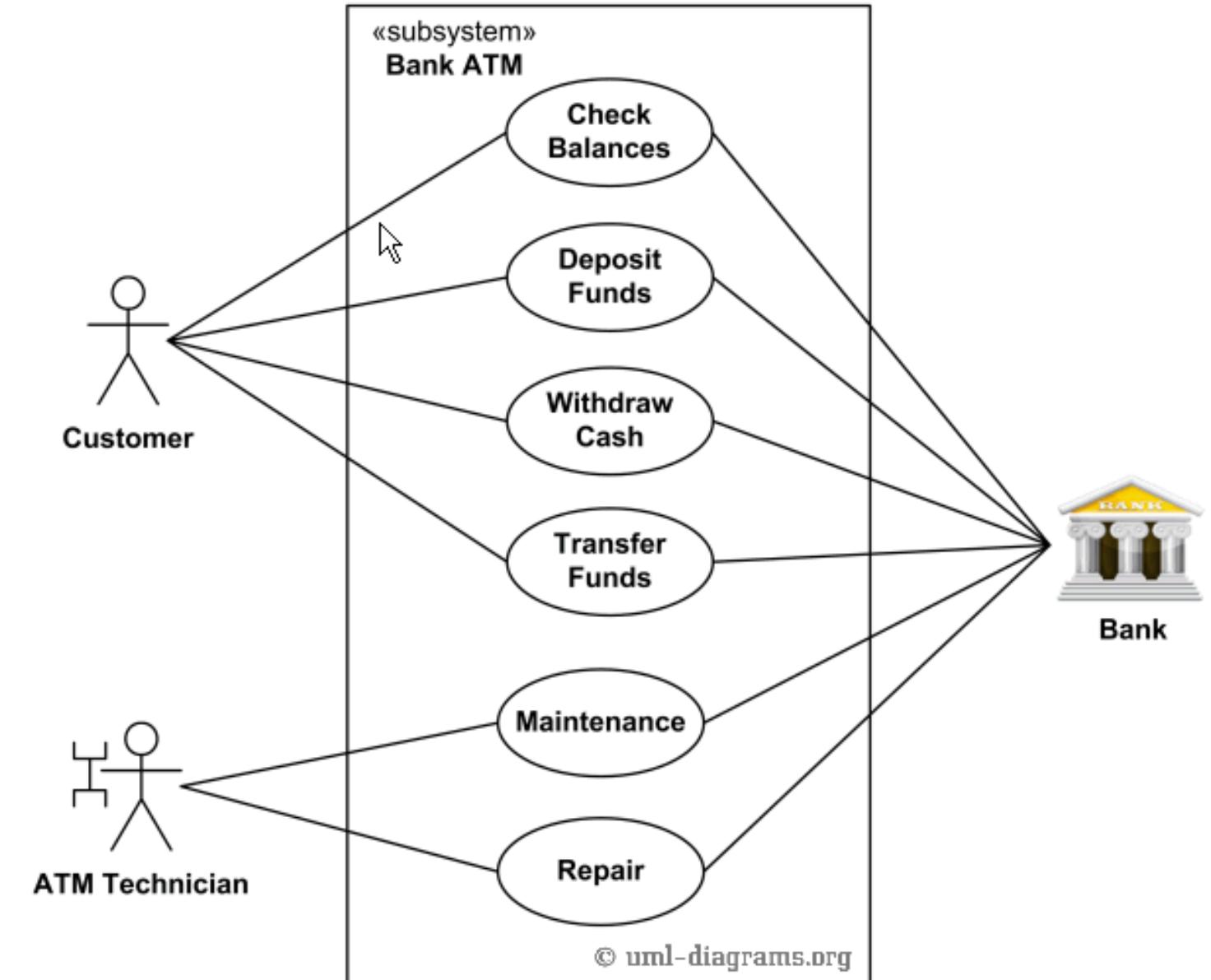
Связи между вариантами использования и действующими лицами

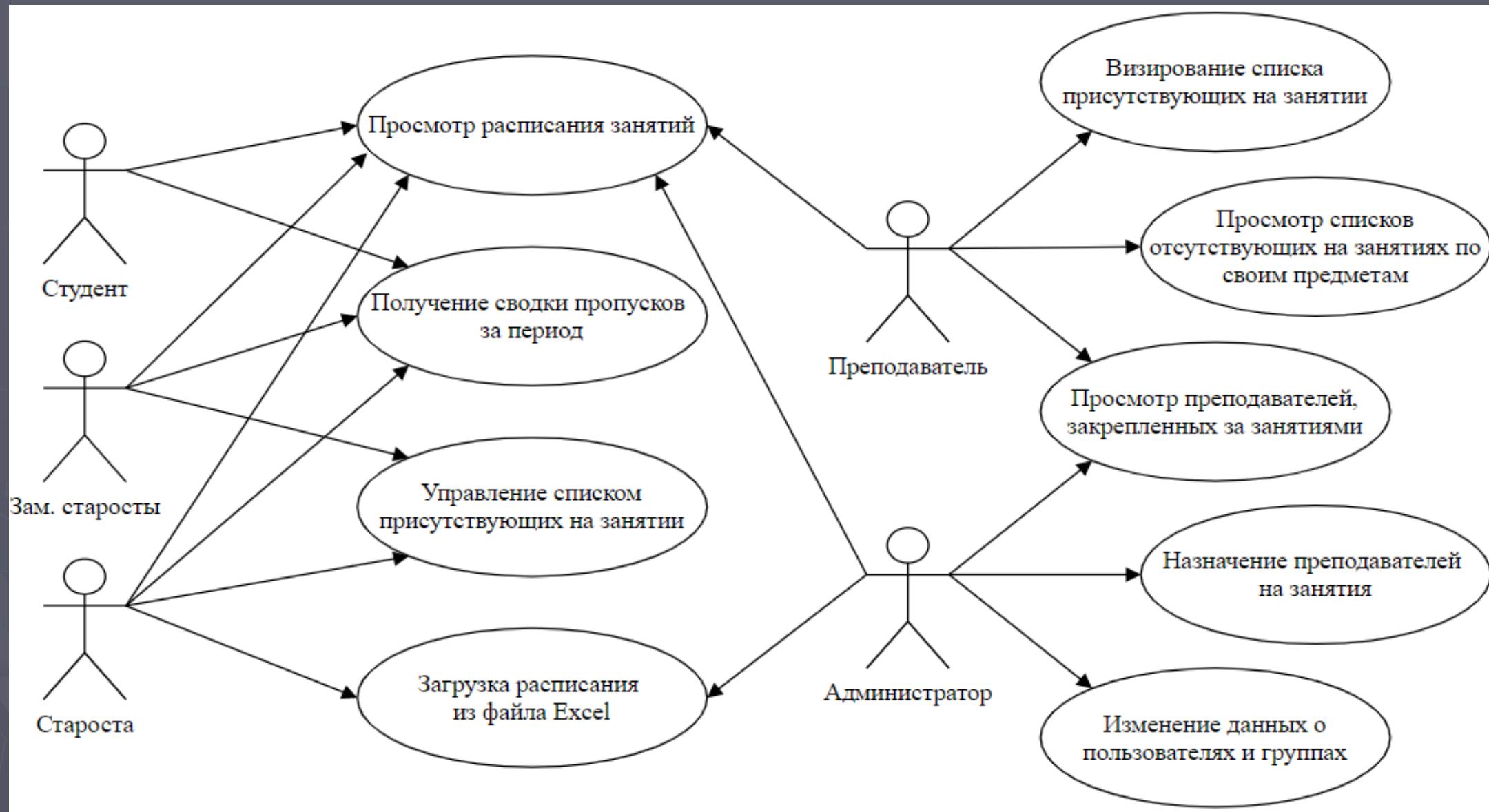
- коммуникации (communication), —————→
- включения (include), <-----
- расширения (extend) - - - - ->
- обобщения (generalization).



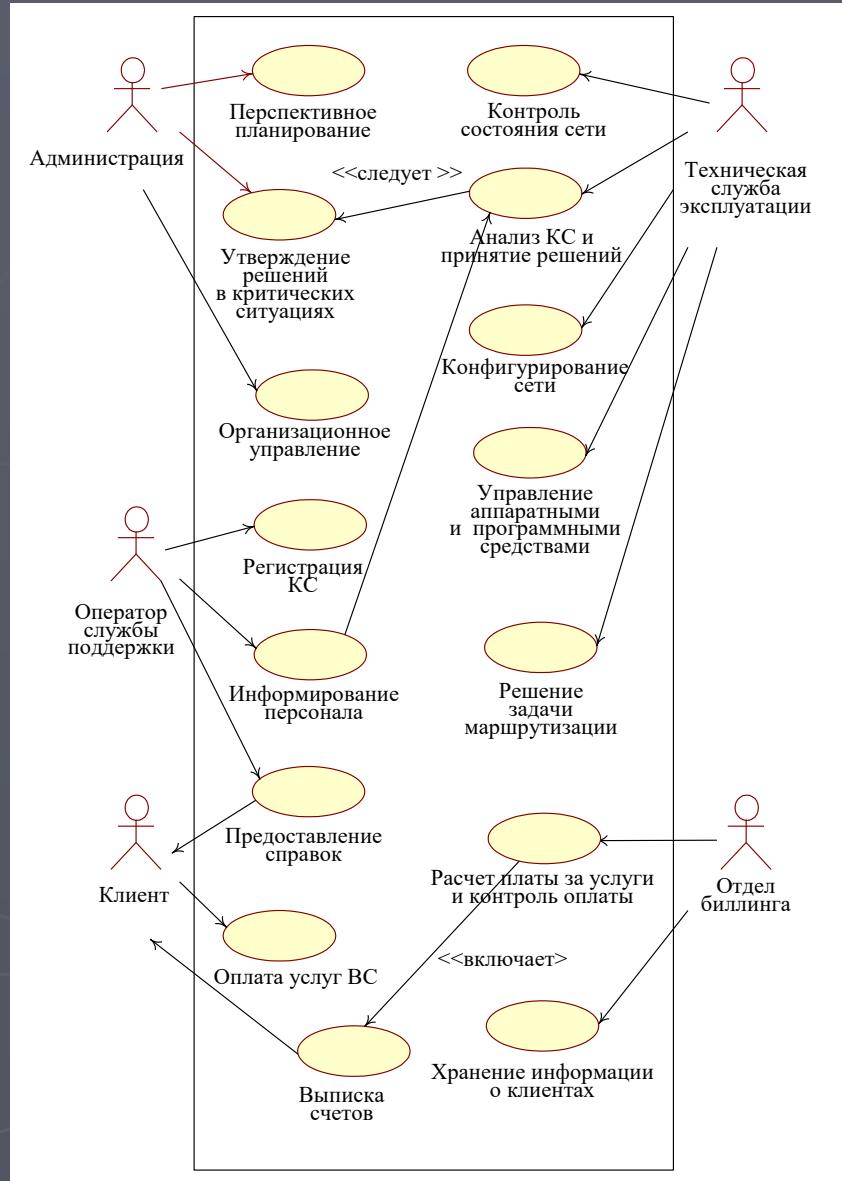
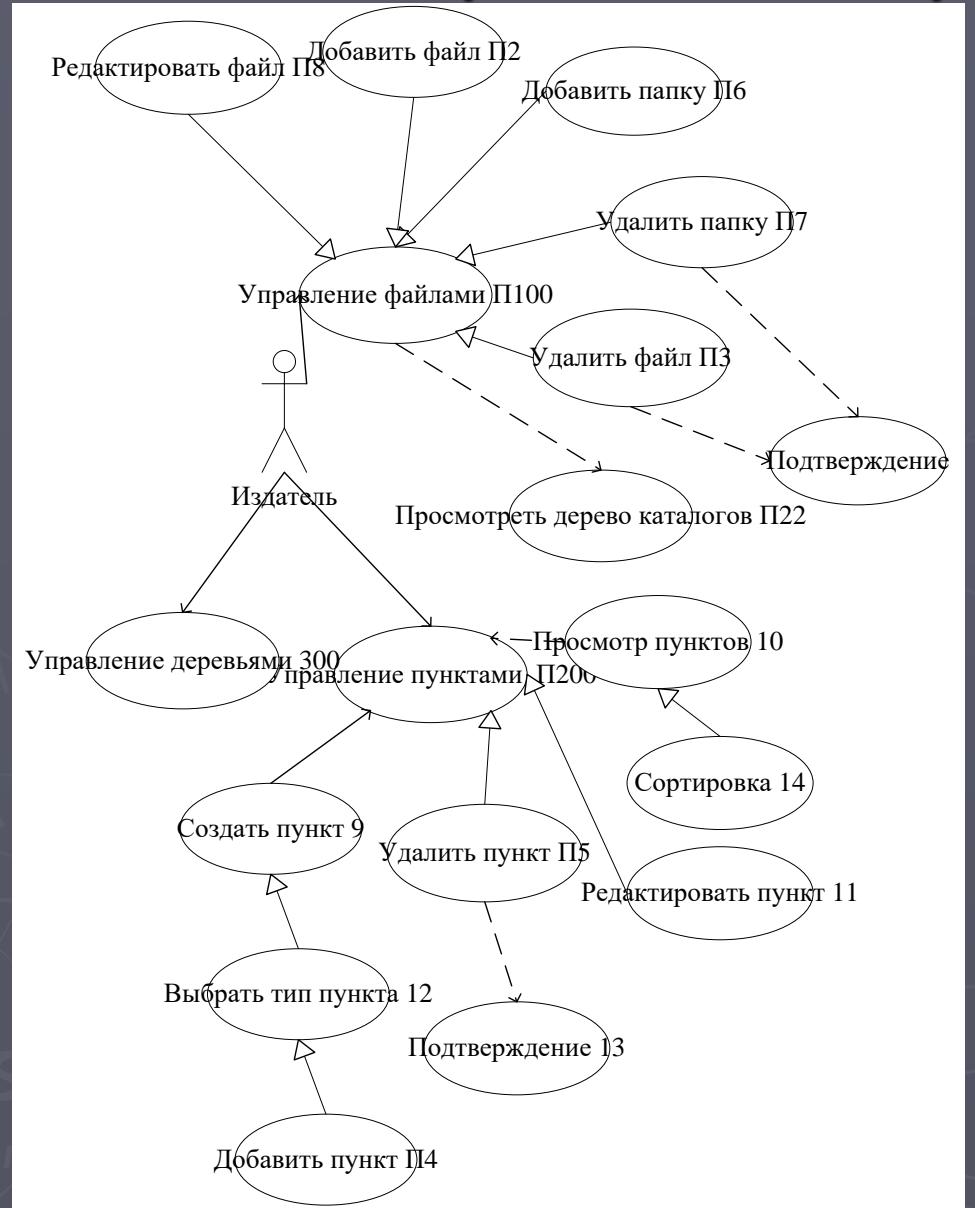
Пример

- ▶ варианты использования – это функции, выполняемые системой
- ▶ действующие лица – это заинтересованные лица по отношению к создаваемой системе





Определение функциональных требований



Вход в систему П1

ПС	«Издатель»
Краткое описание	Пользователь проходит фазу аутентификации
Цель	Только авторизованные пользователи могут использовать функциональность ПС «Издатель»
ID/ Приоритет	П1- UC1
Предусловия	нет
Постусловия	<p>Пользователь получает доступ к главной странице приложения. Имя и пароль не требуется за исключением случаев:</p> <ul style="list-style-type: none">• Пользователь нажал кнопку «Выход»• Пользователь не использовал ПС в течении 30 мин (тайм аут по умолчанию). <p>Если вход выполнен не верно сообщение об ошибке должно отображаться на текущей странице ПС. Количество повторных попыток равно трем.</p>
Актер	Издатель
Описание	Пользователь заполняет поля <i>Имя пользователя</i> и <i>Пароль</i> . Нажимает на кнопку «Вход»
Ограничения	<p>Имя пользователя и пароль не должны превышать 10 символов. Доступ должен быть осуществлен только при совпадении и <i>Имени пользователя</i> и <i>Пароля</i>.</p> <p>Одновременно ПС может использовать только один пользователь.</p>
Расширения	Нет
Включения	Нет
Завершение	По истечении интервала тайм аута должна быть установлена ошибка E_LOG_TIMEOUT. При не верном <i>Имени пользователя</i> и <i>Пароле</i> должна быть установлена ошибка E_LOG_ERROR

ИСПОЛЬЗОВАНИЕ ДИАГРАММ НА ЭТАПЕ ЖЦ «ПРОЕКТИРОВАНИЕ»

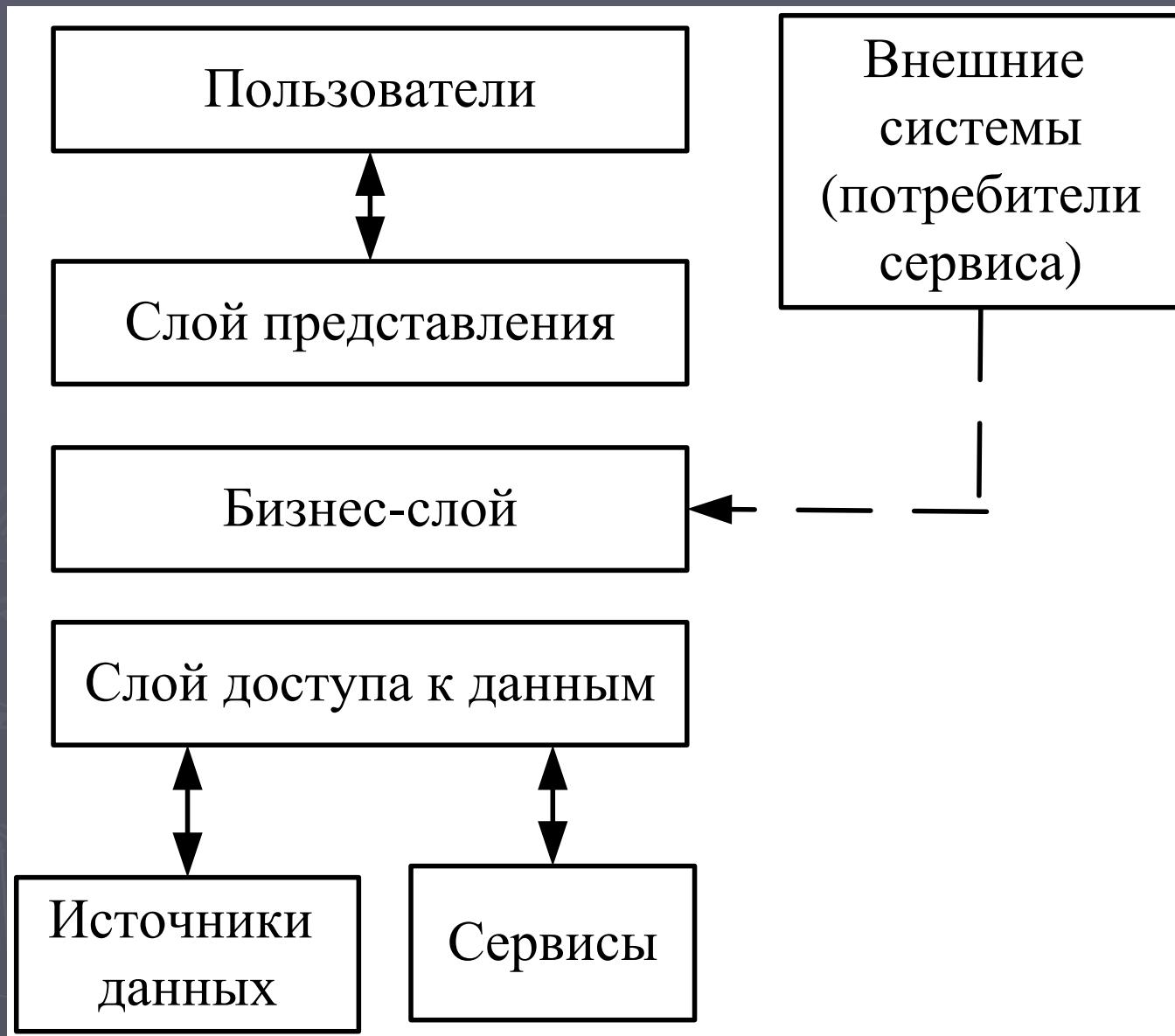
► предварительное проектирование

- формирует абстракции архитектурного уровня
- интерфейсное проектирование
- *Структурирование системы*
- *Декомпозиция подсистем на модули*

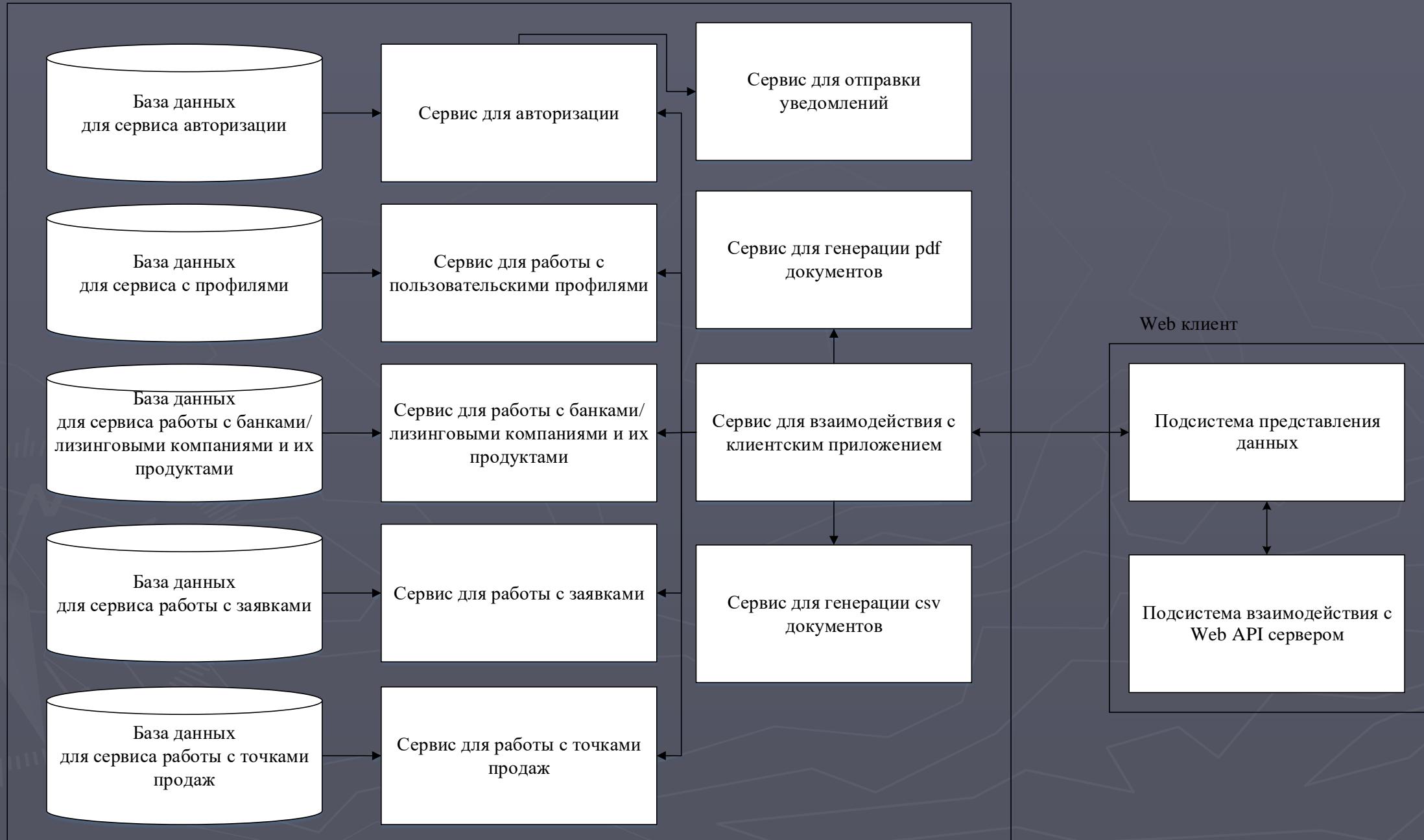


► детальное проектирование

Логическое разделение на слои



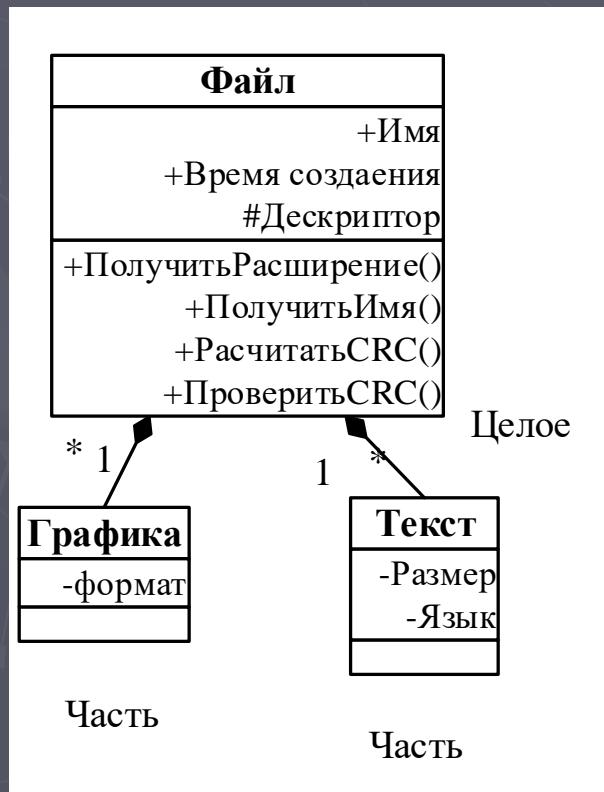
Web API Сервер



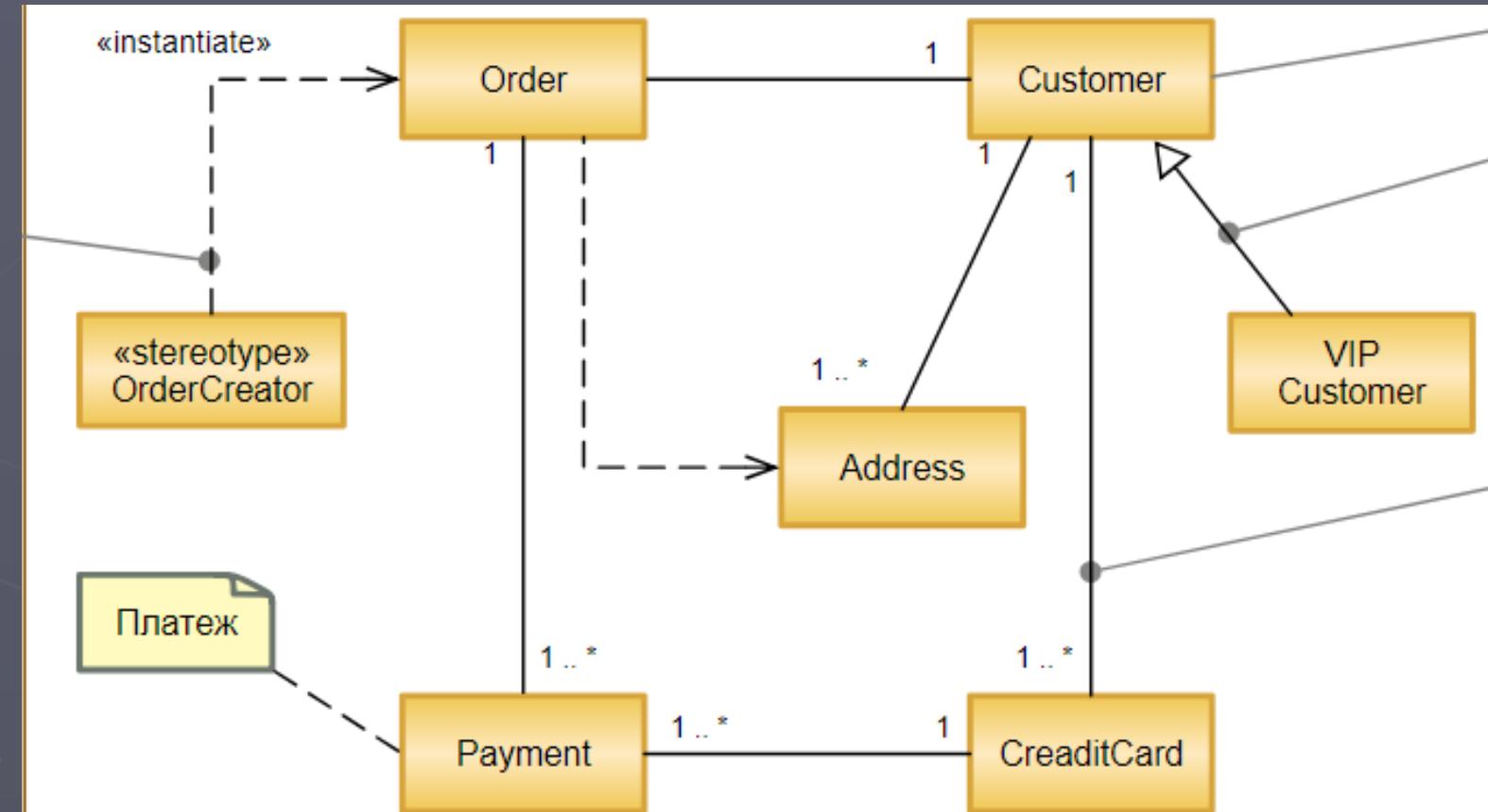
А) Статическое проектирование

Диаграммы классов

- концептуальный уровень
- уровень спецификаций
- уровень реализации



- ▶ **тип сущностей:**
классы (включая интерфейсы, примитивные типы, классы-ассоциации и другие)
- ▶ **типы отношений:**
- ▶ ассоциация между классами
- ▶ обобщение между классами ;
- ▶ зависимости (различных типов)



Классы

секция имени

видимость ИМЯ кратность : тип = начальное_значение {свойства}

секция атрибутов

видимость ИМЯ (параметры) : тип {свойства}

секция операций

стандартные стереотипы классов

Стереотип	Описание
«actor»	Действующее лицо
«auxiliary»	Вспомогательный класс
«enumeration»	Перечислимый тип данных
«exception»	Исключение (только в UML 1)
«focus»	Основной класс
«implementationClass»	Реализация класса
«interface»	Все составляющие абстрактные
«metaclass»	Экземпляры являются классами
«powertype»	Метакласс, экземплярами которого являются все наследники данного класса (только в UML 1)
«process»	Активный класс
«thread»	Активный класс (только в UML 1)
«signal»	Класс, экземплярами которого являются сигналы
«stereotype»	Новый элемент на основе существующего
«type»	Тип данных
«dataType»	Тип данных
«utility»	Нет экземпляров, служебно

ClassName

+ attribute
- privateAttr
- fio : String = "Novikov"
- array : char [10]

+ operationName ()
- staticOperation ()

+ function () : int

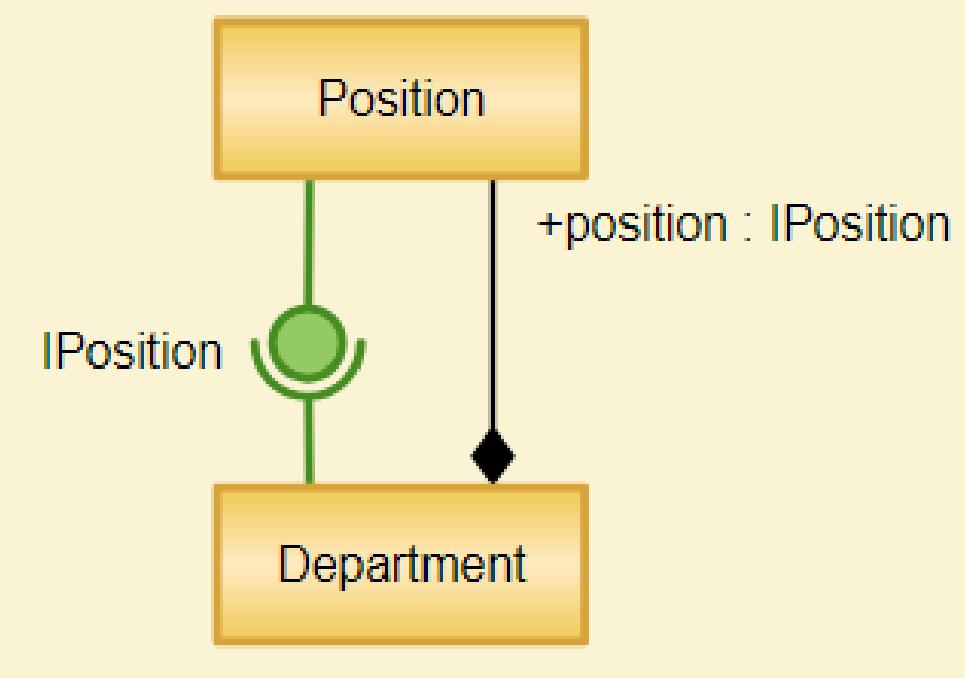
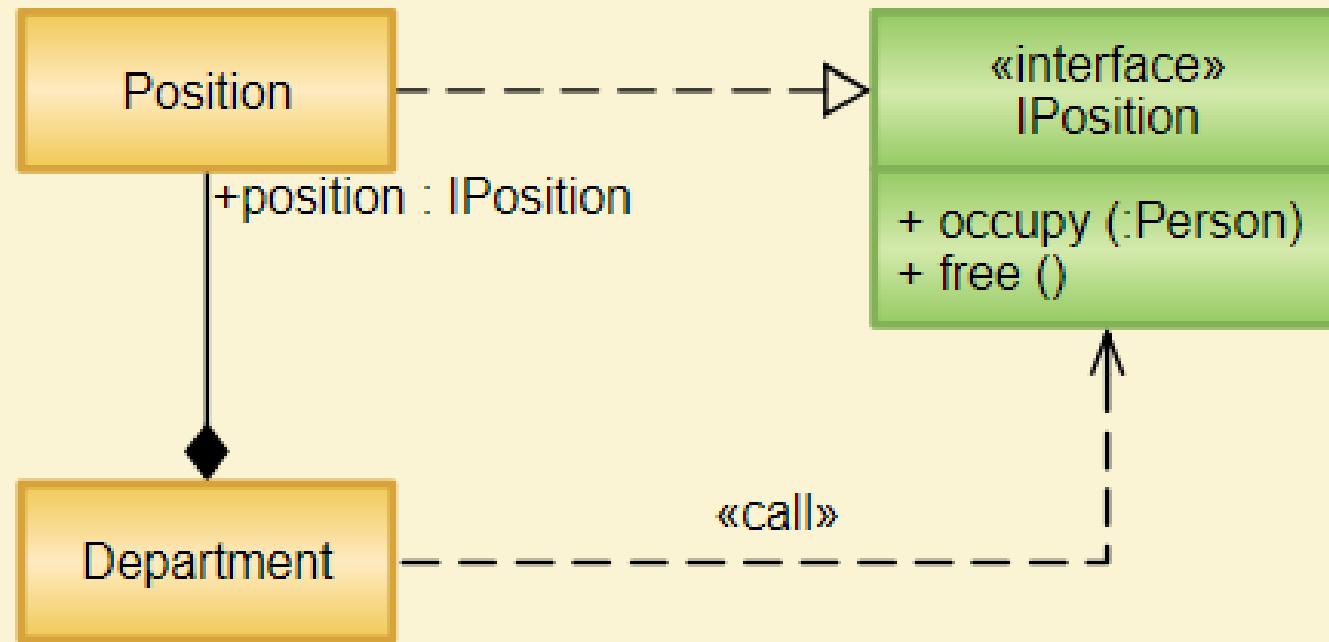
«utility»
Company

Graphic
Draw()
Add(Graphic)
Remove(Graphic)
GetChild(int)

- **открытый** (обозначается знаком `+` или ключевым словом `public`);
- **защищенный** (обозначается знаком `#` или ключевым словом `protected`);
- **закрытый** (обозначается знаком `-` или ключевым словом `private`).
- **пакетный** (обозначается знаком `~` или ключевым словом `package`).

► Отношения на диаграмме классов

1) Отношения зависимости и реализации

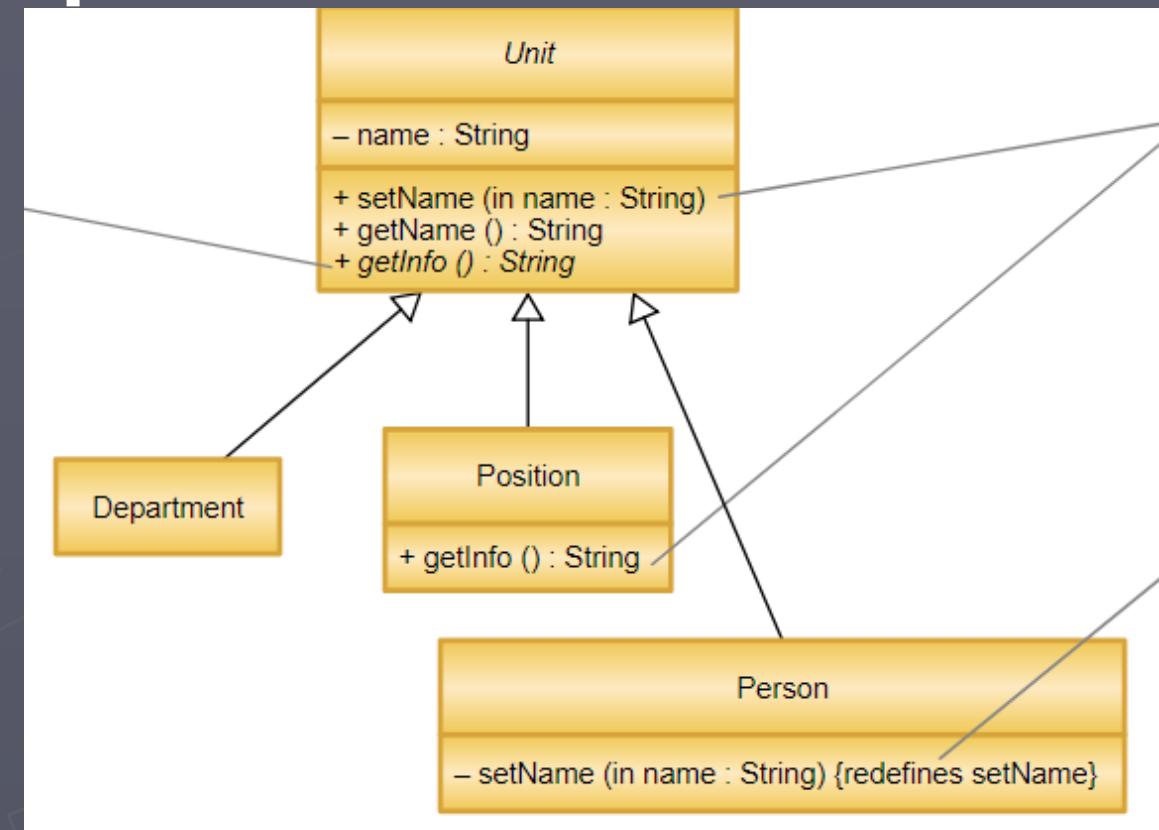
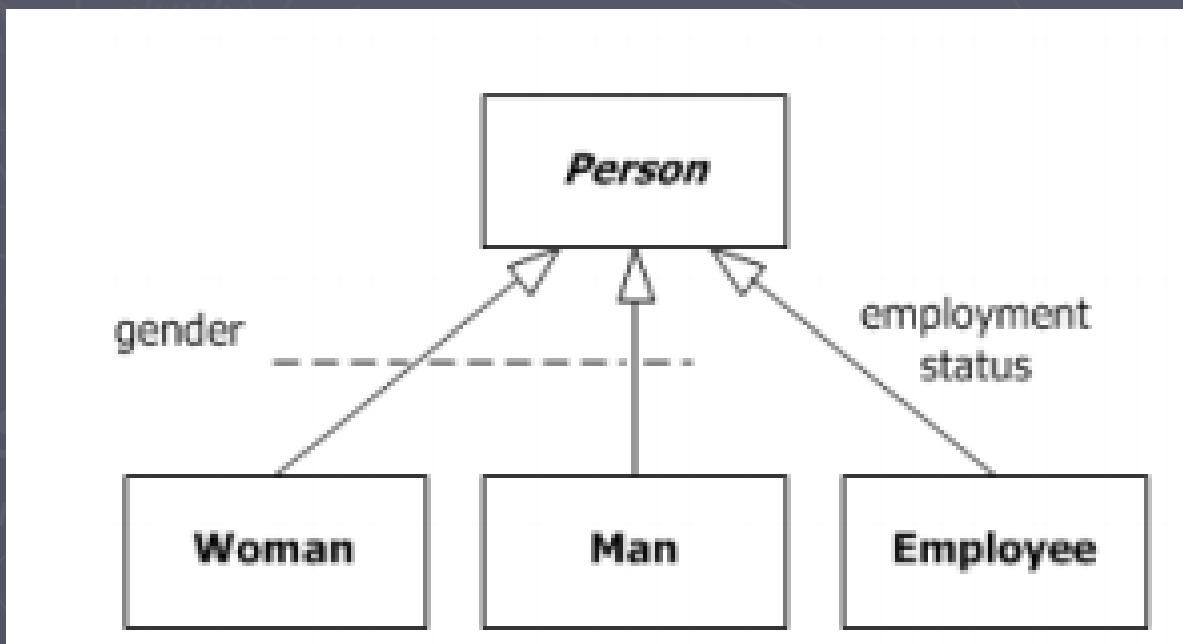


► 2) Отношение обобщения

Генерализация-расширение-наследование

показывают связи наследования между классами

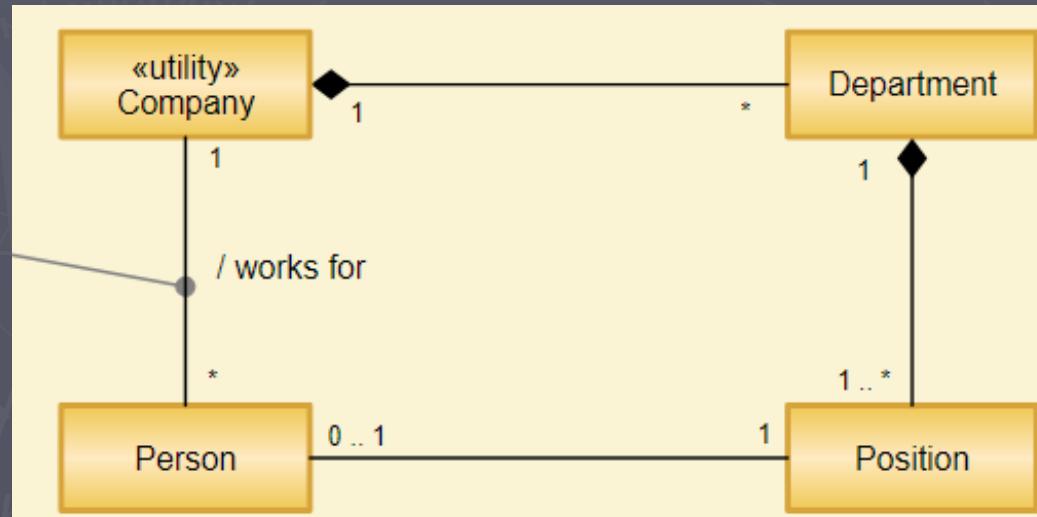
Является IS A



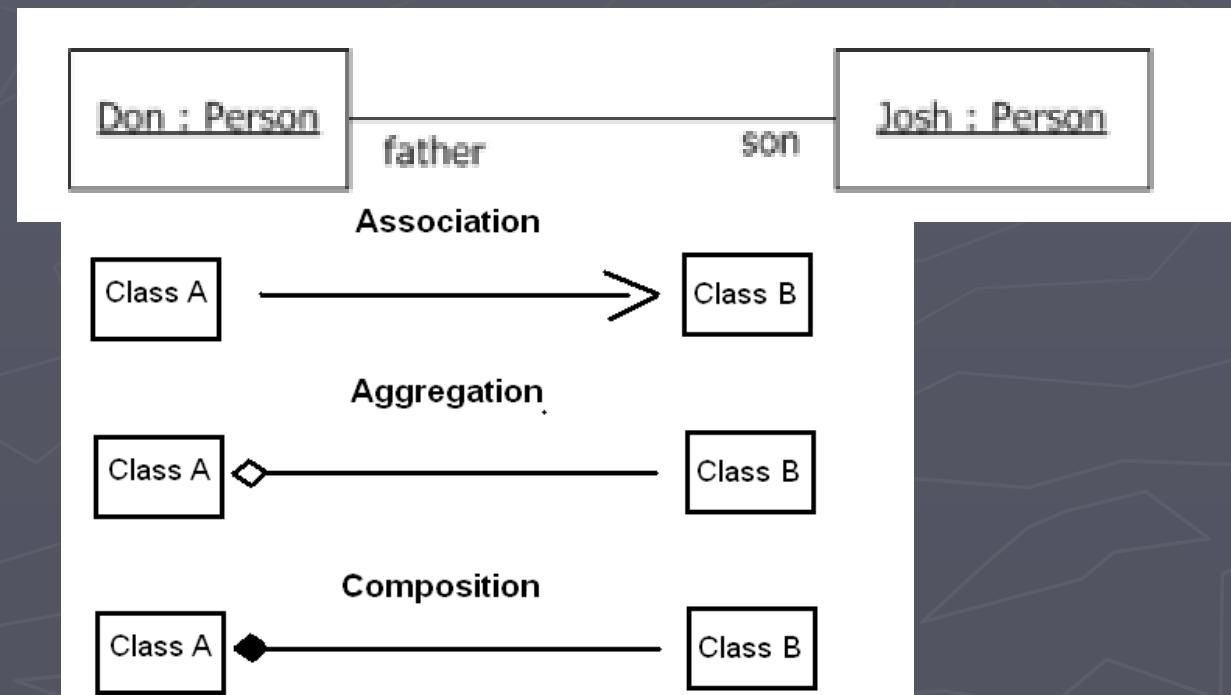
► 3) Ассоциации и их дополнения

экземпляры одного класса связаны с экземплярами другого класса

- Имя ассоциации
- Кратность полюса ассоциации
- Агрегация и композиция



Ассоциация показывает отношения между объектами-экземплярами класса
дву направленные или односторонние



+ Employee

- position : String
- card : IdCard
- room : Room [1..*]
- department : Department
- pastPosition : PastPosition [0..*]
+ Employee(n:String,s:String,p:String)
+ setPosition(newPosition:String)
+ getPosition() :String
+ setIdCard(newIdCard:IdCard)
+ getIdCard() :IdCard
+ setRoom(newRoom:Room)
+ getRoom() :Room[1..*]
+ deleteRoom(r:Room)
+ setDepartment(d:Department)
+ getDepartment() :Department
+ setPastPosition(p:PastPosition)
+ getPastPosition() :PastPosition[1..*]
+ deletePastPosition(p:PastPosition)

+ выдана

+ 1..1 + 1..1

+ IdCard

- number : int
- dateExpire : String
+ IdCard(n:int)
+ setNumber(newNumber:int)
+ getNumber() :int
+ setDateExpire(newDate:Date)
+ getDateExpire() :Date

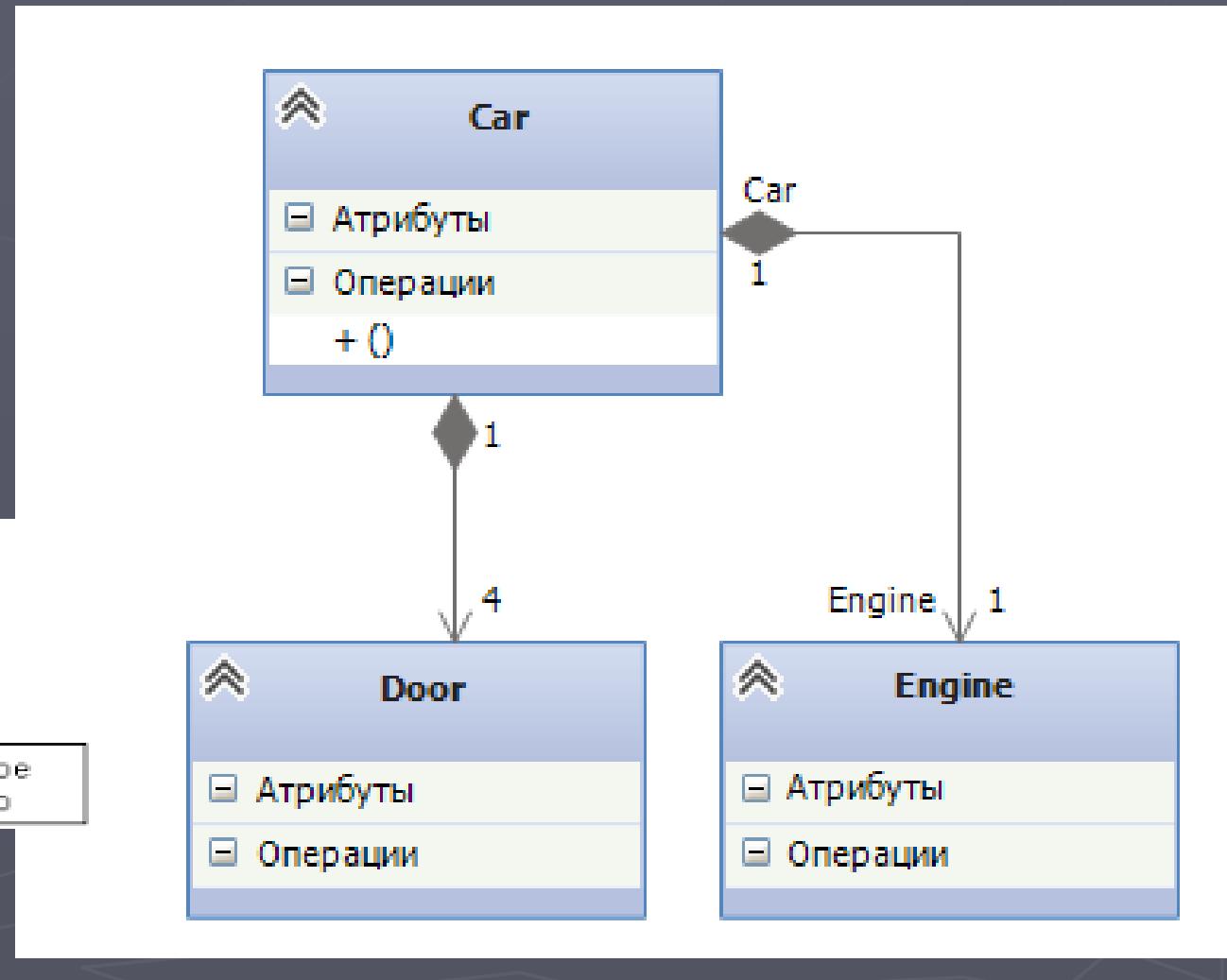
Композиция

Состоит из

Отношение «содержит как часть»

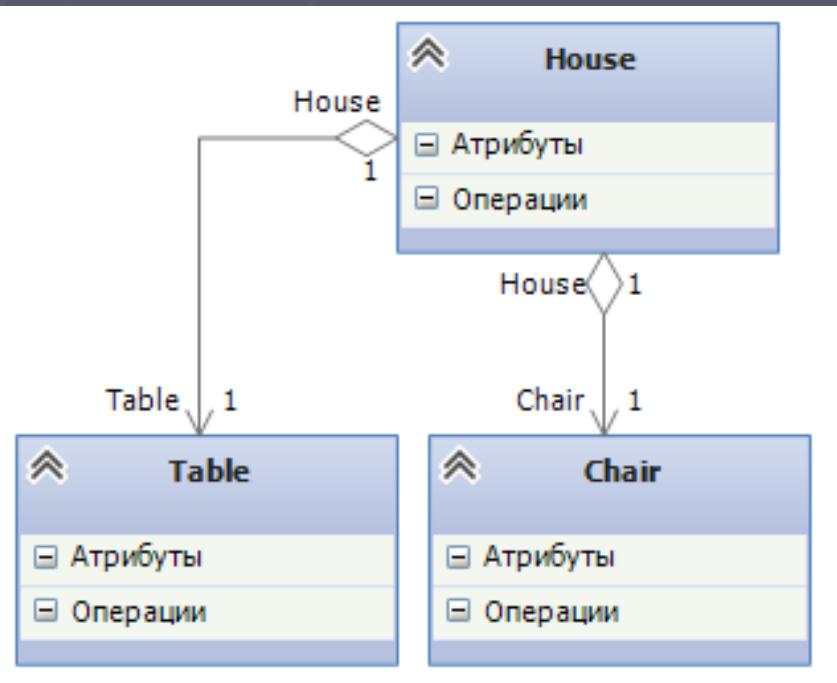
HAS-A

Целое контролирует
время жизни своей
составной части (часть
не существует без целого)
- сильно связана



Агрегация

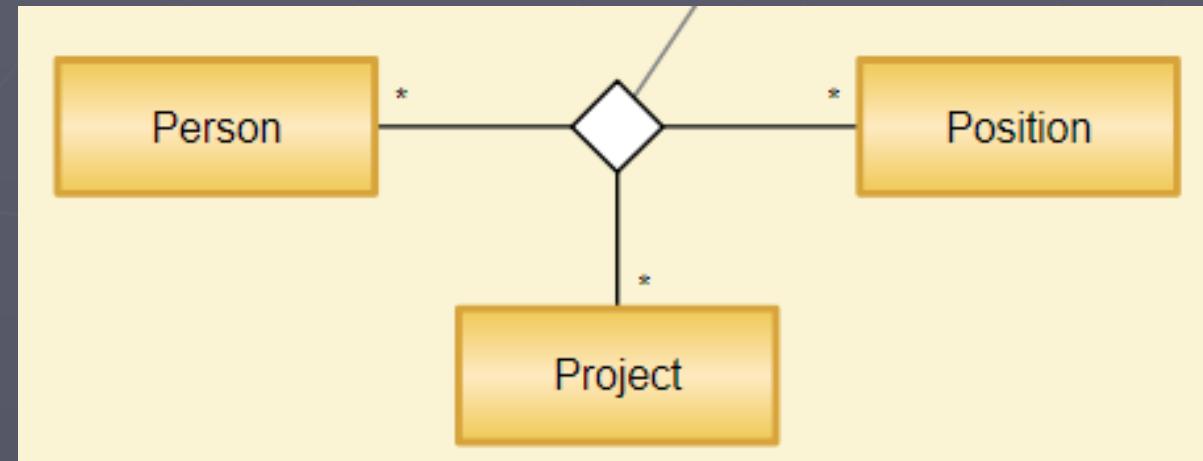
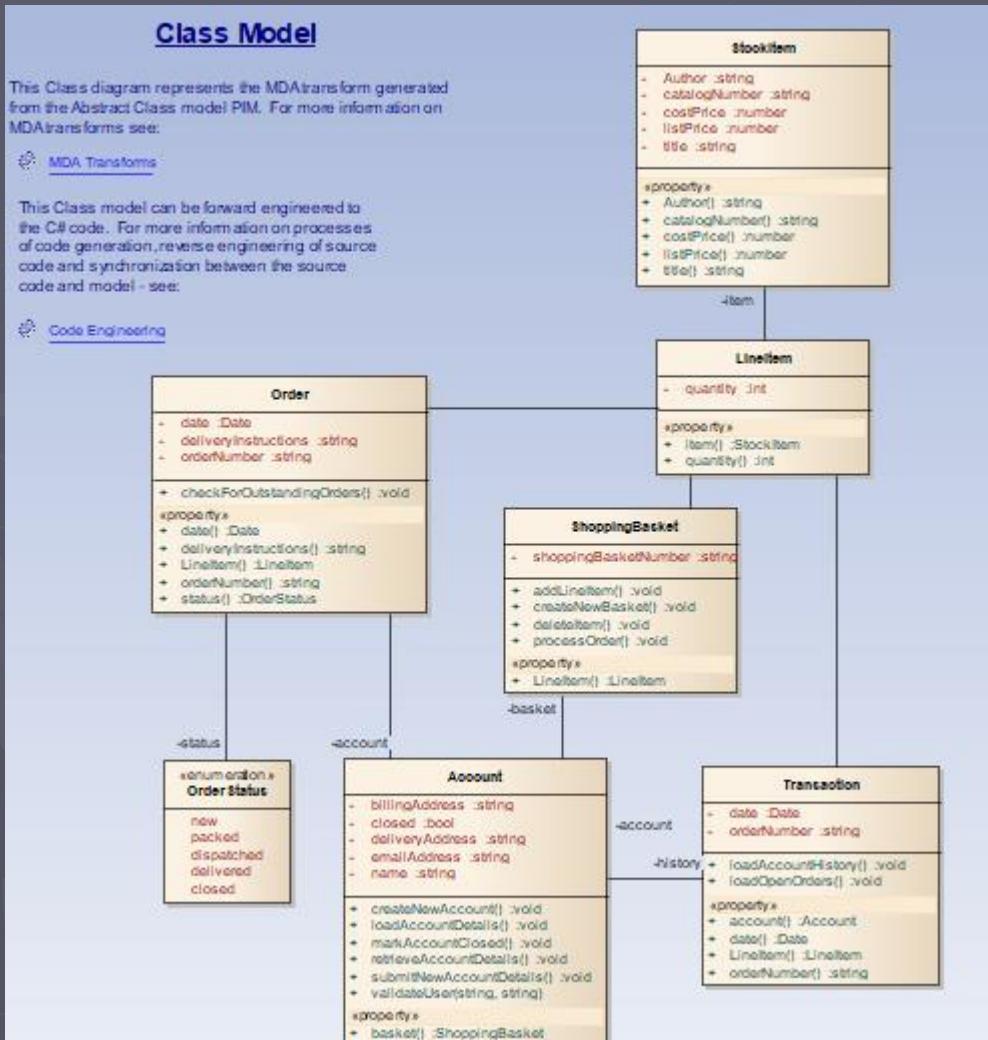
- ▶ СВЯЗЬ МЕЖДУ ЦЕЛЫМ И ЕГО ЧАСТЬЮ
- ▶ Включает в себя - contains
- ▶ Объекты м.б. равноправные

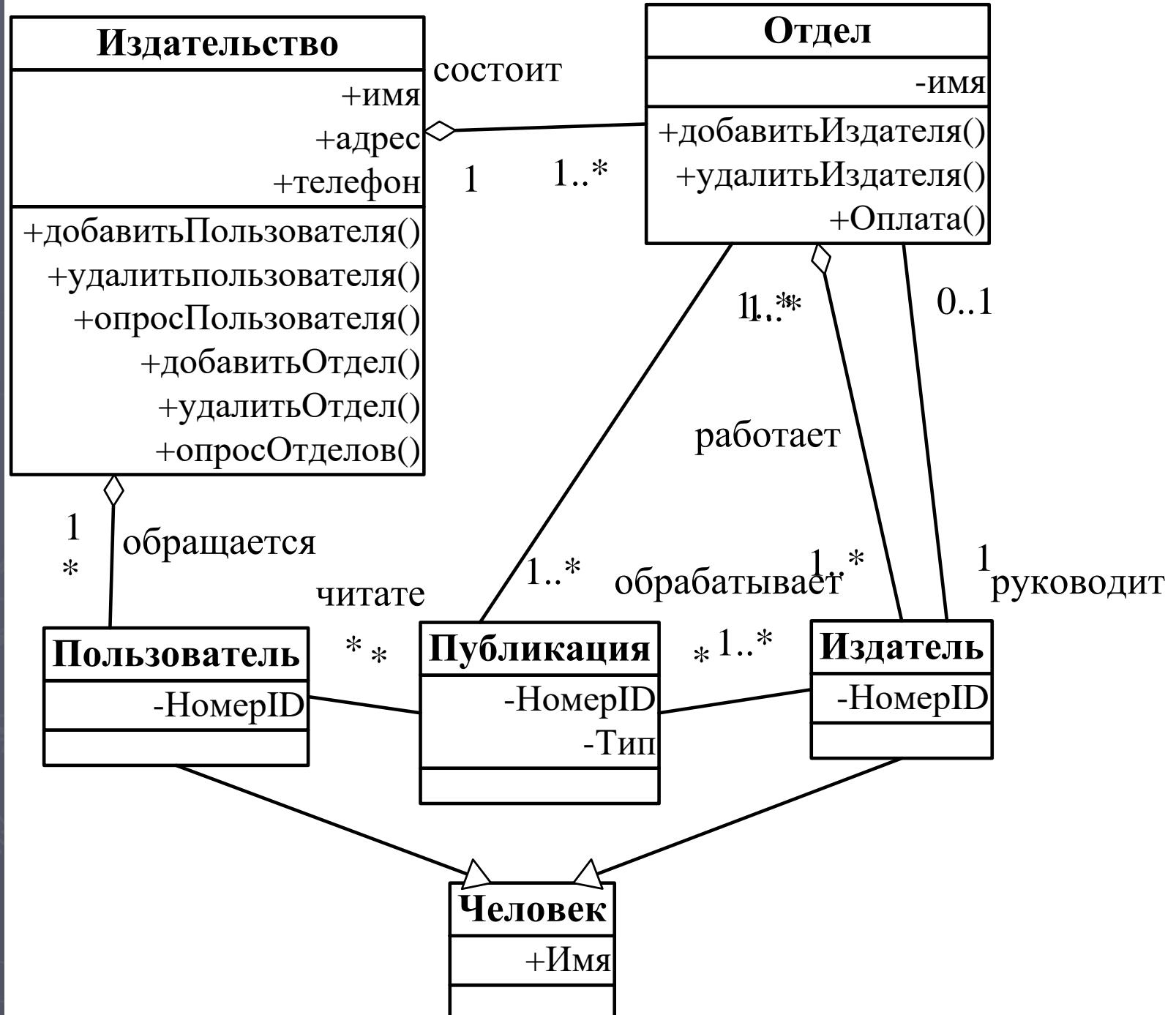


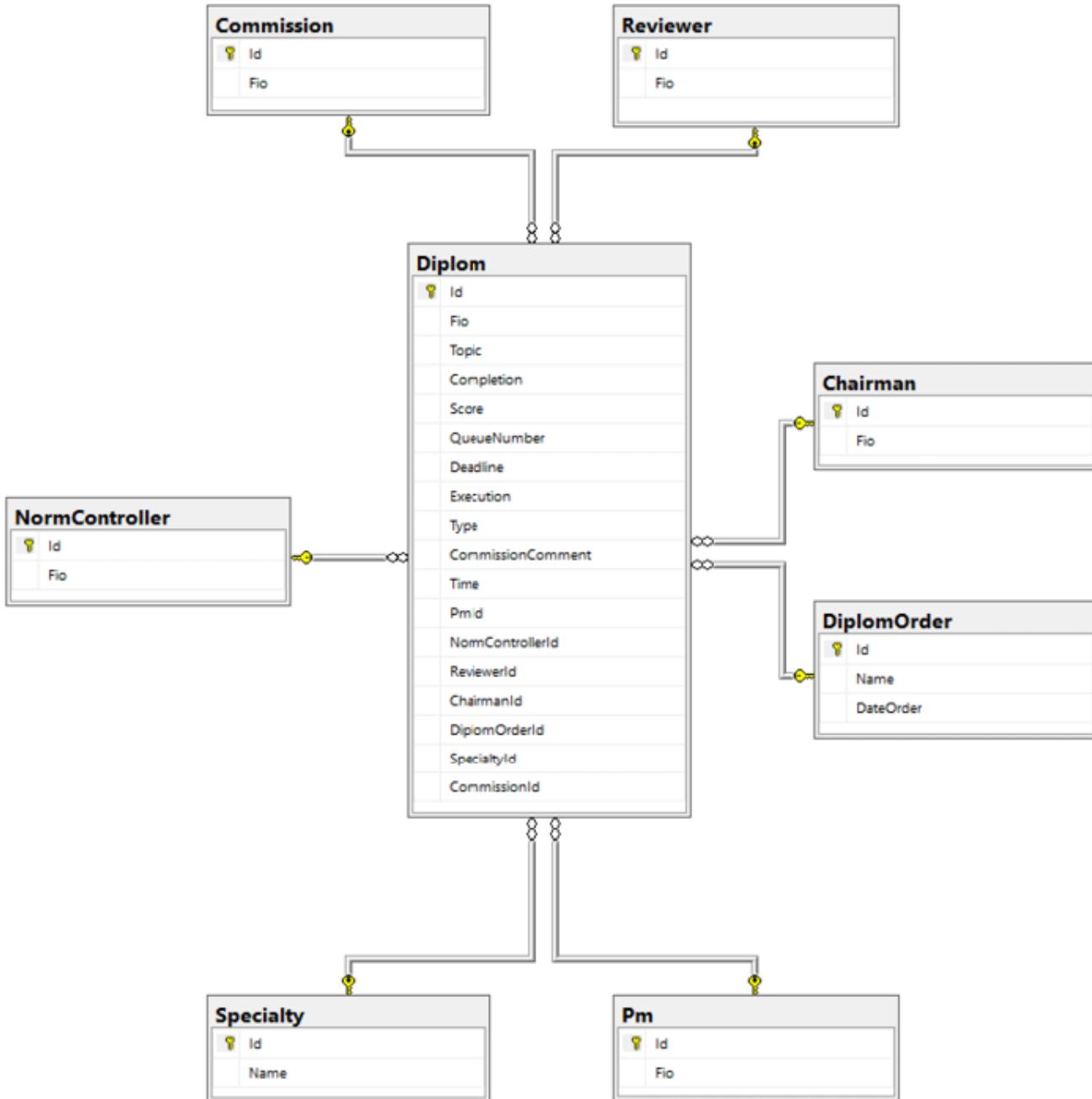
+ Department	+ относится к	+ Employee
- name : String - employees : Employee [1..*] + Department(n:String) + setName(newName:String) + getName() :String + addEmployee(newEmployee:Employee) + getEmployees() :Employee[0..*] + removeEmployee(e:Employee)	+ 1..1 + 0..*	- position : String - card : IdCard - room : Room [1..*] - department : Department - pastPosition : PastPosition [0..*] + Employee(n:String,s:String,p:String) + setPosition(newPosition:String) + getPosition() :String + setIdCard(newIdCard:IdCard) + getIdCard() :IdCard + setRoom(newRoom:Room) + getRoom() :Room[1..*] + deleteRoom(r:Room) + setDepartment(d:Department) + getDepartment() :Department + setPastPosition(p:PastPosition)

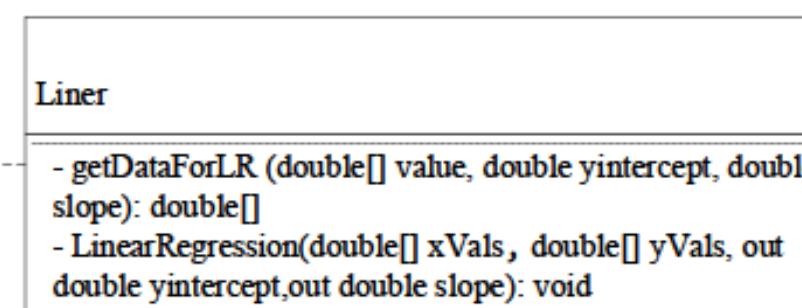
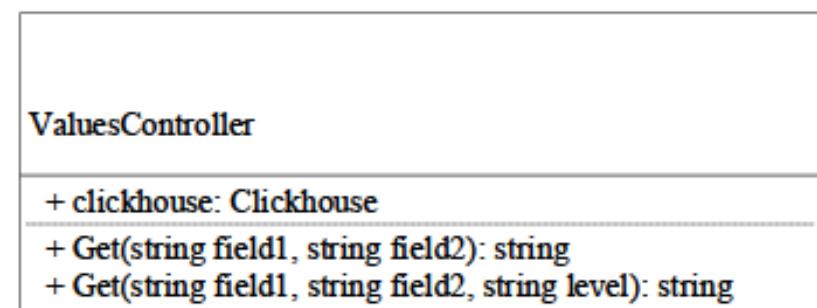
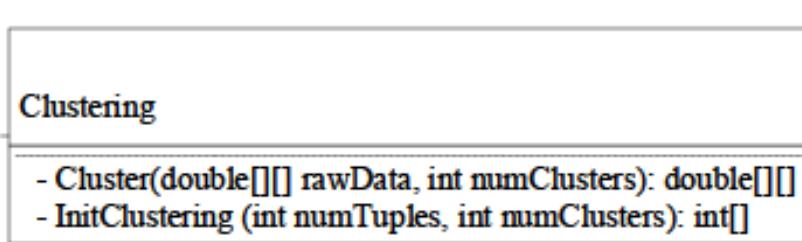
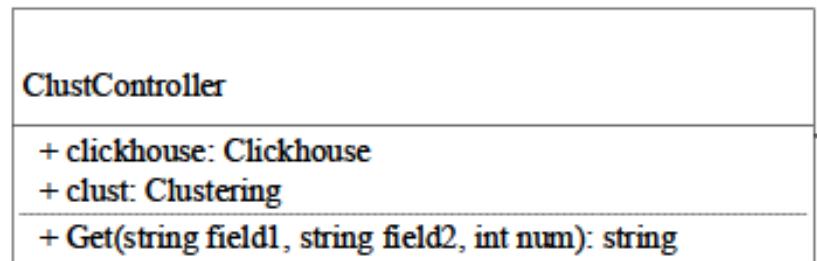
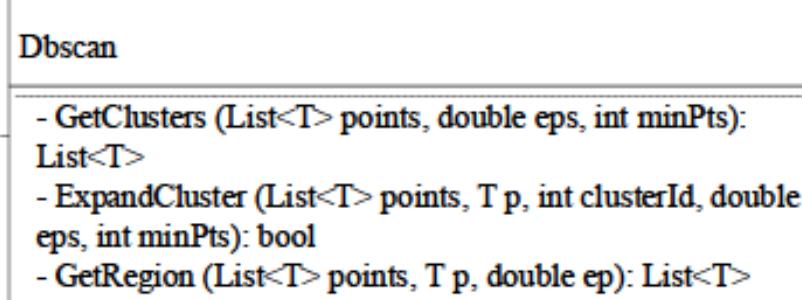
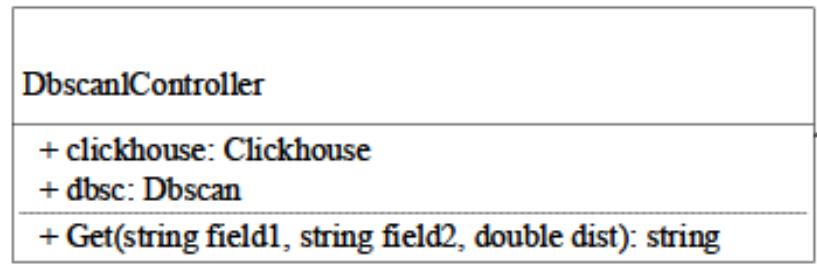
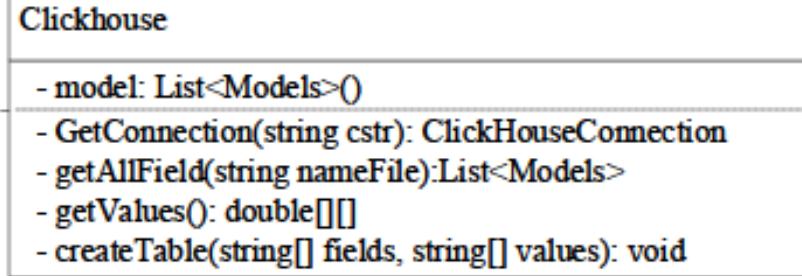
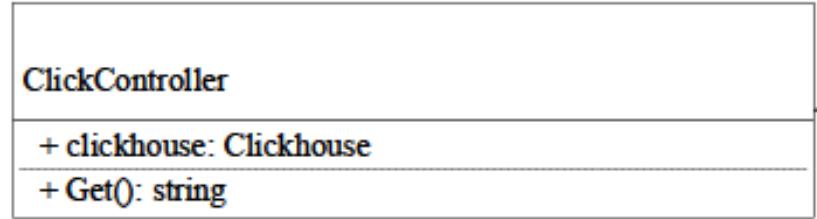
Целое хоть и содержит свою
составную часть, время их жизни
не связано

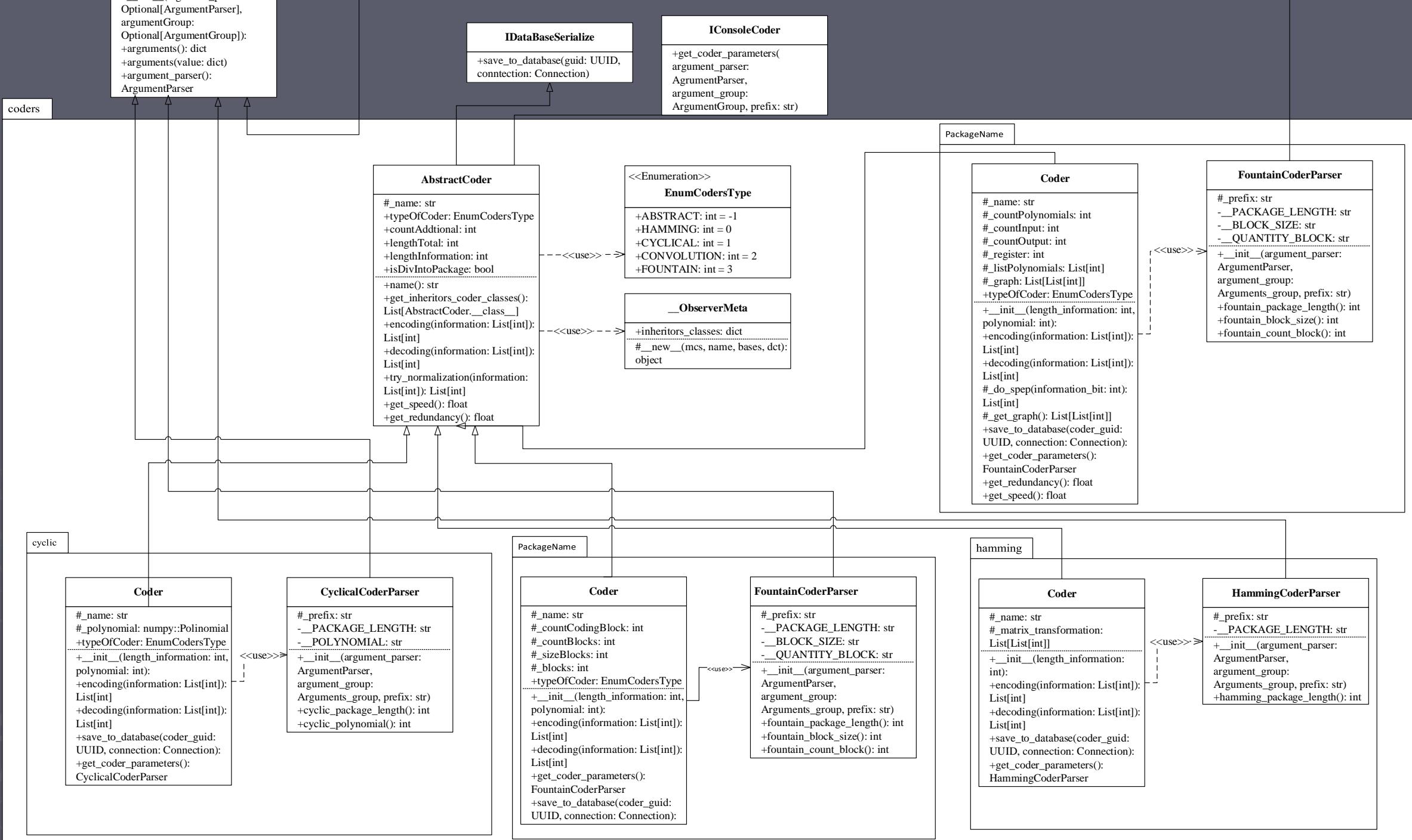
► Многополюсная ассоциации







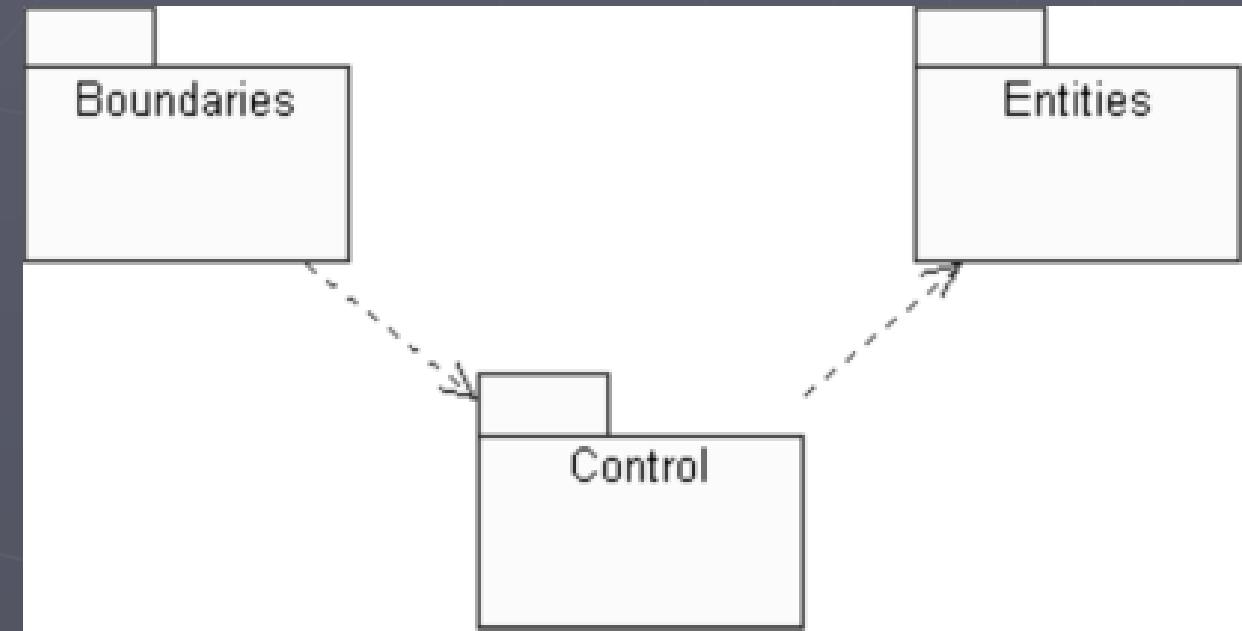




Стереотипы классов

механизм , позволяющий разделять классы на категории

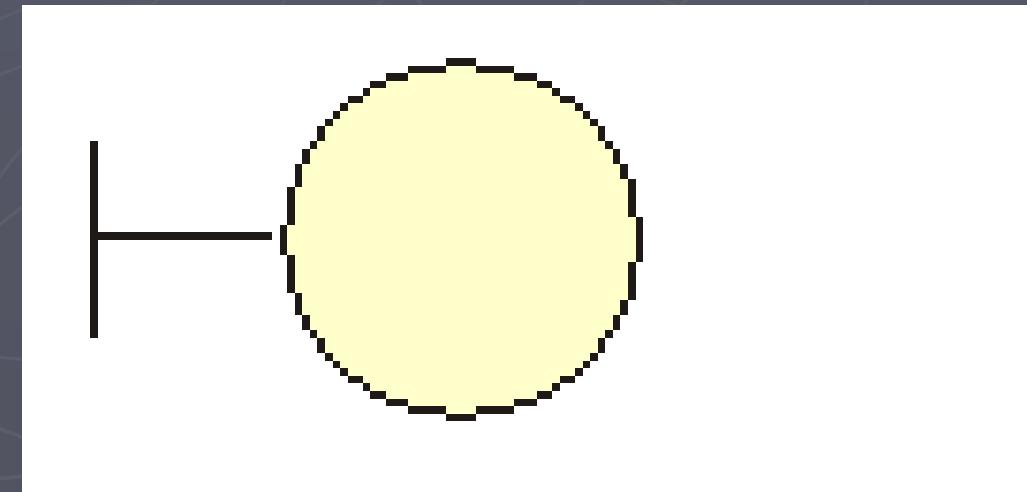
- ▶ Boundary (граница)
- ▶ Entity (сущность)
- ▶ Control (управление).



1) Границные классы

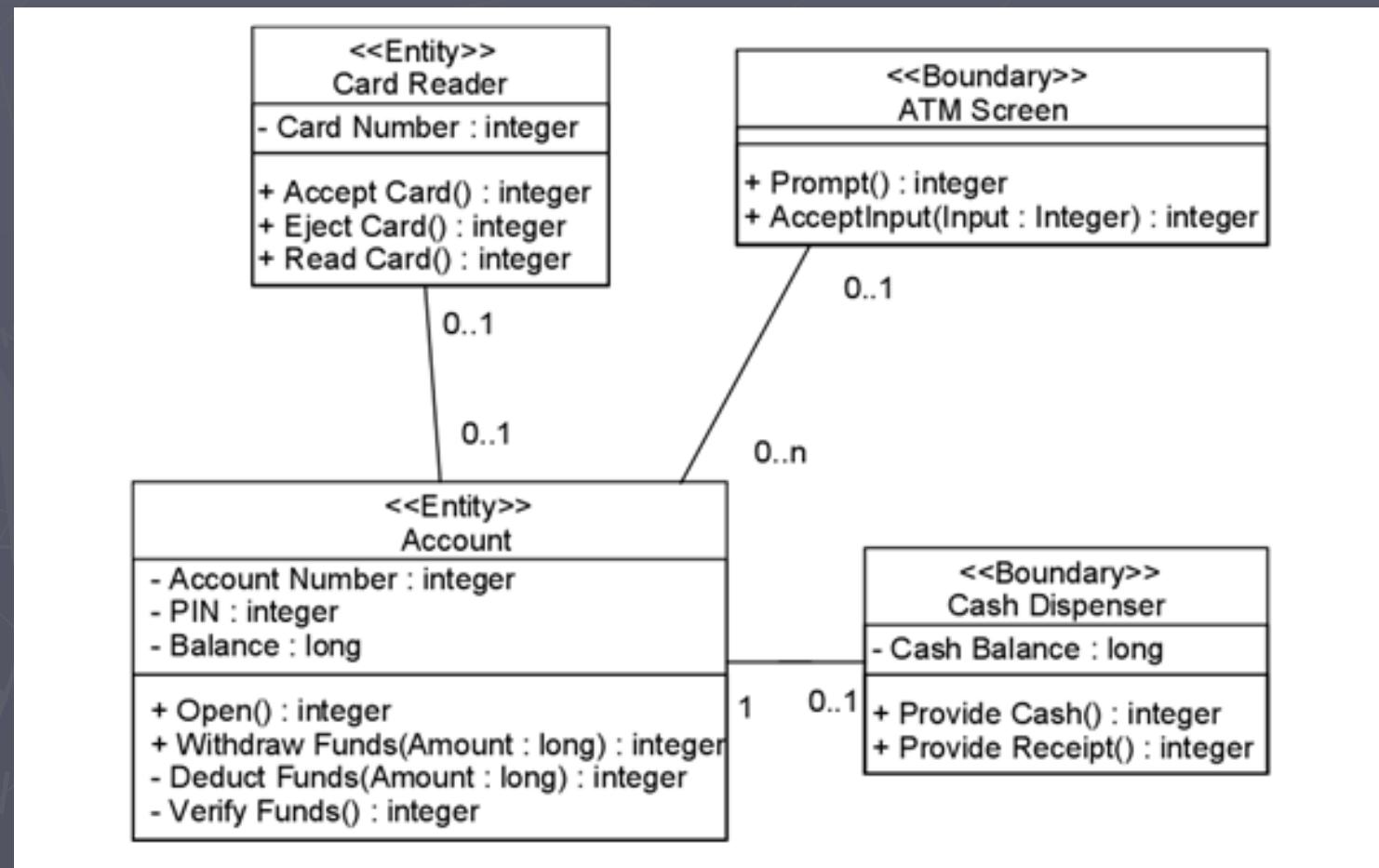
расположены на границе системы и всей
окружающей среды

- ▶ экранные формы ,
- ▶ отчеты ,
- ▶ интерфейсы с аппаратурой
- ▶ интерфейсы с другими системами



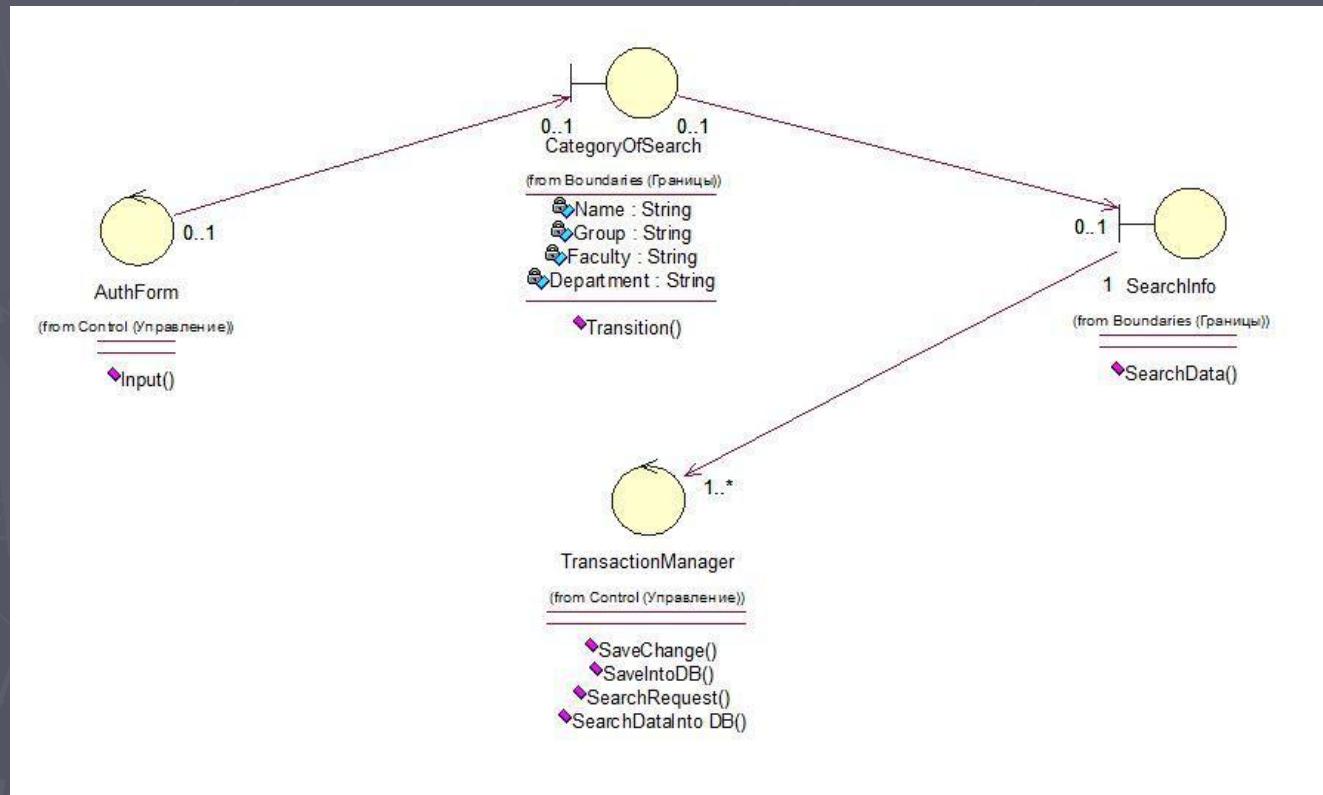
2) Классы –сущности

содержат хранимую информацию



3) Управляющие классы

отвечают за координацию действий других классов



Б) Динамическое проектирование

Диаграммы последовательности (sequence diagram)

- Диаграмма последовательности используется для представления временных особенностей передачи и приема сообщений между объектами.

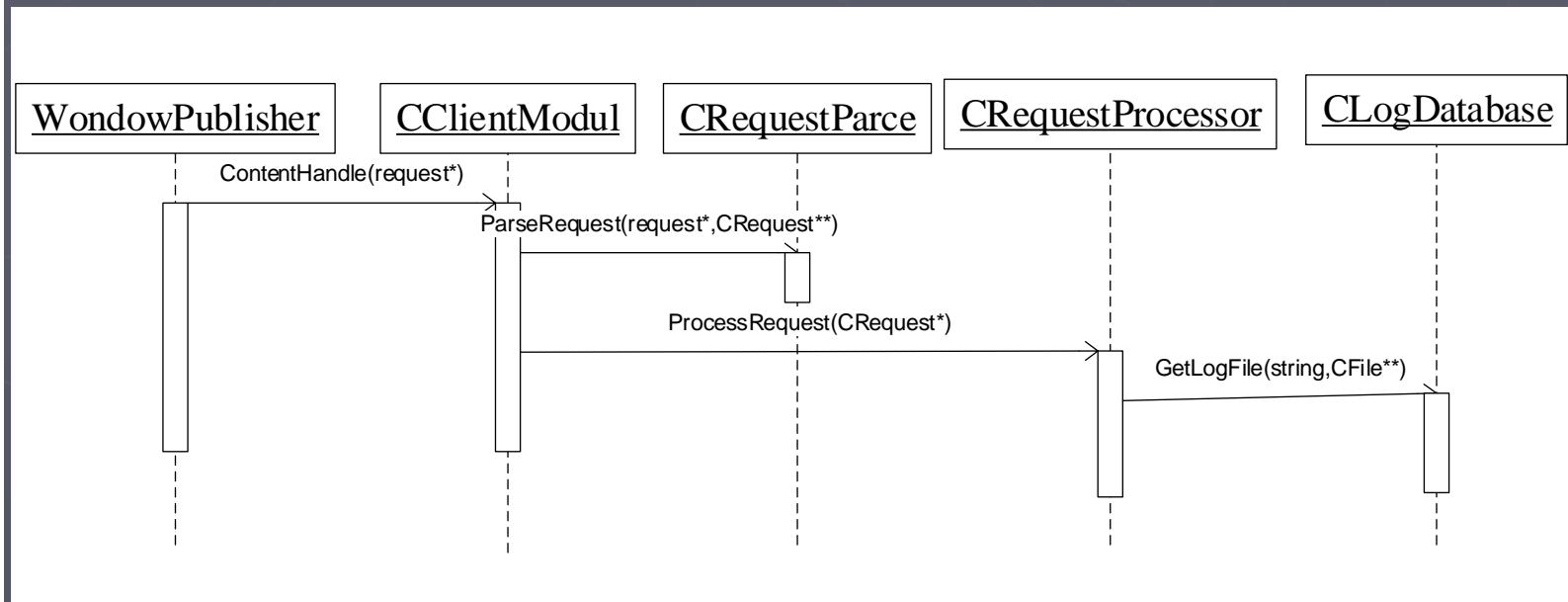
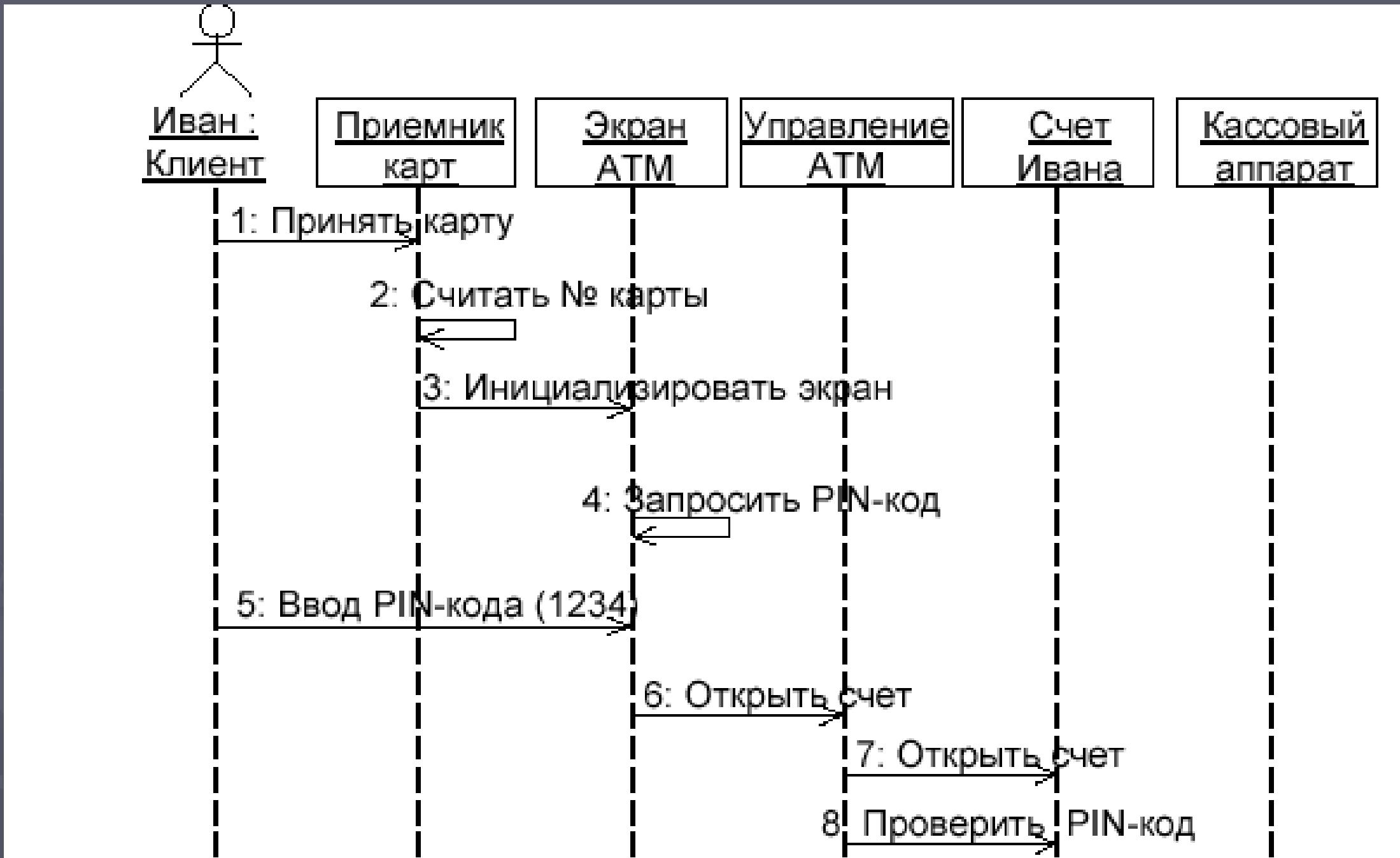


Диаграмма последовательности (sequence diagram) – это способ описания поведения системы на основе указания последовательности передаваемых сообщений.



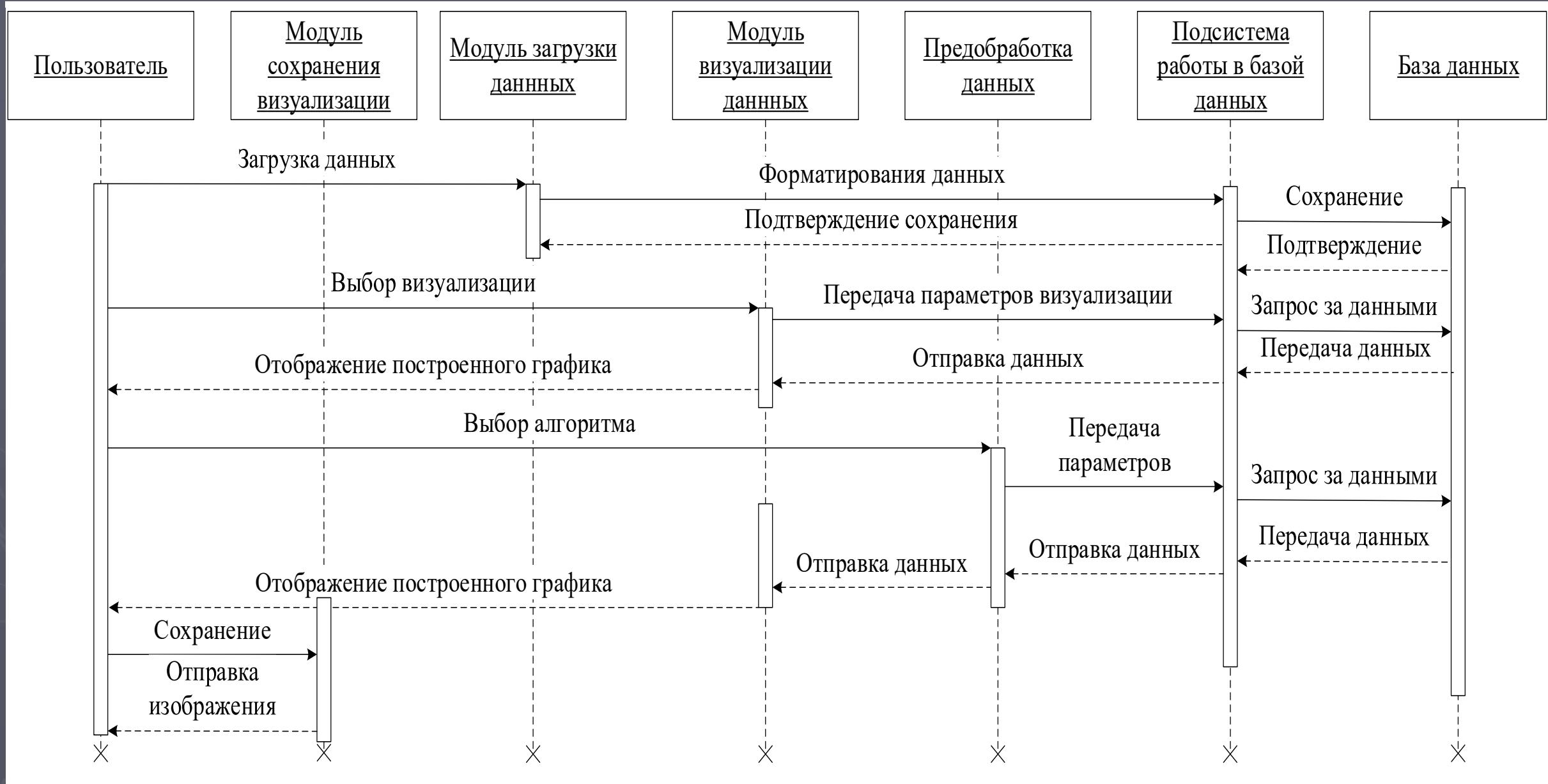


Диаграмма коммуникации (Communication diagram)

- отображают поток событий через конкретный сценарий варианта использования
- легче понять связи между объектами , однако , труднее уяснить последовательность событий

Условие

$[x > 0]$ 2.1: f3()

итерация

*[i:=1 .. n] 1: f1()

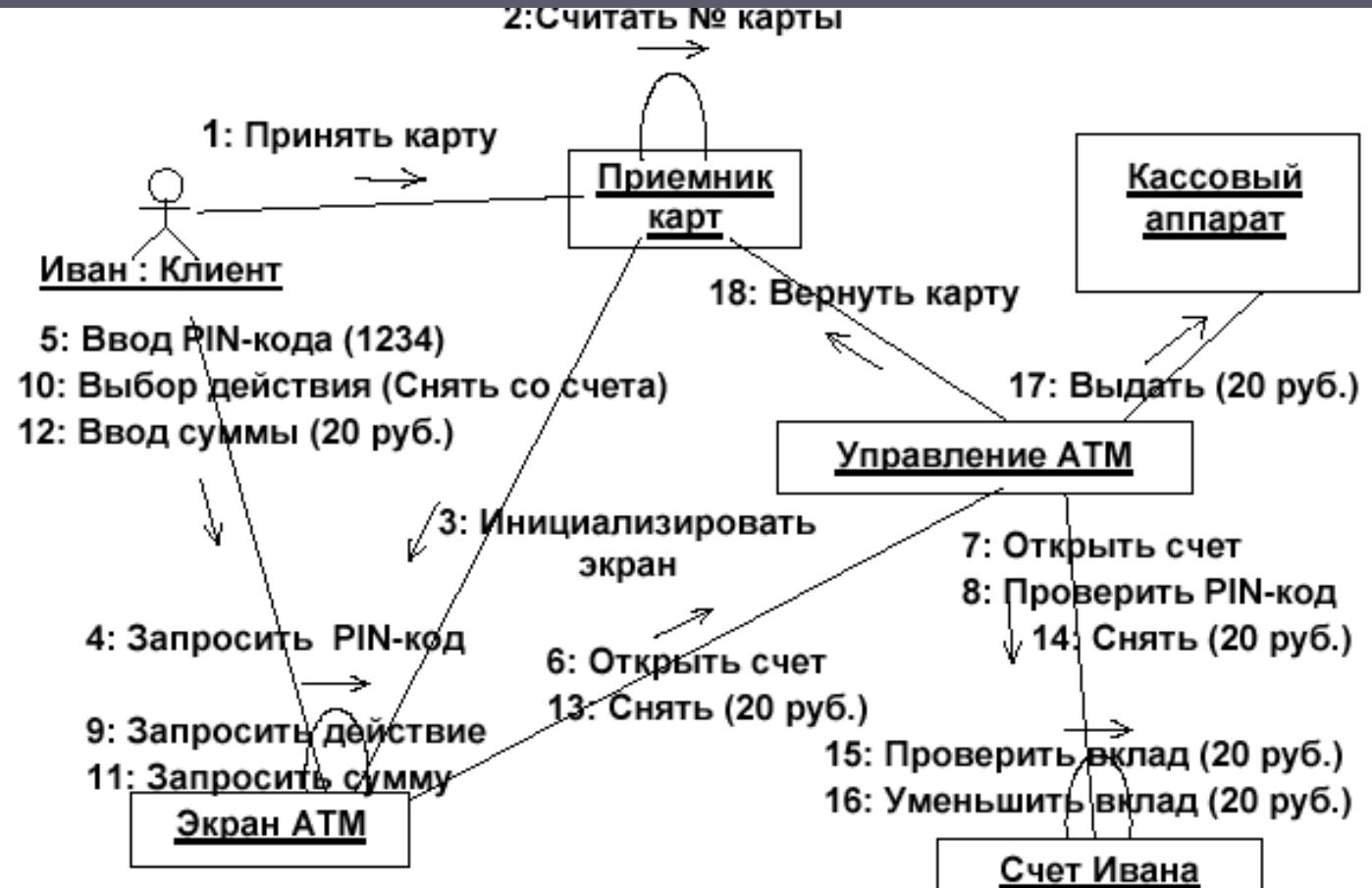
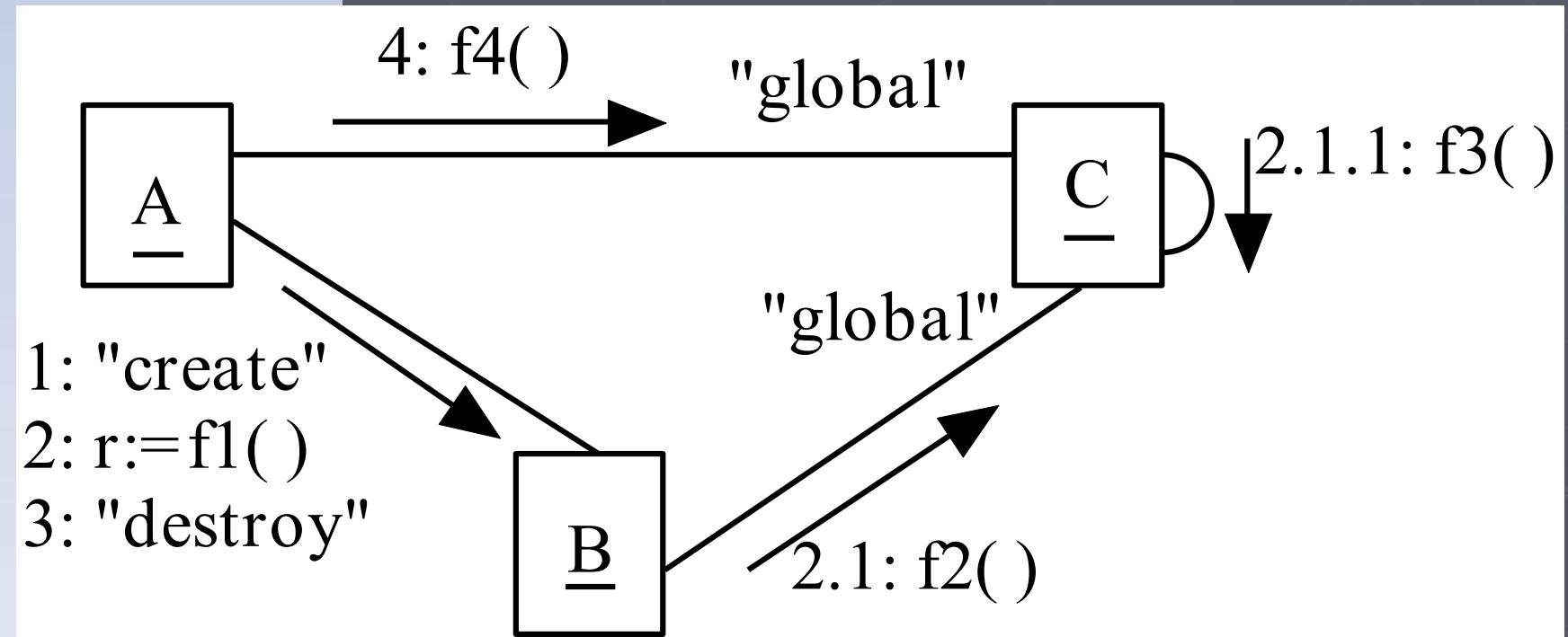
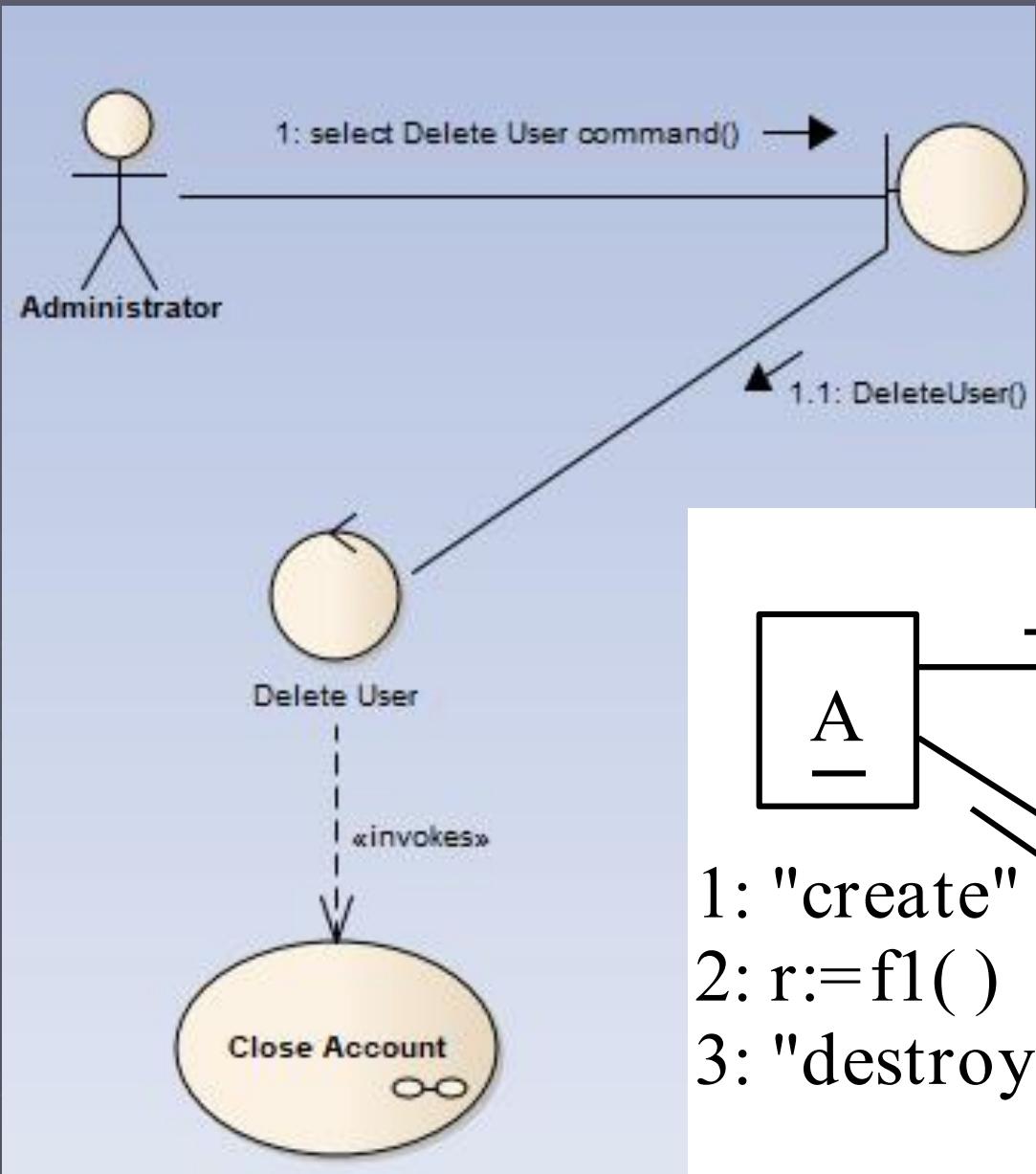


Диаграмма коммуникации (communication diagram) – способ описания поведения, семантически эквивалентный диаграмме последовательности.

Примеры



Диаграммы автомата - state machine diagram

Диаграммы деятельности (activity diagram)

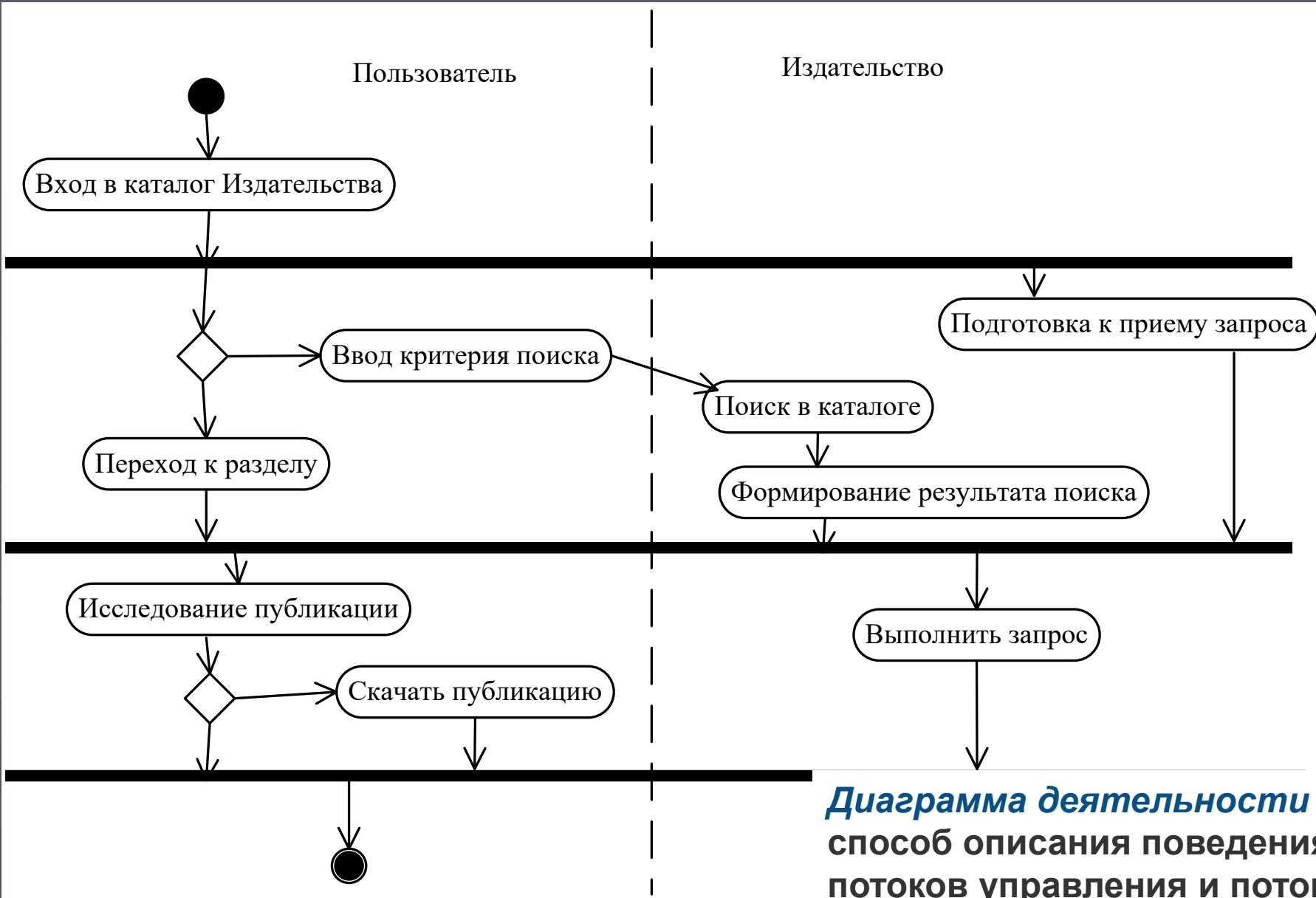
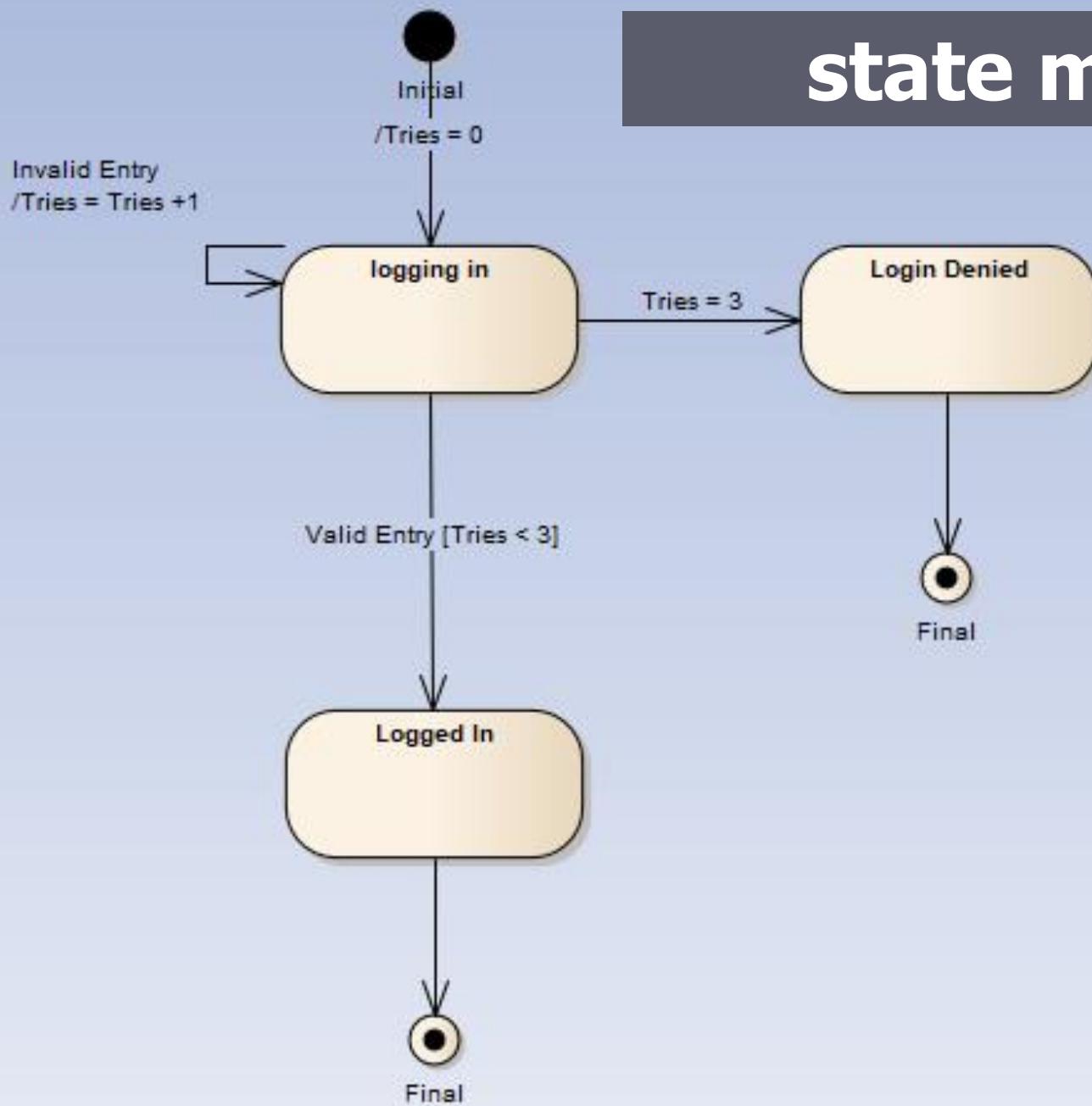
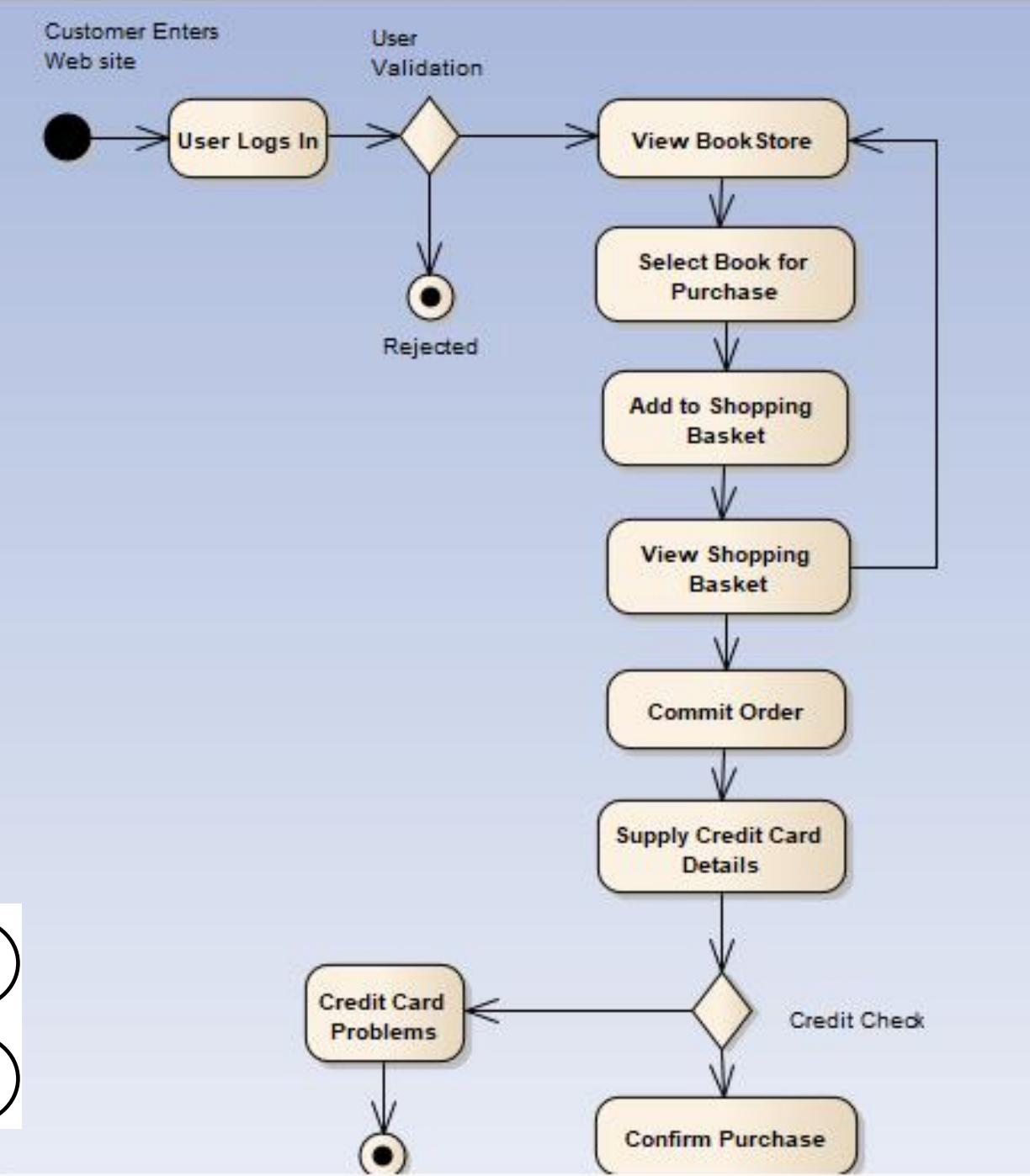
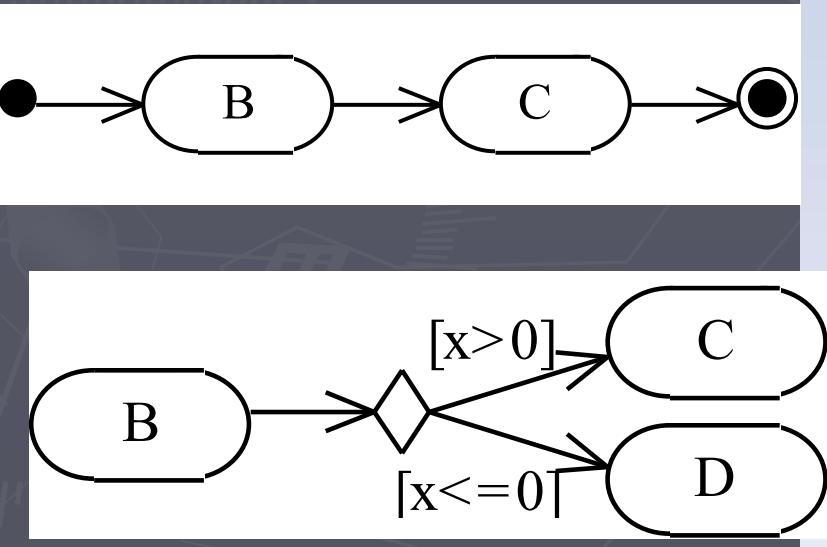


Диаграмма деятельности (activity diagram) – способ описания поведения на основе указания потоков управления и потоков данных

state machine diagram

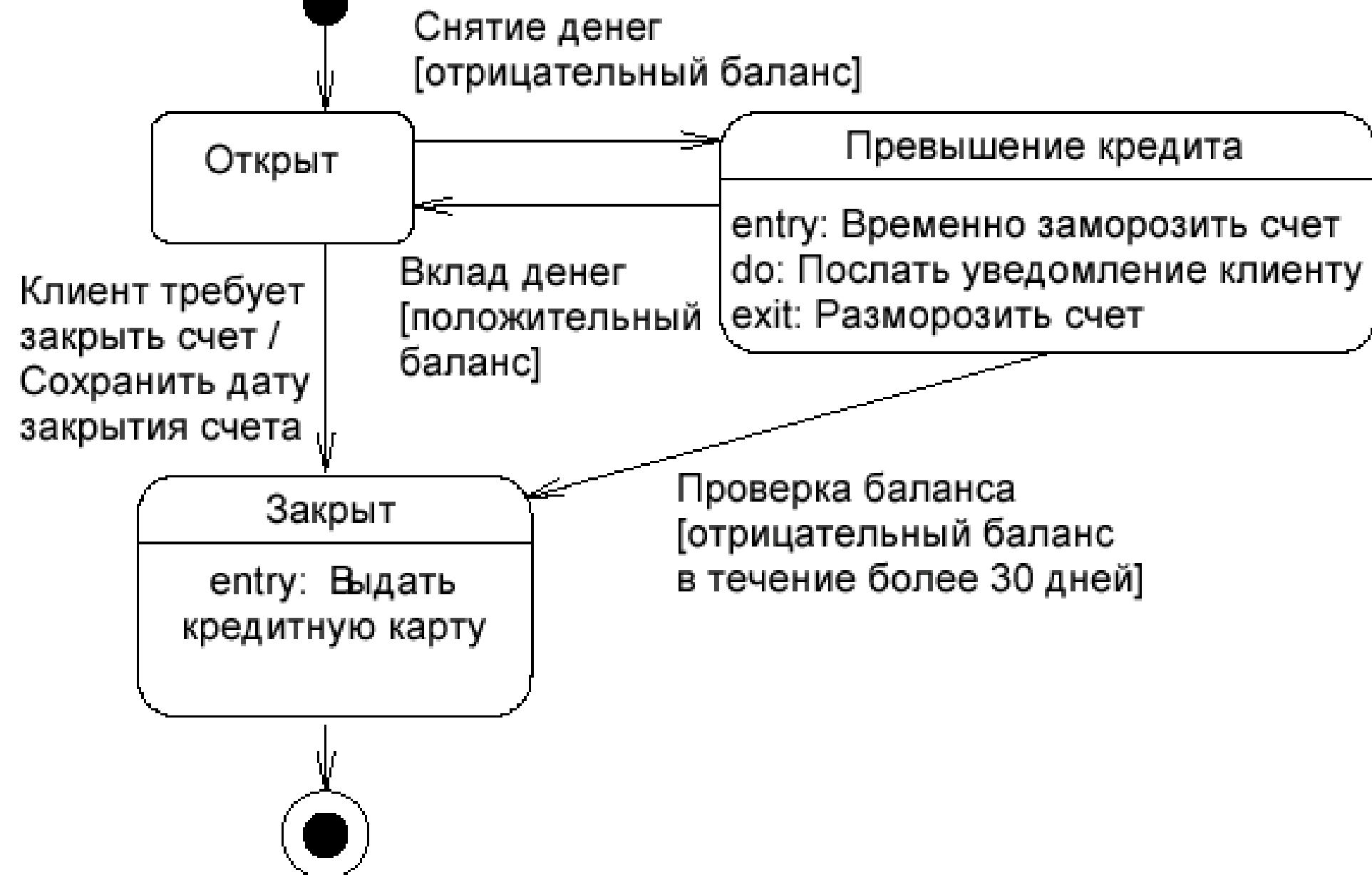


activity diagram



Пример

событие /
действ



Проектирование развертывания

► Компонентные диаграммы

Диаграмма компонентов (component diagram) – показывает взаимосвязи между модулями (логическими или физическими), из которых состоит моделируемая система.

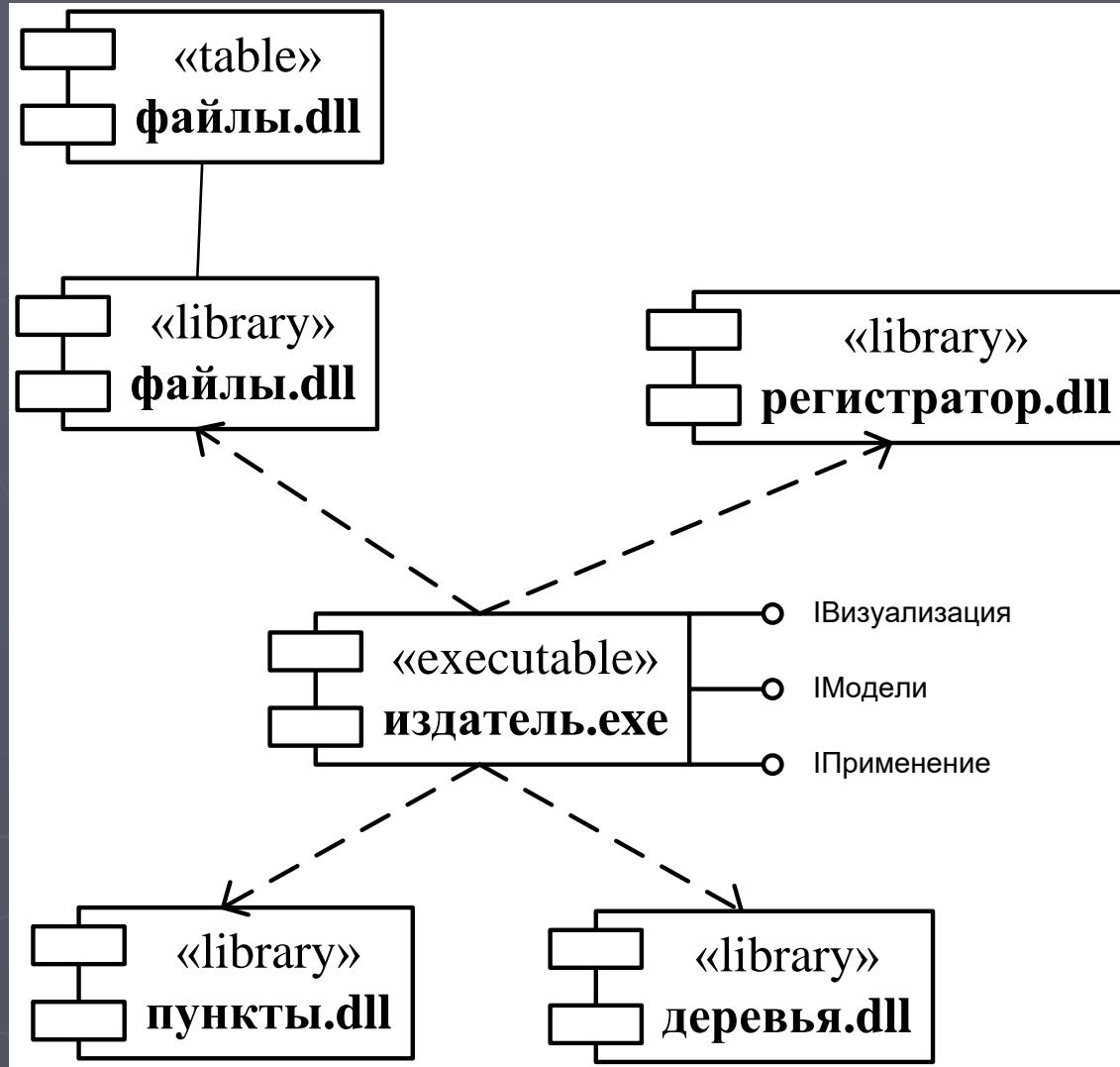


Диаграмма размещения (deployment diagram)

► модель на физическом уровне

исполняемые компоненты

библиотеки кода

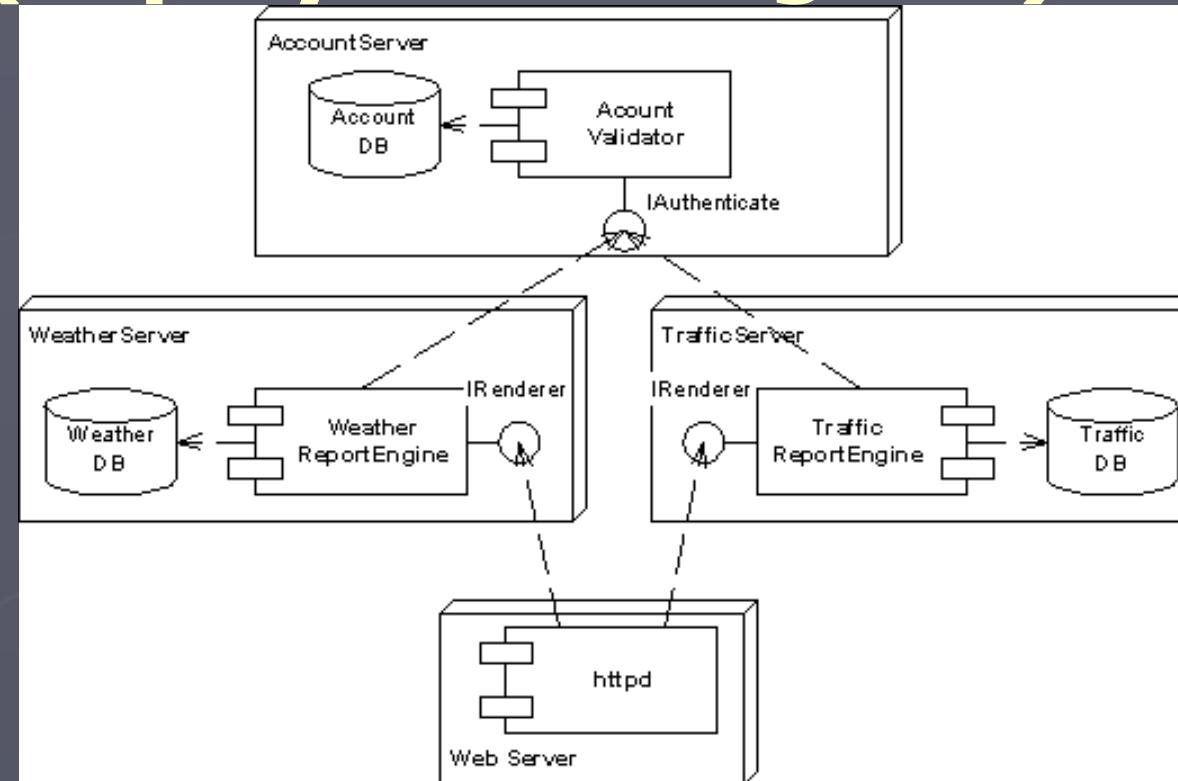
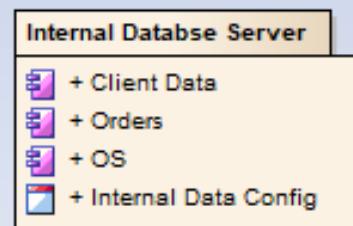
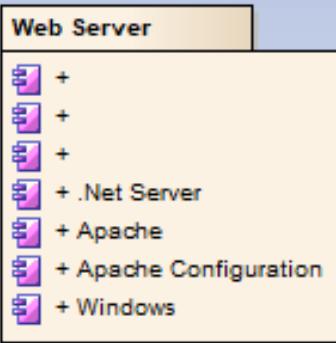
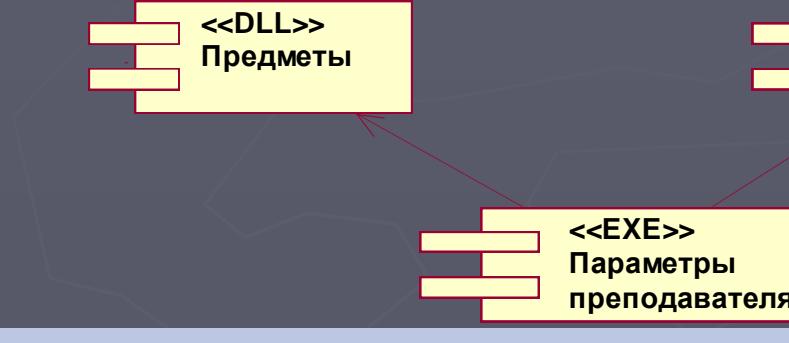
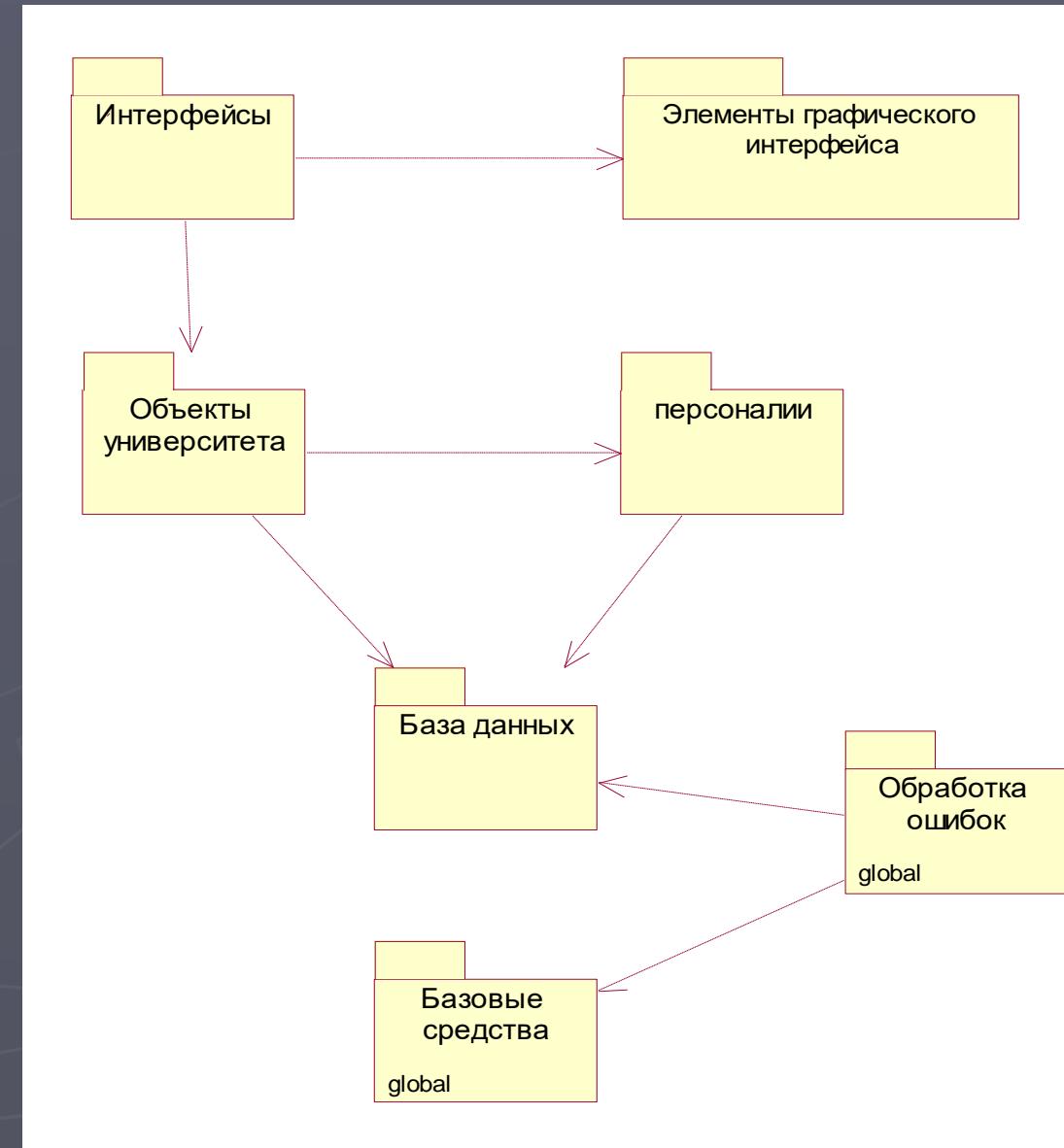
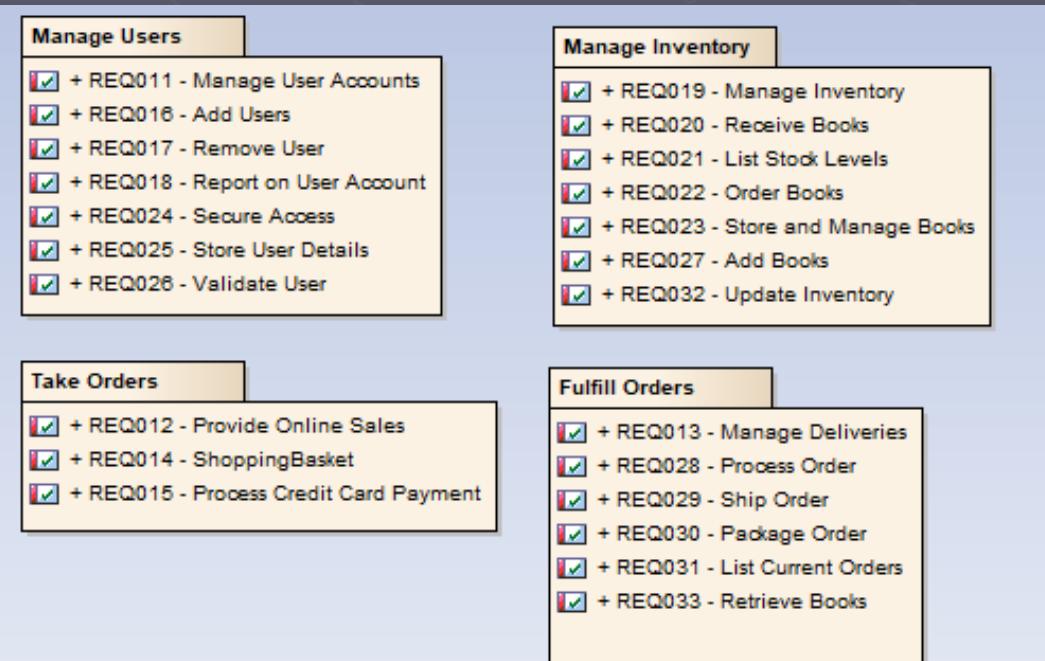


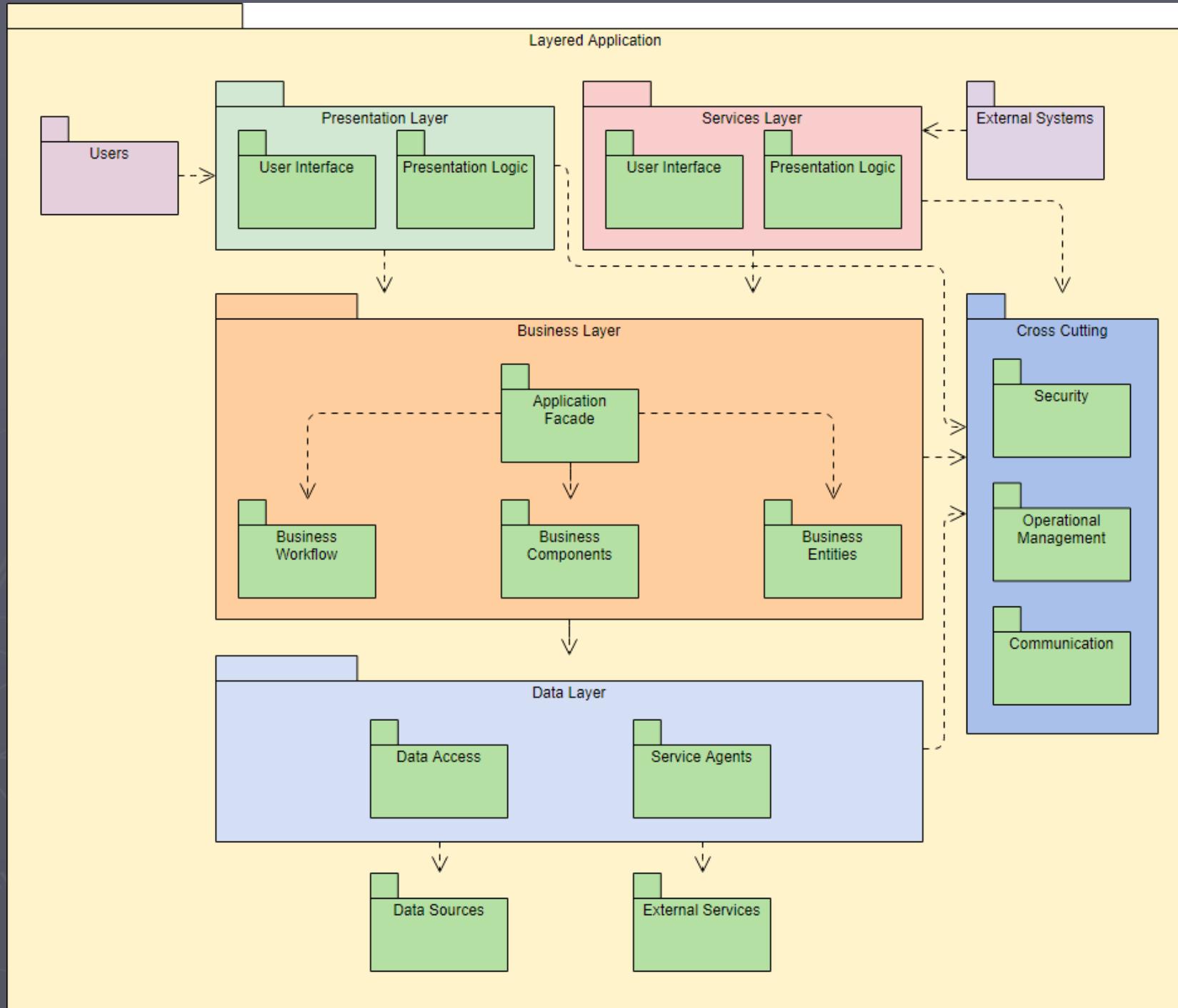
Диаграмма размещения (deployment diagram) наряду с отображением состава и связей элементов системы показывает, как они физически размещены на вычислительных ресурсах во время выполнения.

Диаграмма пакетов (package diagram) – средство группирования элементов модели.

Пакеты применяют , чтобы сгруппировать классы , обладающие некоторой общностью

- ▶ группировать их по стереотипу
- ▶ по функциональности

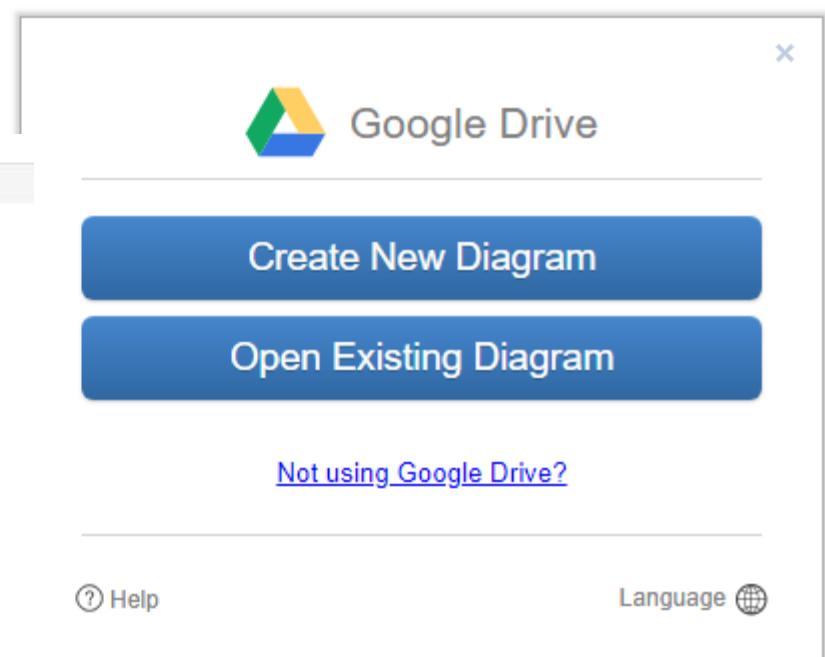
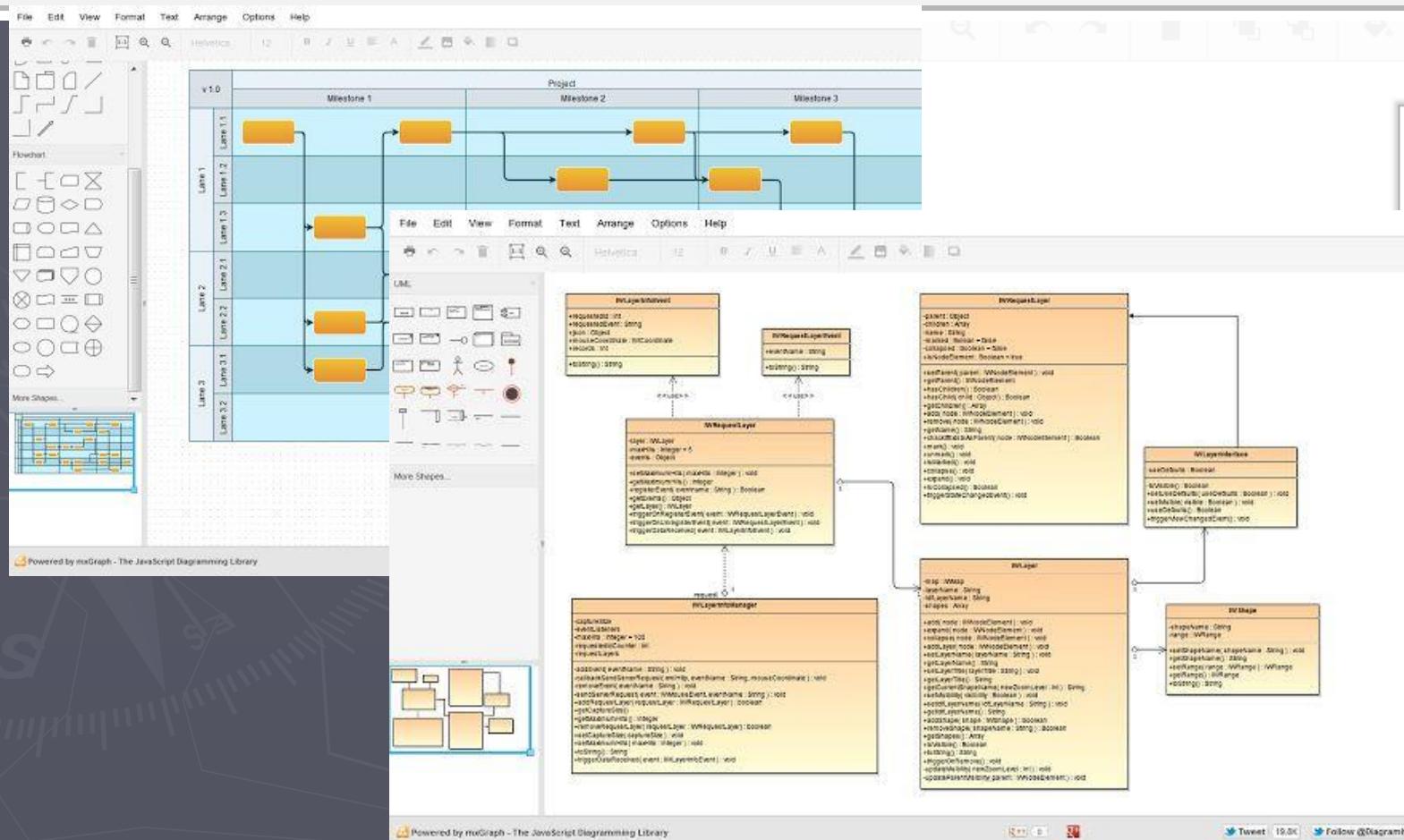




Инструменты

Защищено

<https://www.draw.io>

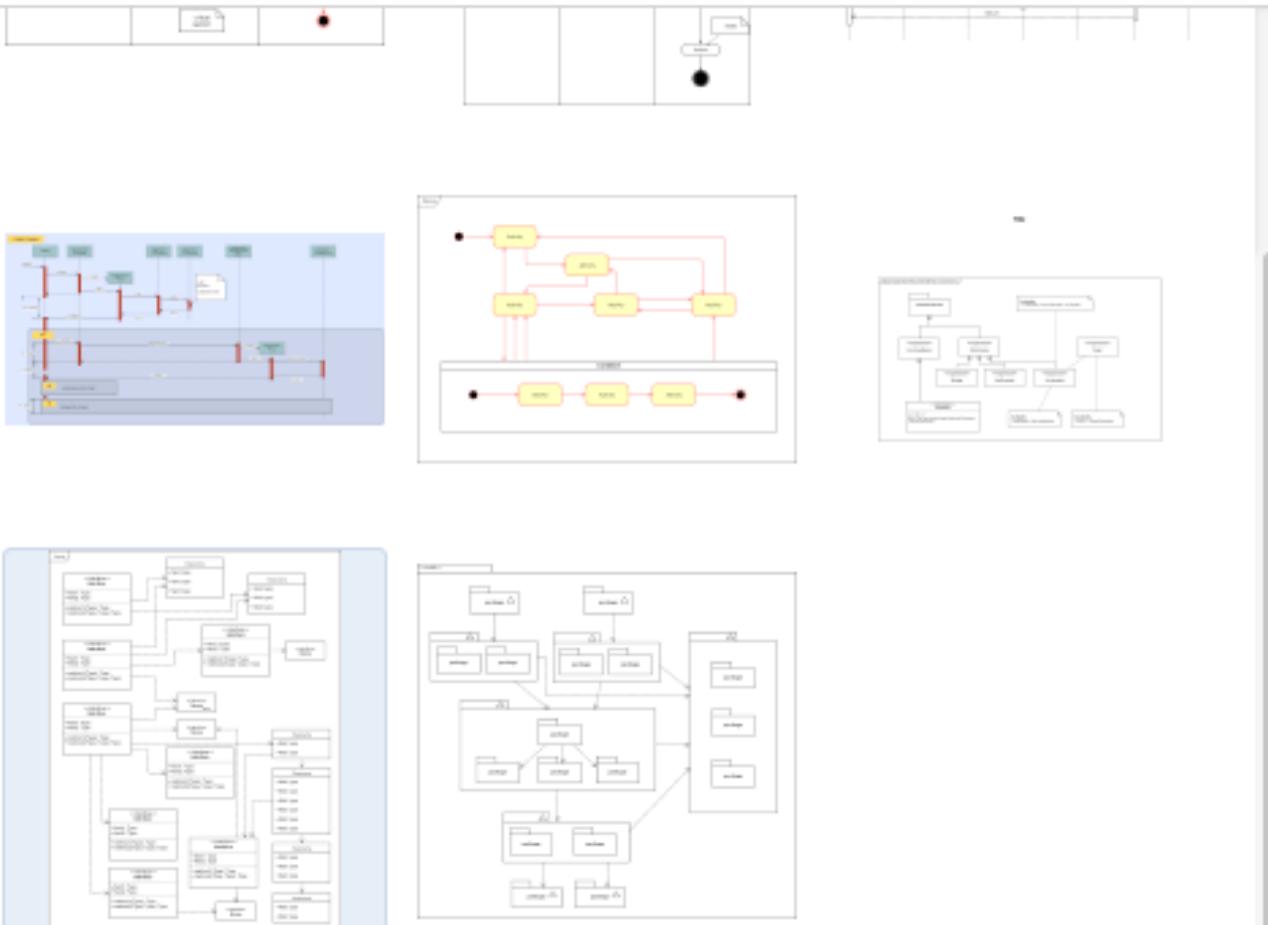


X



Diagram Name: Untitled Diagram.html

- Basic (1)
- Business (11)
- Charts (3)
- Engineering (3)
- Flowcharts (5)
- Layout (3)
- Maps (2)
- Network (18)
- Other (7)
- Software (8)
- Tables (4)
- UML (8)
- Venn (7)



Cancel

Help

From Template URL

Create



Untitled Diagram.html

File Edit View Arrange Extras Help Last change 1 minute ago

Share...

Chat...

Open from...

Open Recent

New...

Rename...

Make a Copy...

Move to Folder...

Import from...

Export as...

Embed

Publish

New Library

Open Library from...

Revision History...

Create Revision

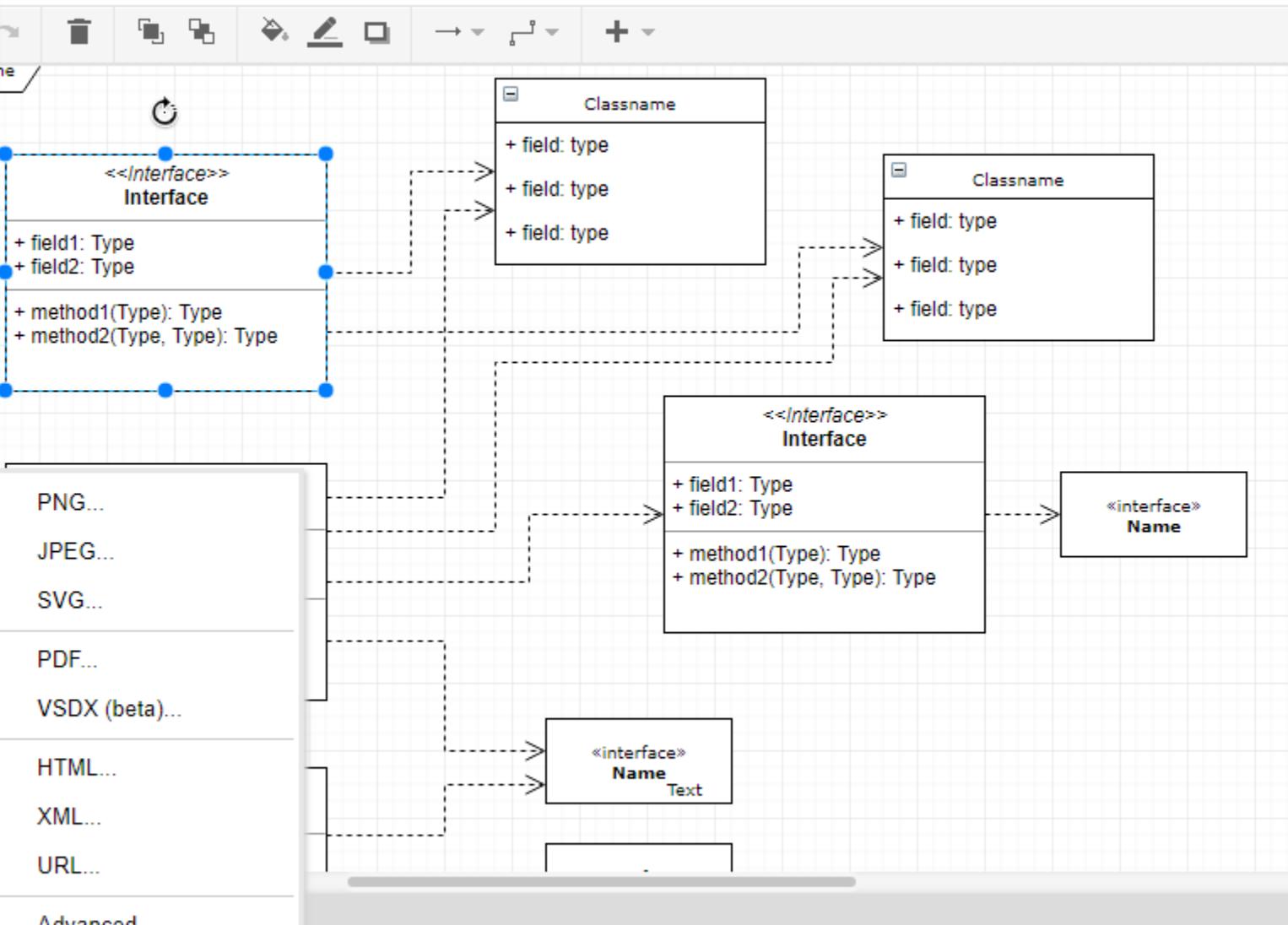
Ctrl+S

Page Setup...

Print...

Ctrl+P

More S...



Style



Fill

Gradient

Line



Perimeter

Opacity

Rounded

Glass

Edit Style

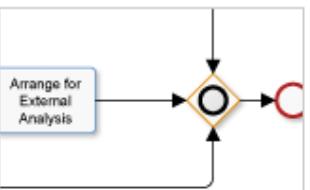
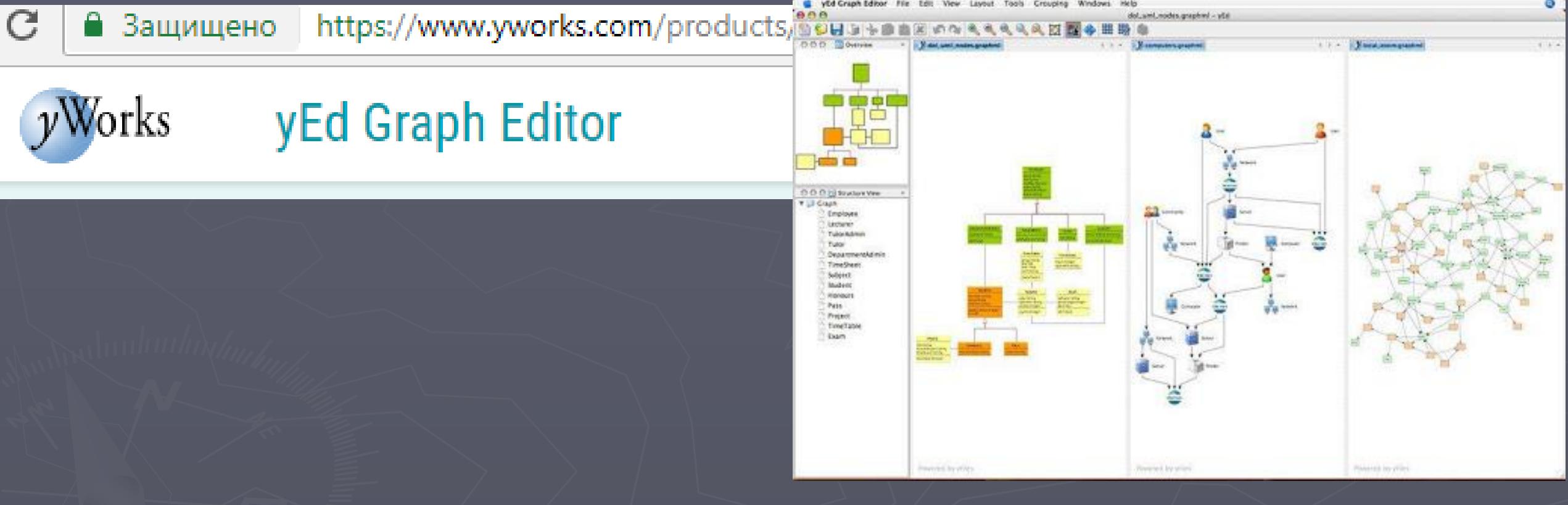
Copy Style

Set a

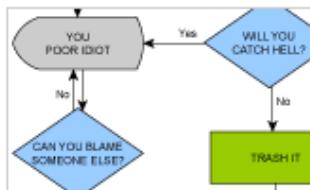
Twitter Enjoying draw.io? Tell the world!

Please Rate Us

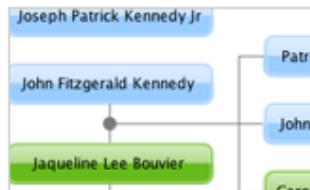
yEd Graph Editor



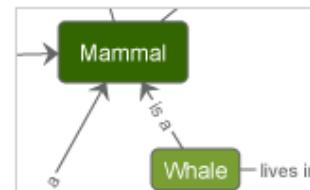
BPMN Diagrams



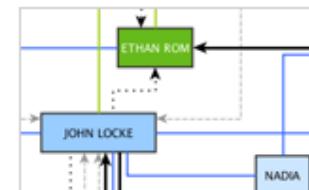
Flowcharts



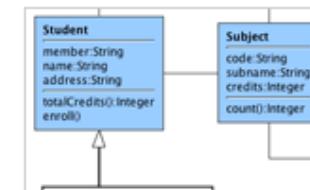
Family Trees



Semantic Networks



Social Networks



UML Class Diagrams



Enterprise Architect

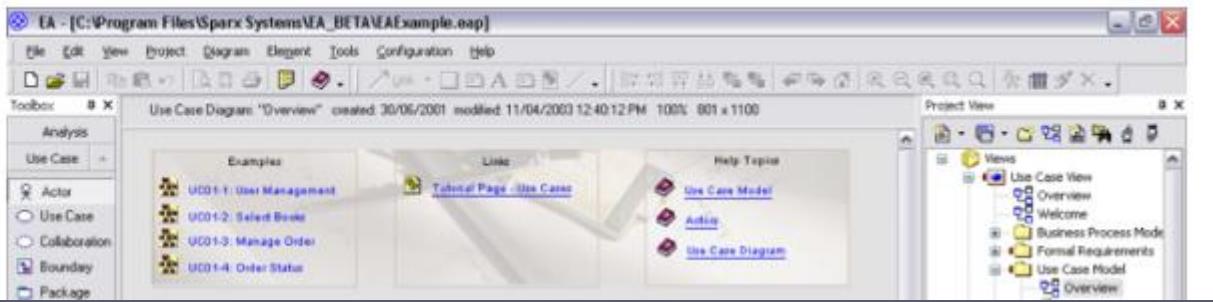
by Sparx Systems

Category: Others

Last Updated: 2018-04-30

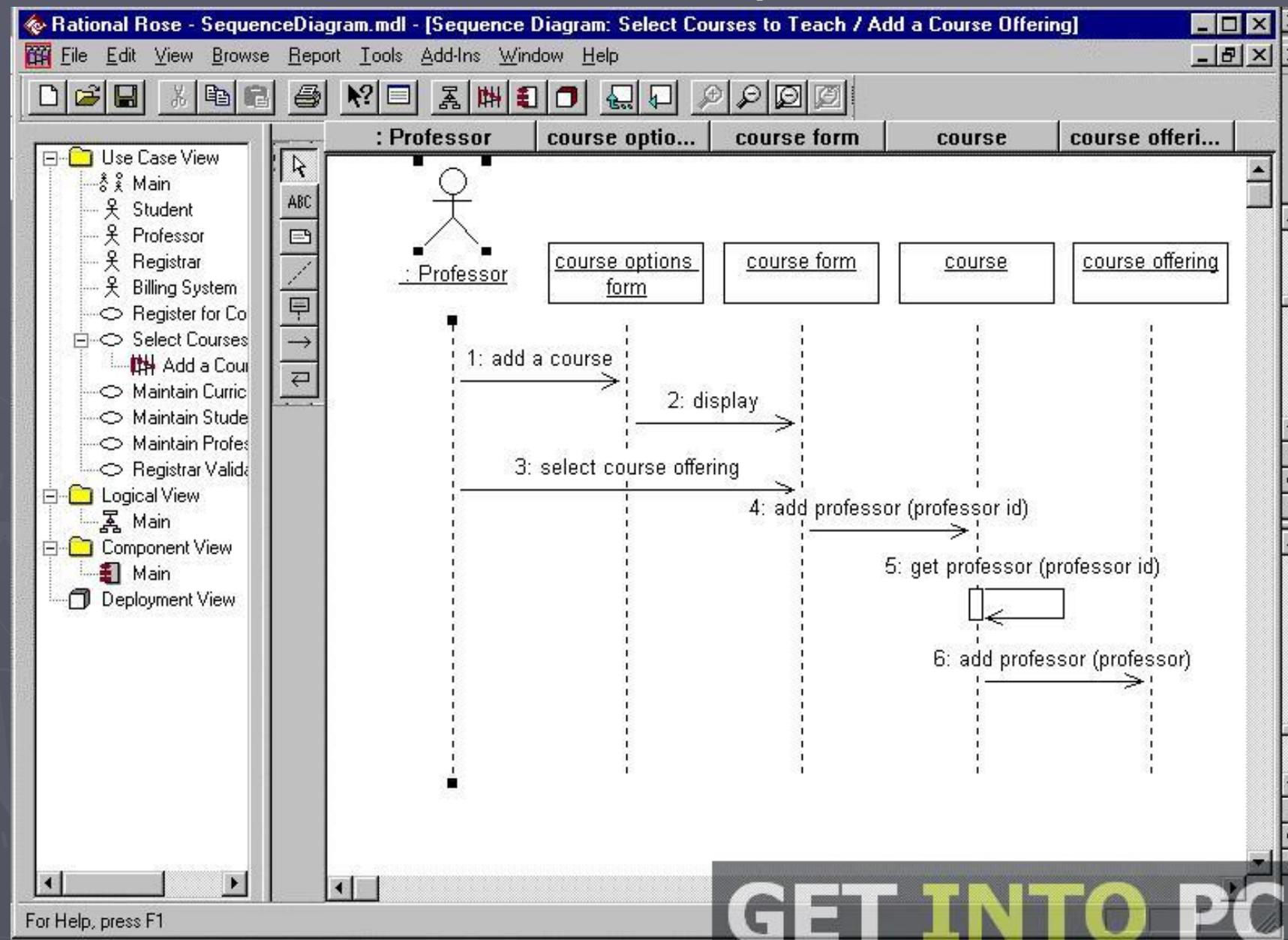
File size: 37.97 MB

Operating system: Windows 7/8/8.1/10



UML 2.5
SysML 1.5
BPMN 2.0
ДМН
BPEL
SoaML
SPeM
WSDL
XSD
ДДС

Rational Rose Enterprise



GET INTO PC

Записка КП

титульный лист установленного образца

техническое задание содержание ;

введение;

главы основной части;

Обзор аналогов и прототипов

Проектирование (диаграммы, бд)

Разработка (классы, методы, блок схемы)

Руководство пользователя

заключение;

список использованных источников ;

Приложения (листинги, скриншоты программы)

Обзор литературы:

- обзор и анализ существующих аналогов и прототипов (2-4стр)
(не копировать из интернета описание и обзор технологий)

Проектирование:

- построение модели данных (логич или физическая структура дб) → выбор структур информации в базе данных, проектирование базы данных
- Диаграммы (диаграммы вариантов, классов и диаграмма последовательности и др.) → описание архитектуры программного средства

Разработка - Программная часть:

- ▶ Функциональное описание ПО (классы, методы...);
- ▶ Описание интерфейсов программы....
- ▶ Тестирование

Руководство пользователя

- ▶ Настройка и установка ПО
- ▶ Инструкция по работе с ПО по ролям
- ▶ *экспериментальный раздел;*