



Нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики

Визуальное моделирование при анализе и проектировании. Основы Unified Modeling Language (UML)

Мееров И.Б., Сысоев А.В., Лебедев И.Г.

Системное моделирование

- Требования позволяют точно описать, что необходимо сделать.
- Если требований много, то понять все требования в совокупности сложно.
- Для упрощения понимания и общения используется моделирование (в частности, визуальное).
- Для моделирования могут применяться, например:
 - Диаграмма прецедентов (сценариев) использования (Use-case diagram).
 - Диаграмма деятельности (Activity diagram).
 - Диаграмма последовательности (Sequence diagram).



- **Модель** строят для того, чтобы лучше понять исследуемую систему.
- **Задачи моделирования:**
 - Визуализация системы в ее некотором состоянии.
 - Определение структуры и поведения системы.
 - Получение шаблона для создания системы.
 - Документирование принятых решений.

- **Объектный подход** – один из ключевых подходов к моделированию.

В результате ООА (Объектно-ориентированный анализ) & ООД (Объектно-ориентированный дизайн) мы получаем «хороший» проект программной системы, прозрачный, удовлетворяющий требованиям, удобный для тестирования и отладки, коллективной разработки, развиваемый, допускающий повторное использование компонентов.




Идея визуального моделирования

- **Визуализация** упрощает понимание проекта в целом.
- **Визуализация** помогает согласовать терминологию и убедиться, что все одинаково понимают термины.
- **Визуализация** делает обсуждение конструктивным и понятным.



UML – это язык

	Формальный	Неформальный
Искусственный	Паскаль	Эсперанто
Естественный	Математика	Русский



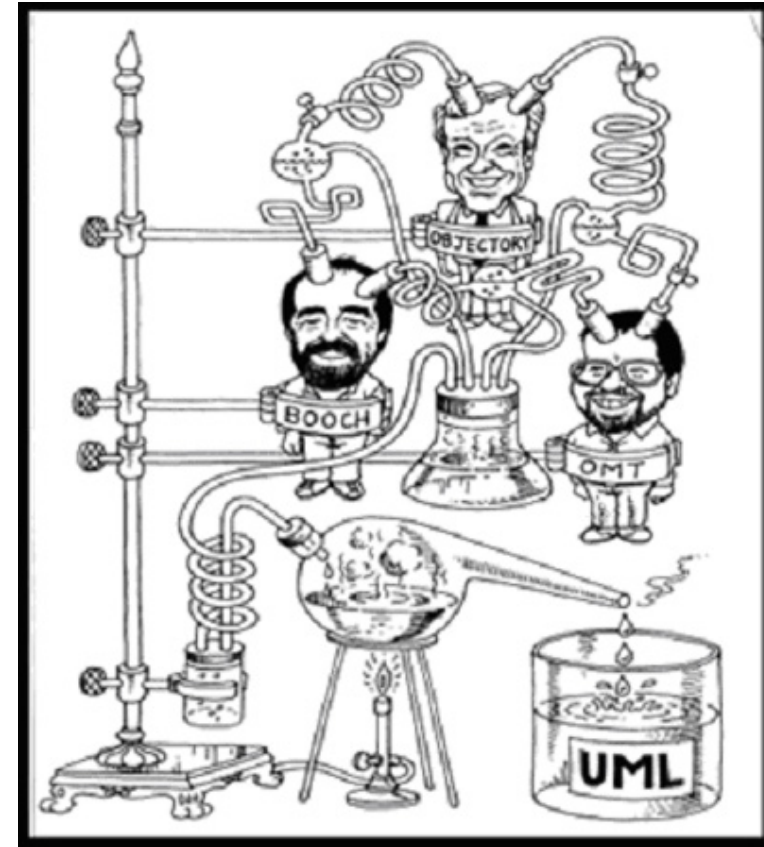
UML как воплощение идеи визуального моделирования

- Для визуального моделирования нужна специальная нотация или **язык**.
- **UML** (unified modeling language) – это **язык** для
 - визуализации,
 - специфицирования,
 - конструирования,
 - документированияэлементов программных систем [3].
- **UML** – язык общего назначения, предназначенный для объектного моделирования.



Unified Modeling Language

- UML – Unified Modeling Language, унифицированный язык для визуального моделирования, результат объединения многих идей и принципов (Booch, Rumbaugh, Jacobson).
- Применяется для проектирования и документирования программных систем.



Three Amigos, How it all began

Источник:

<http://www.ibm.com/developerworks/rational/library/998.html>



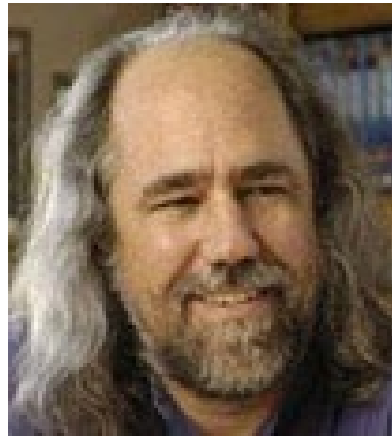
История UML. Этапы большого пути...*

- **1994:** Grady Booch & James Rumbaugh (Rational Software) объединили методы **Booch** (проектирование) и **OMT** (анализ) -> **Unified method**
- **1995:** присоединился Ivar Jacobson (**OOSE** метод)

“Three amigos”



James Rumbaugh



Grady Booch



Ivar Jacobson

История UML. Этапы большого пути...

- **1996** – Идея о **Unified Modeling Language** (three amigos)
- **1996** – **UML Partners** консорциум под руководством three amigos
- Июнь, Октябрь 1996 – UML 0.9 & UML 0.91
- **Январь 1997** – спецификации **UML 1.0** предложены OMG (Object Management Group)
- **Август 1997** – спецификации **UML 1.1** предложены OMG
- **Ноябрь 1997** – **UML 1.2** результат адаптации OMG
- **Июнь 1999** – UML 1.3
- **Сентябрь 2001** – UML 1.4
- **Март 2003** – UML 1.5

История UML. Этапы большого пути*

Принятый стандарт:

- **ISO/IEC 19501:2005** Information technology – Open Distributed Processing – Unified Modeling Language (UML) **Version 1.4.2**
- **Октябрь 2004 – UML 2.0.**
- **Июне 2015 года – UML 2.5**
- **Декабрь 2017 – UML 2.5.1**



UNIFIED MODELING LANGUAGE™



Взято с сайта www.uml.org

• *Источник:* www.wikipedia.org; <http://www-306.ibm.com/software/rational/bios>; <http://www.ivarjacobson.com>

UML позволяет описывать систему следующими **моделями**:

- **Модель функционирования**

Как описывается функциональность системы с точки зрения пользователя.

- **Объектная модель**

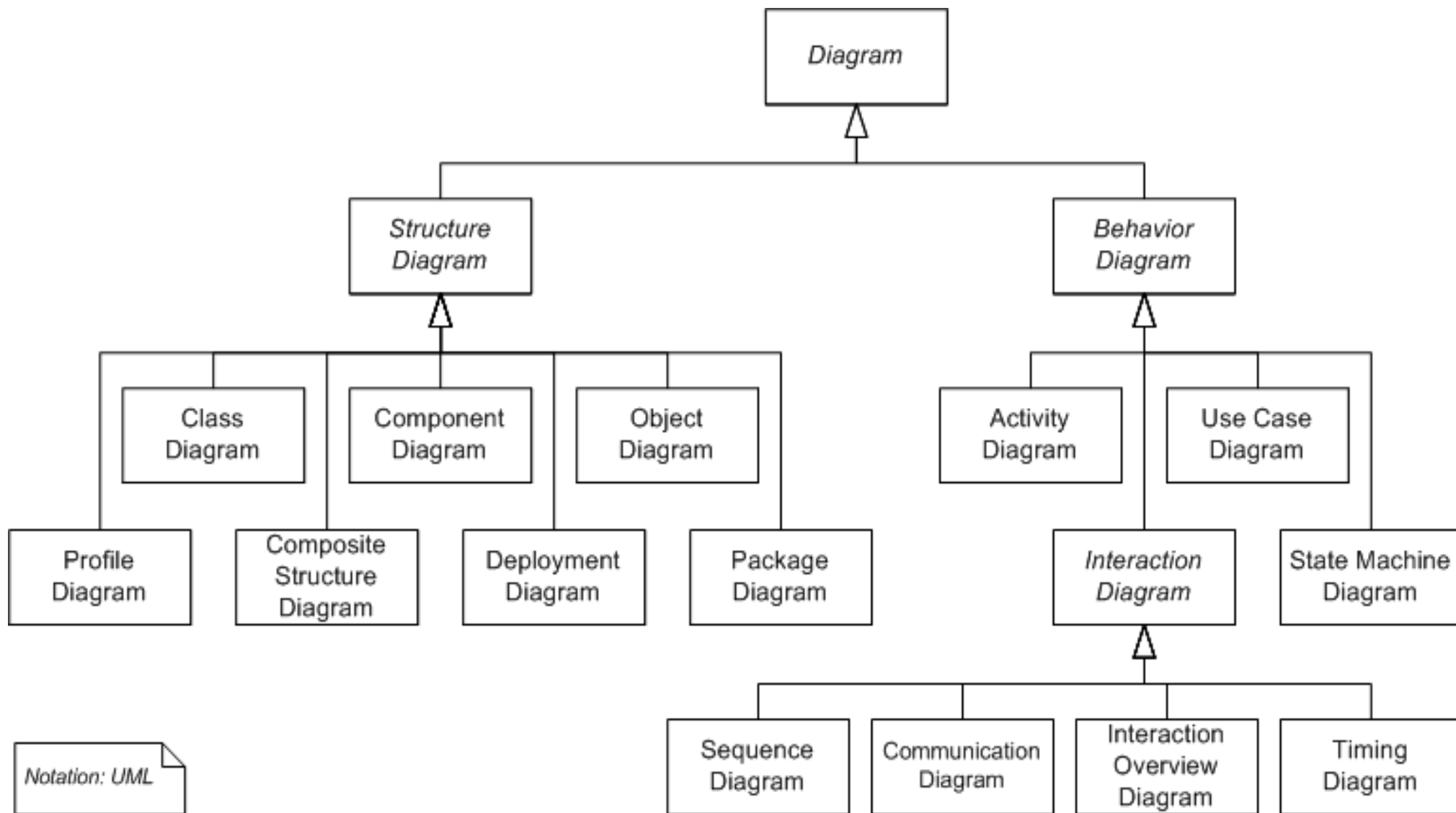
Как выглядит проект системы с точки зрения объектного подхода.

- **Динамическая модель**

Как взаимодействуют друг с другом компоненты системы в динамике, с течением времени. Какие процессы происходят в системе.



Диаграммы UML



Структурные диаграммы (Structure Diagrams)

- **Диаграмма классов (Class diagram)**

Показывает классы, их атрибуты и связи между классами.

- **Диаграмма компонентов (Component diagram)**

Показывает компоненты и связи между ними

- **Структурная диаграмма (Composite structure diagram)**

Показывает внутреннюю структуру классов и связи с внешним миром

- **Диаграмма развертывания (Deployment diagram)**

Показывает, как ПО размещается на аппаратуре (серверах, рабочих станциях...)

- **Диаграмма объектов (Object diagram)**

Показывает структуру системы в конкретный момент времени, объекты, их атрибуты...

- **Диаграмма пакетов (Package diagram)**

Показывает, как система раскладывается на крупные составные части и связи между этими частями

- **Диаграмма Профилей (Profile diagram)**

описывает механизм расширения, позволяющий приспособить UML к разнообразным предметным областям и сферам деятельности.



Диаграммы поведения (Behavior Diagrams)

- **Диаграмма действия (Activity diagram)**

Показывает потоки информации в системе.

- **Диаграмма состояния (State Machine diagram)**

Представляет собой конечный автомат, показывающий функционирование системы.

- **Диаграмма вариантов использования (Use case diagram)**

Показывает работу системы с точки зрения пользователей.



Диаграммы взаимодействия (Interaction Diagrams)

- **Диаграмма коммуникации (Communication diagram)**
Показывает структурную организацию участвующих во взаимодействии объектов
- **Диаграмма взаимодействия (Interaction overview diagram)**
Создание высокоуровневого представления об общем характере взаимодействий в проектируемой системе
- **Диаграмма последовательности (Sequence diagram)**
Показывает временную упорядоченность событий
- **Временная диаграмма (Timing diagram)**
Диаграмма связана с временными рамками



- **Для описания структуры:**
Актер, Атрибут, Класс, Компонент, Интерфейс, Объект, Пакет.
- **Для описания поведения:**
Действие, Событие, Сообщение, Метод, Операция, Состояние, Вариант использования.
- **Для описания связей:**
Агрегация, Ассоциация, Композиция, Зависимость, Наследование.
- **Некоторые другие понятия:**
Стереотип, Кратность, Роль.



- SRS – Seat reservation system.
- Авиакомпания «GlobalAvia».
- SRS должна содержать 2 части:
 - Занесение информации.
 - Работа с клиентами.
- Дополнительная информация:
 - Рейсы спланированы так, что до пункта назначения можно долететь с пересадками.
 - Система должна помогать покупать билеты в зависимости от пожеланий пользователя.

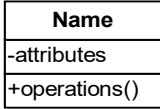



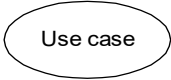
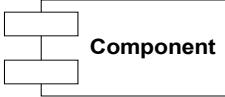
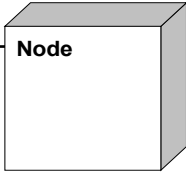


Как функционирует программная система?





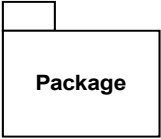
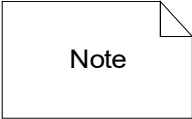
- Программная система **не функционирует сама по себе.**
- Программная система функционирует под воздействием **актеров – пользователей, машин и других программ.**
- **Актер** ожидает, что система ведет себя строго определенным образом.
- **Актер** оказывает **воздействие** — система выдает ожидаемый **результат.**
- Модель того, как воздействие приводит к результату — **Вариант использования.**



Нотация для структурных сущностей

Класс	
Действующее лицо	 Actor
Интерфейс	
Кооперация	
Вариант использования	
Компонент	
Узел	

Нотация для других сущностей

Состояние	
Деятельность	
Развилка / слияние	
Ветвление	
Пакет	
Примечание	

- Зависимость (dependency)
 - Зависимое -----> Независимое
- Ассоциация (association) ————
 - Агрегирование (aggregation) ————◇
 - Композиция (composition) ————◆
- Обобщение (generalization)
 - Потомок ————▷ Предок
- Реализация (implementation)
 - Implementation -----▷ Interface

Диаграмма прецедентов использования

- Способ получения (например, через «мозговой штурм»):
 1. Выделяем как можно больше участвующих сторон – актеры.
 2. Выделяем прецеденты и соединяем их с актерами.
 3. Для каждого из прецедентов определяем, нужна ли связь с другими актерами или прецедентами.
 4. Описываем по абзацу каждого актера и каждый прецедент или выставляем связи с существующими требованиями.
 5. Дополняем глоссарий новыми определениями (если есть).
 6. Проверяем прецеденты на удовлетворение ранее описанным требованиям.
 7. Выделяем наиболее важную функциональность.



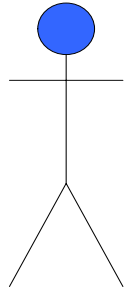
Элементы диаграмм использования

- Сущности
 - Действующие лица
 - Варианты использования
 - Примечания
- Отношения
 - Ассоциации между действующими лицами и вариантами использования
 - Обобщения между действующими лицами
 - Обобщения и зависимости между вариантами использования



Диаграмма прецедентов использования...

- *Диаграмма прецедентов использования* содержит два элемента:
 - **Актер в UML** – человек, машина или программа, воздействует на систему, является внешним по отношению к ней.
 - **Прецедент использования в UML** – описание последовательности действий – (часто прецедент связан с диаграммой деятельности).



- Действующие лица находятся ВНЕ проектируемой системы
- Действующее лицо – это множество логически взаимосвязанных РОЛЕЙ
- Действующее лицо – это стереотипный КЛАСС
- Типовые случаи: категории пользователей, внешние программные и аппаратные средства

Диаграмма прецедентов использования...

- Связи:
 - Актеры и варианты использования общаются посредством посылки сообщений.
 - Сообщения могут идти в обе стороны.
 - Стрелка показывает инициатора общения (актер на рисунке) и может быть опущена.

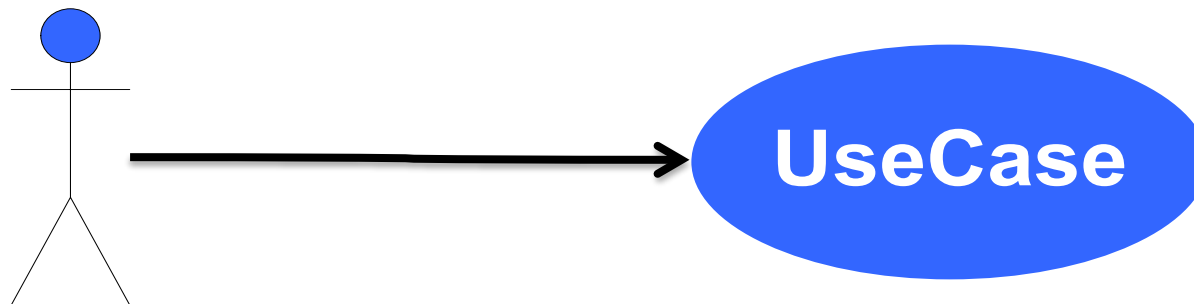


Диаграмма использования

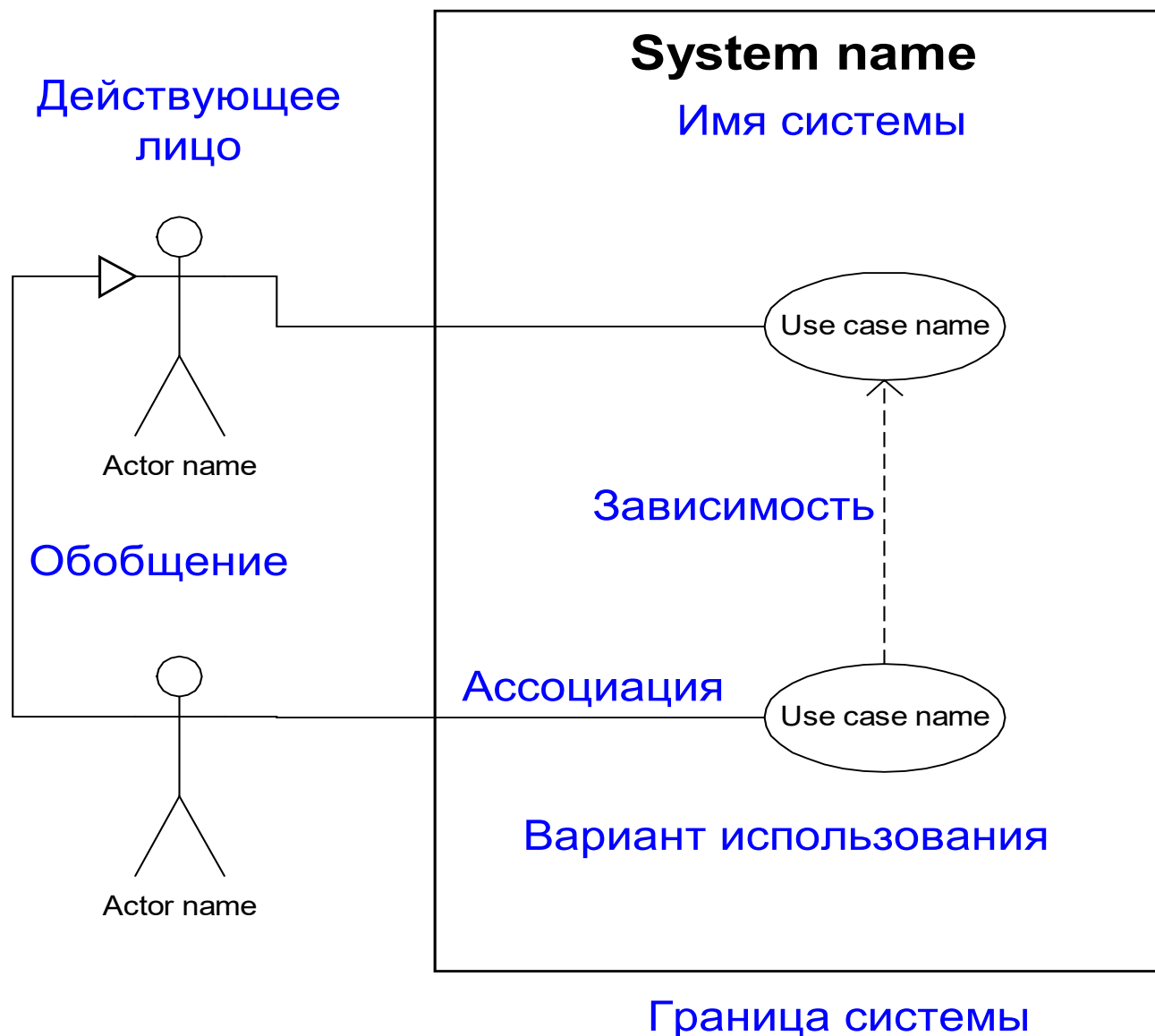


Диаграмма прецедентов использования...

- Диаграмма прецедентов использования не должна содержать прецедентов, не имеющих отношения к разрабатываемой системе.

Пример:

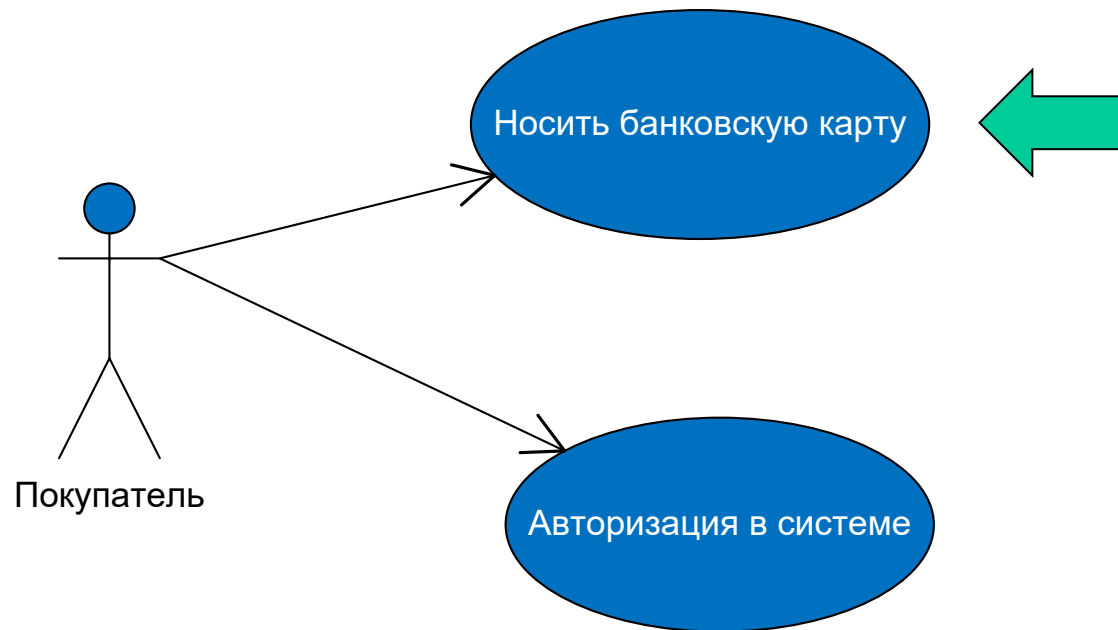


Диаграмма прецедентов использования...

- Диаграмма прецедентов использования не должна содержать «висячих» прецедентов.

Пример:

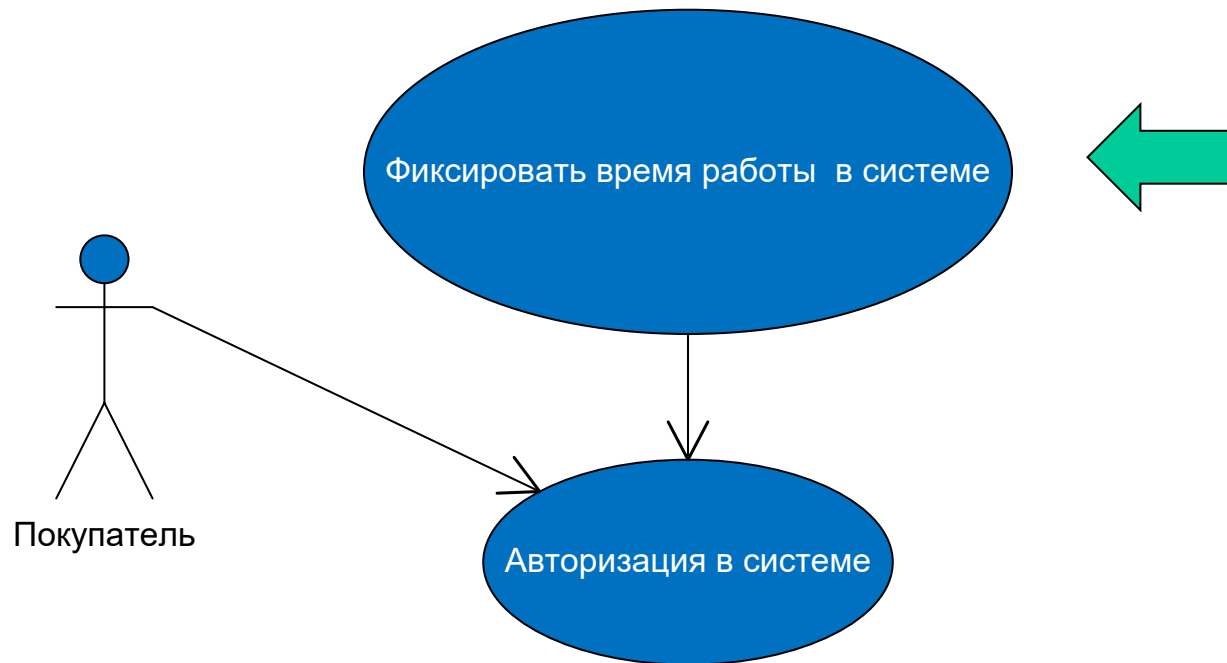


Диаграмма прецедентов использования...

- Диаграмма прецедентов использования не должна отображать последовательность действий.

Пример:

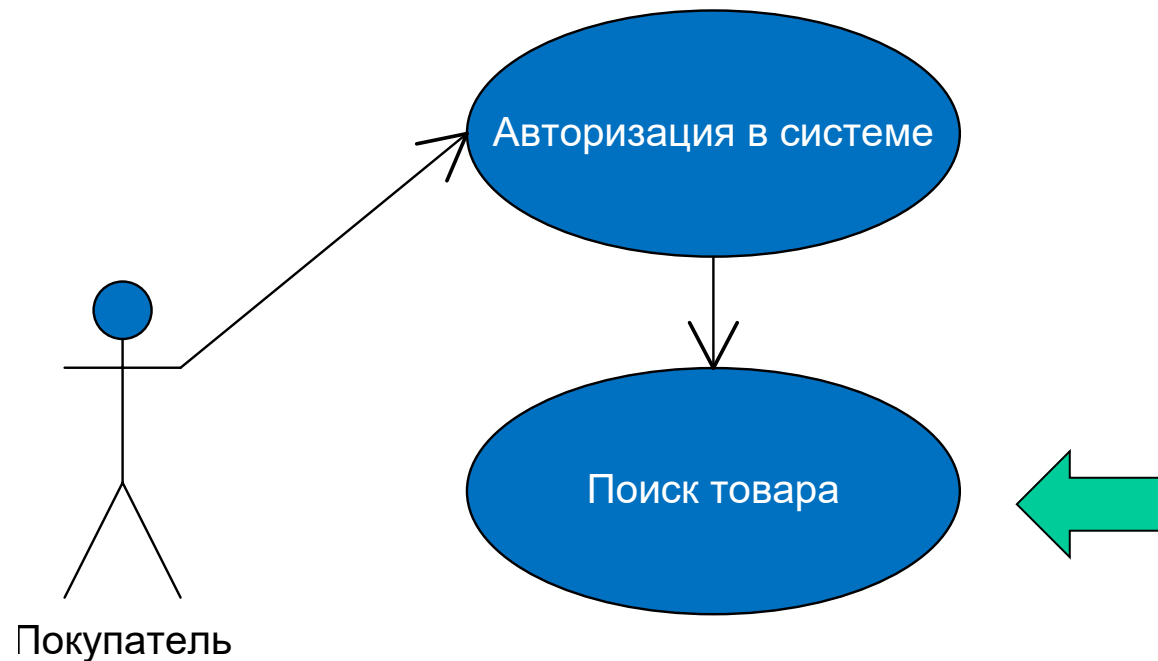
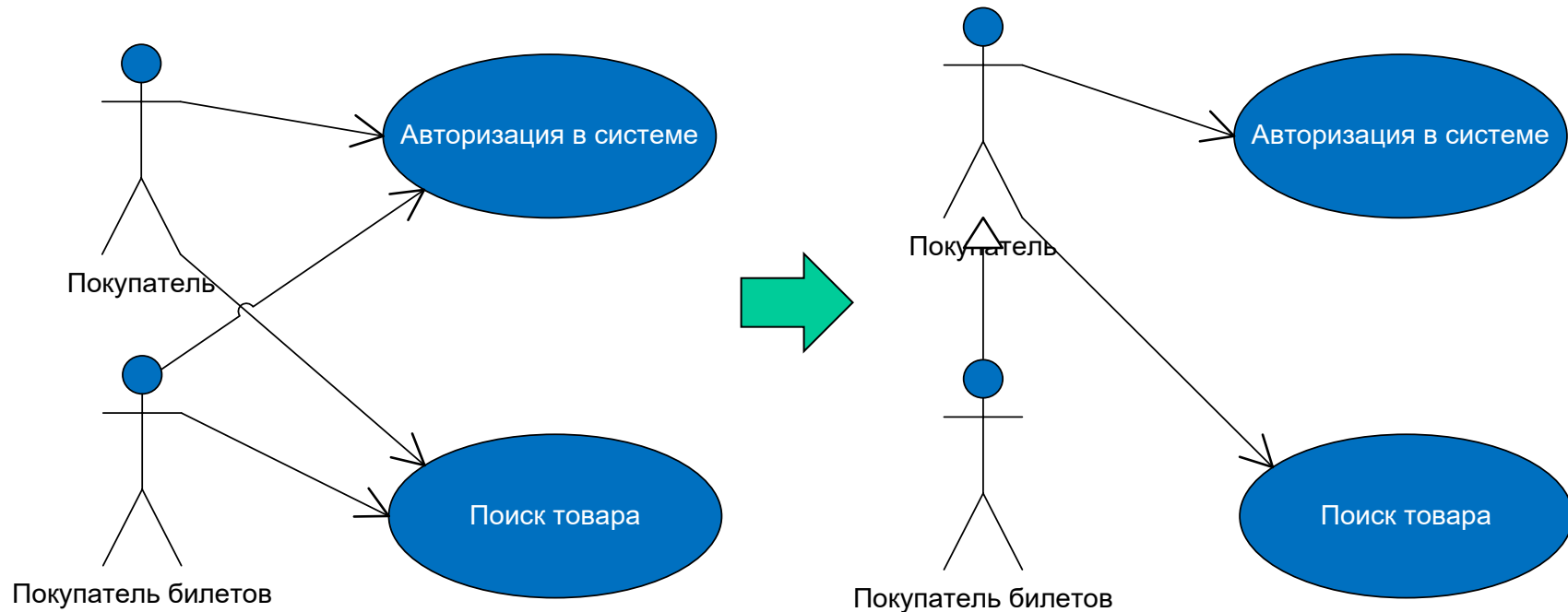


Диаграмма прецедентов использования...

- Диаграмма прецедентов использования не должна содержать актеров, дублирующих роли.

Пример:

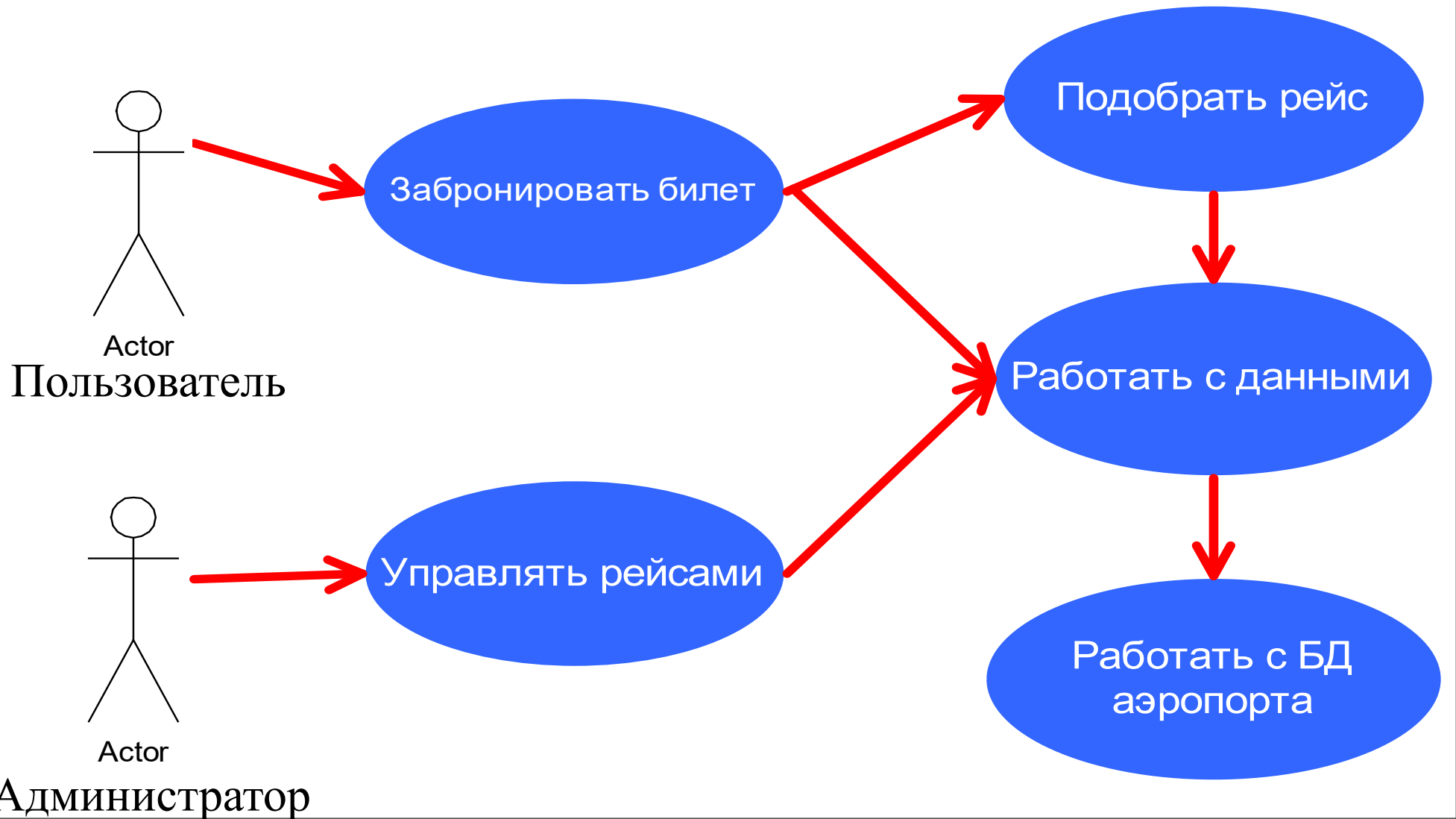


Актеры и Варианты использования в SRS

- **Актеры:**
 - Пользователь.
 - Администратор.
- **Варианты использования:**
 - Забронировать билет.
 - Подобрать рейс.
 - Работать с данными.
 - Управлять рейсами.
 - Работать с БД аэропорта.

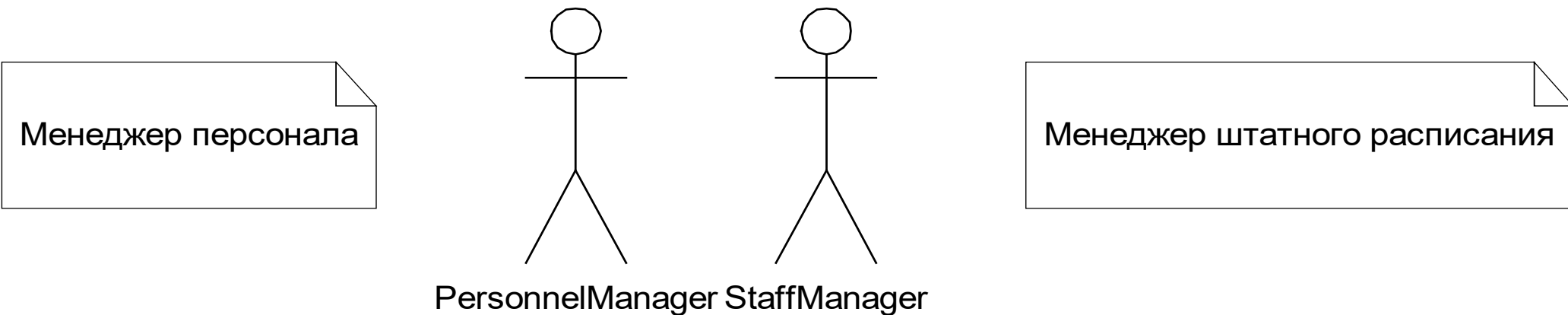


Диаграмма вариантов использования (Use case diagram)



Пример: действующие лица ИС ОК

- Менеджер персонала
 - Работает с конкретными людьми
- Менеджер штатного расписания
 - Работает с абстрактными должностями и подразделениями



Варианты использования и их идентификация

- Вариант использования – множество возможных последовательностей событий/действий (сценариев), приводящих к значимому для действующего лица результату
- Типичные случаи: пункты ТЗ
- Если ТЗ смутное, его можно попробовать переписать фразами субъект – предикат – объект

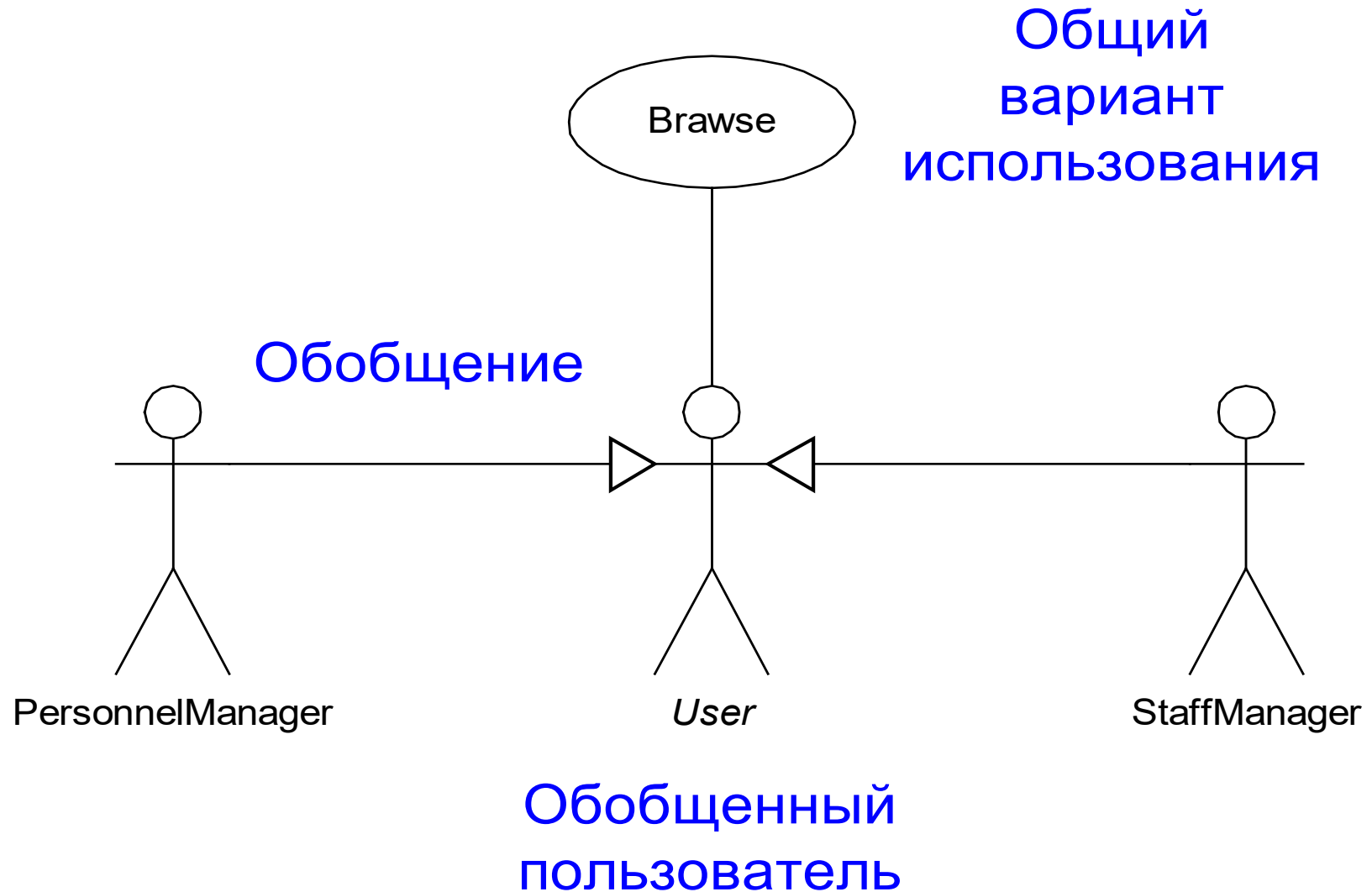


Пример: варианты использования ИС ОК

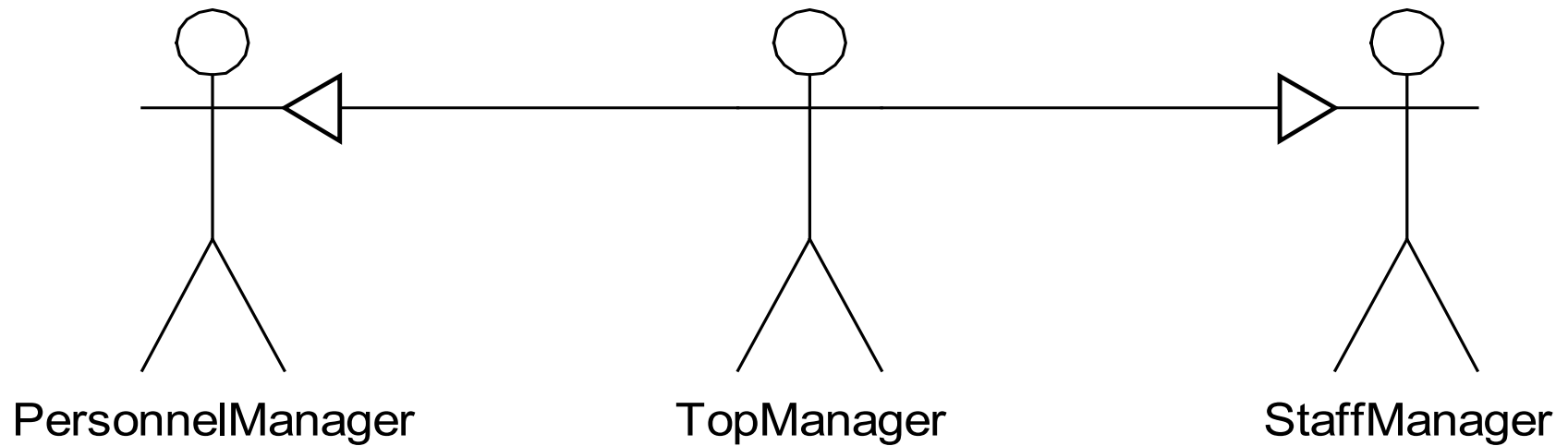
- **Менеджер персонала выполняет действия**
 - Прием сотрудника
 - Перевод сотрудника
 - Увольнение сотрудника
- **Менеджер штатного расписания выполняет действия**
 - Создание подразделения
 - Ликвидация подразделения
 - Создание вакансии
 - Сокращение должности

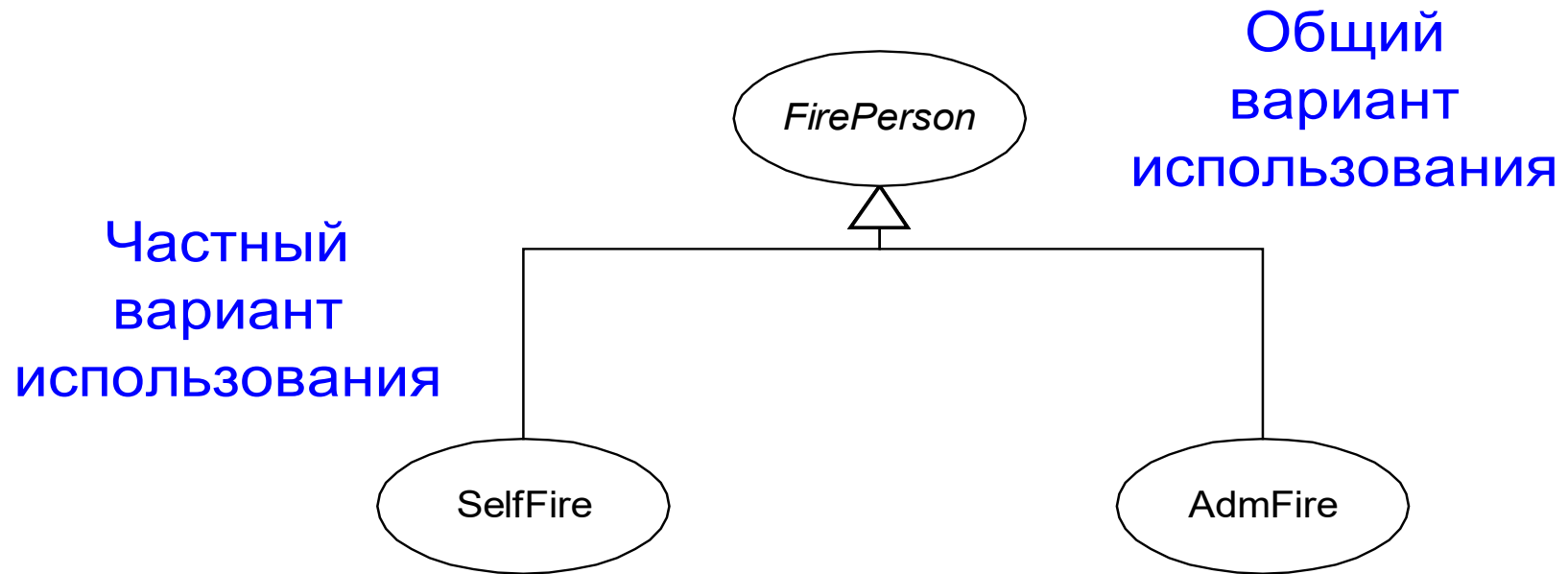


Обобщение вариантов использования (1)

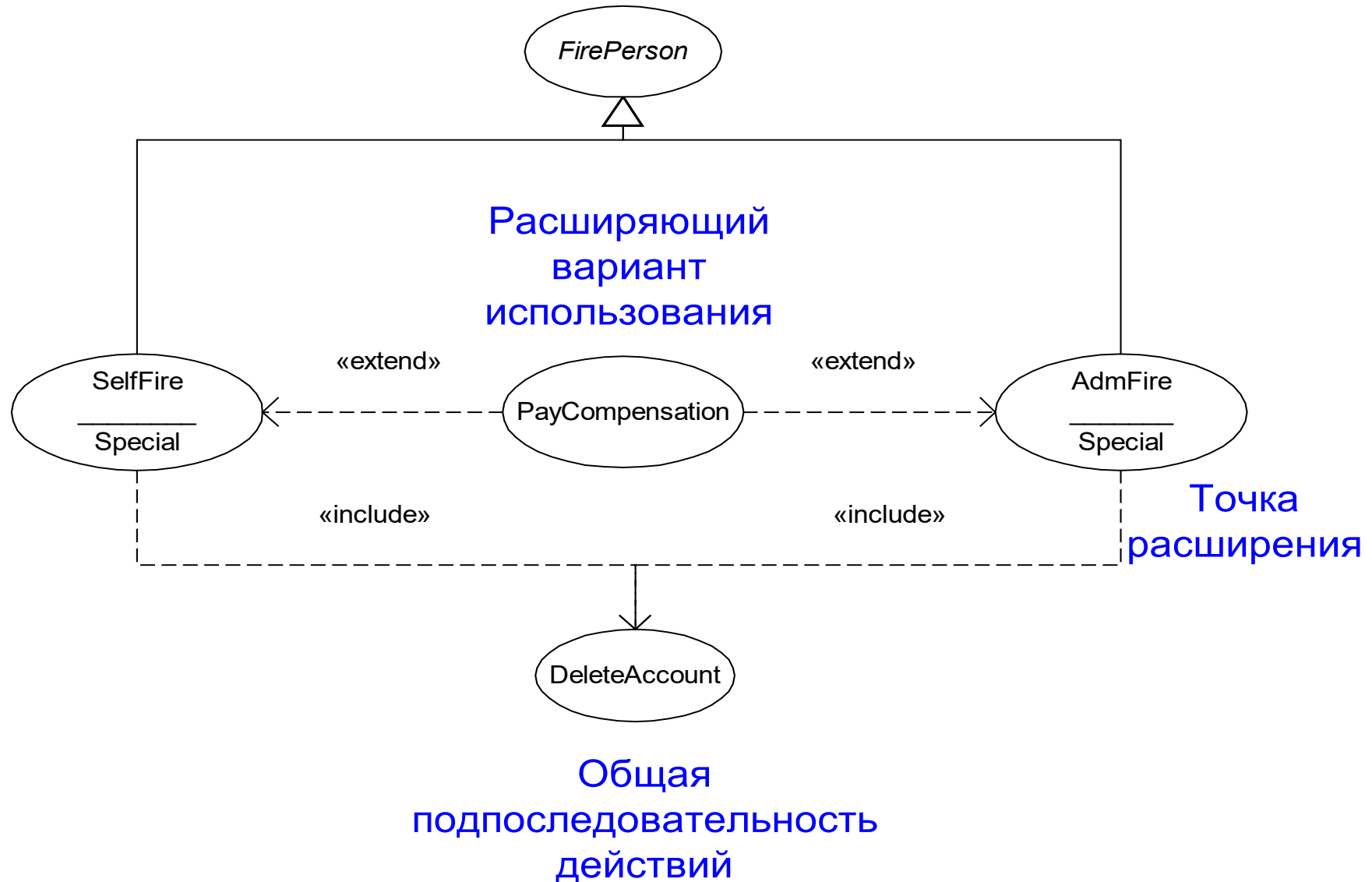


Обобщение вариантов использования





Зависимости между вариантами использования



- Для описания сценариев Варианта использования используется **Диаграмма деятельности** (она же Диаграммы действия и Диаграмма активностей).
- Диаграмма деятельности это блок-схема, которая отображает динамику в поведении системы.
- Может использоваться **не только** для описания сценариев Варианта использования.

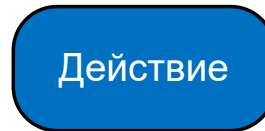
Диаграмма деятельности ...

- Диаграмма деятельности отображает последовательность действий.
- Основные элементы:

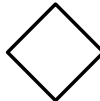
- Начало действий



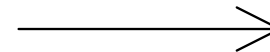
- Действие



- Условие



- Передача потока управления



- Конец действий



Диаграмма деятельности ...

- Диаграмма прецедентов использования:

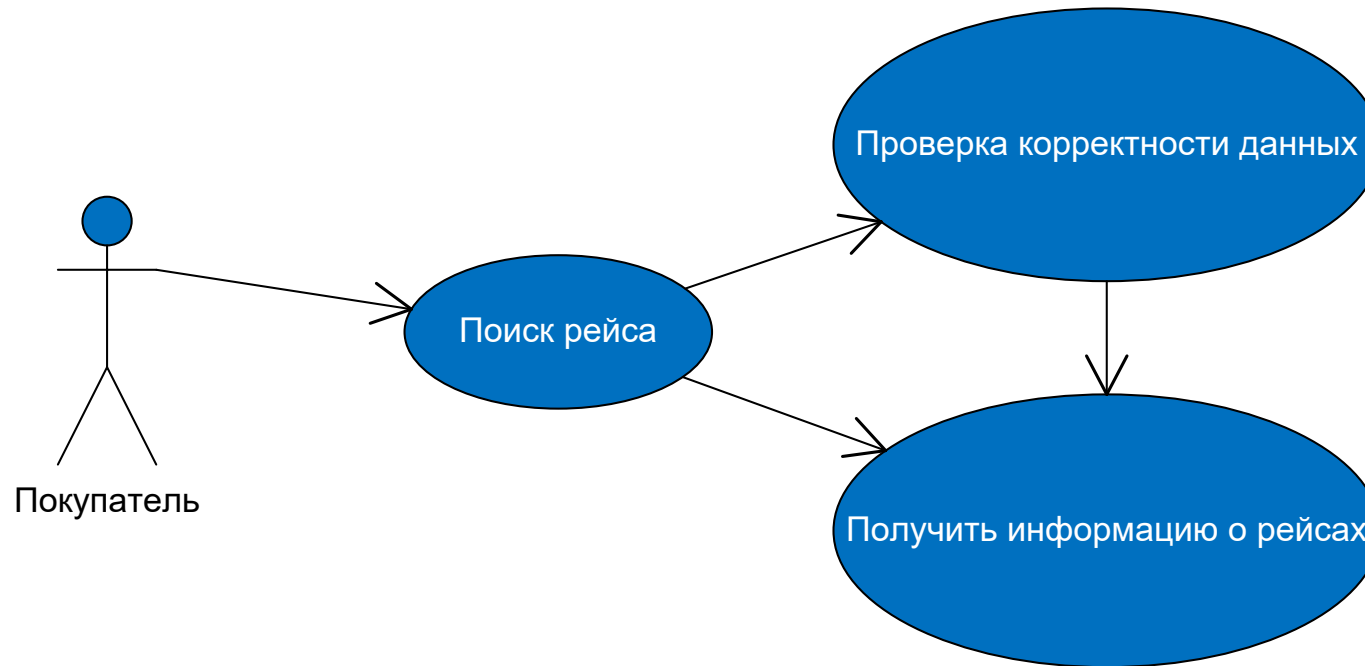
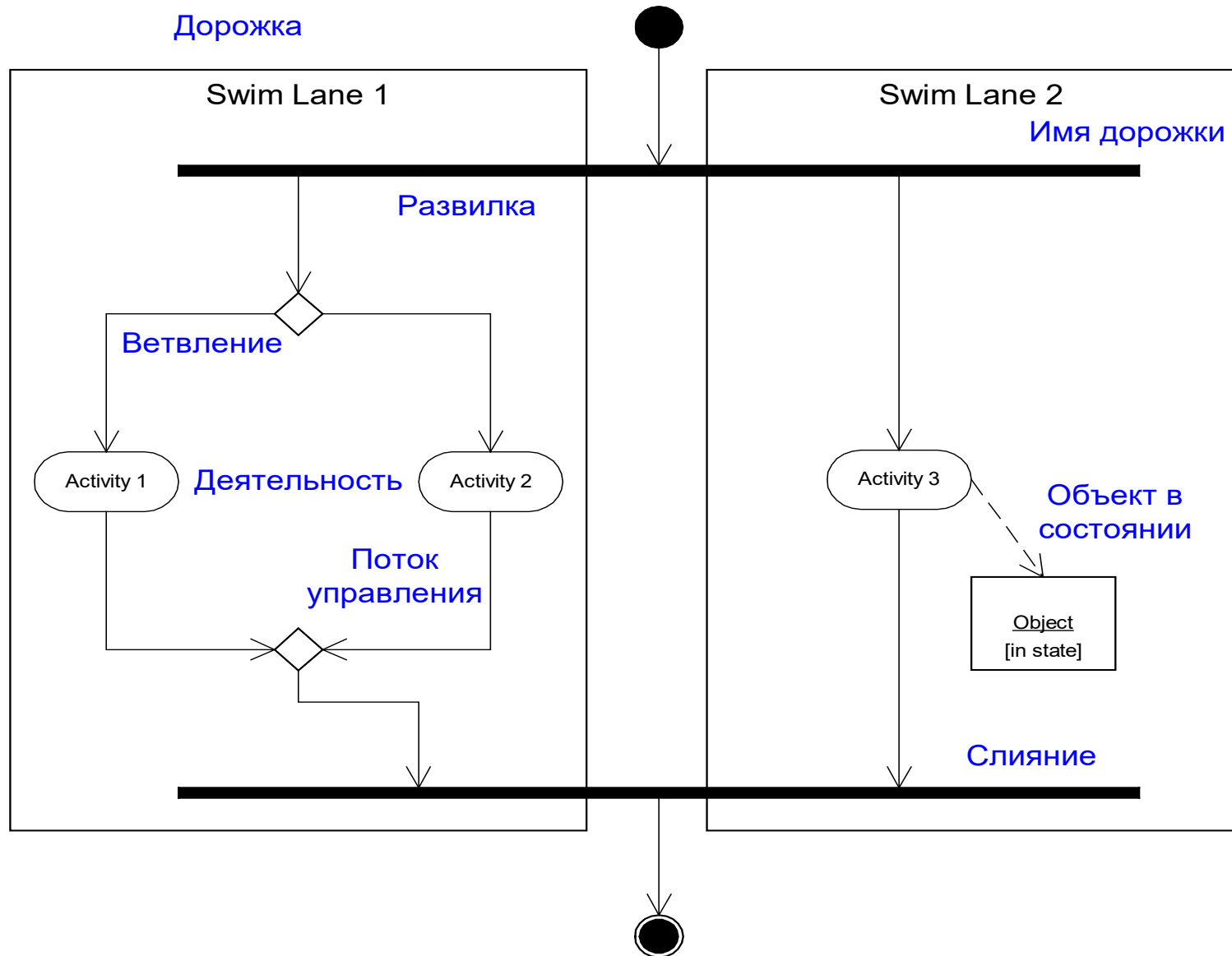


Диаграмма деятельности

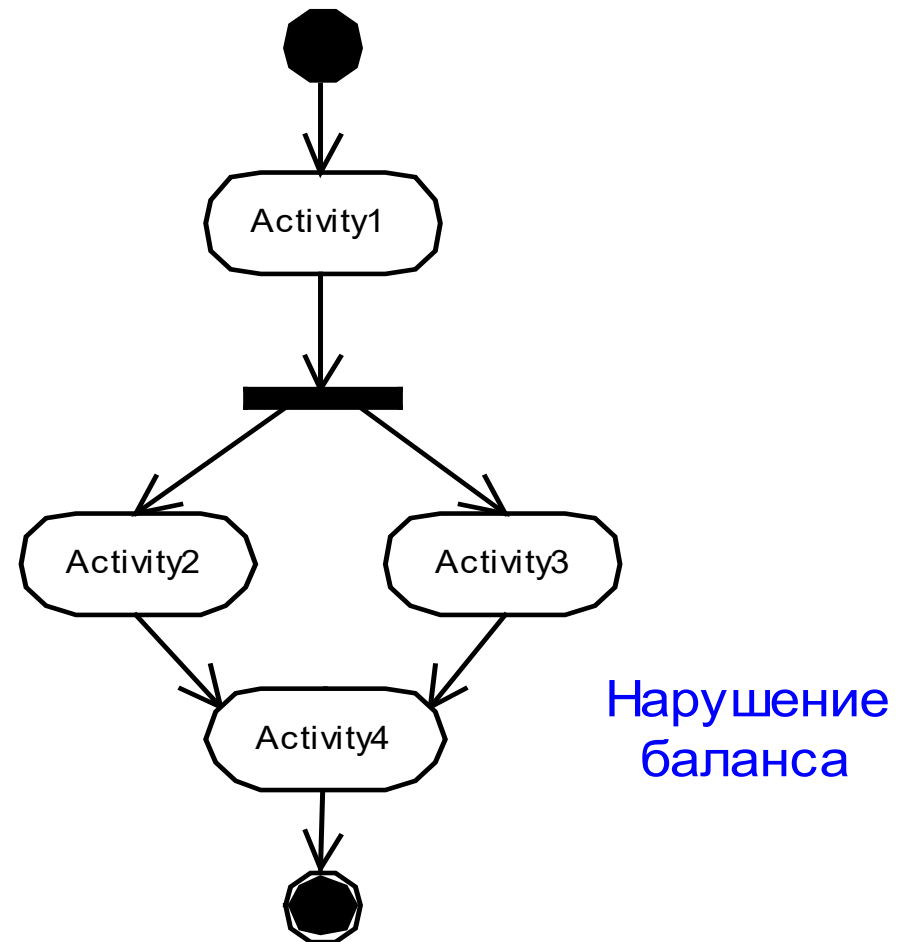
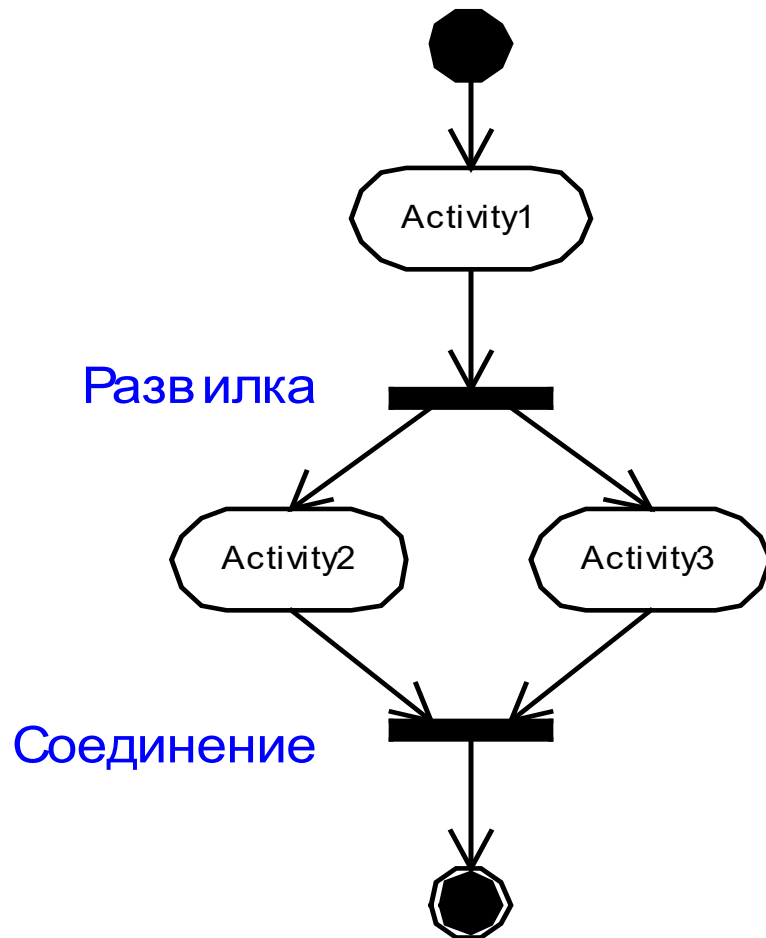
- Пример диаграммы:



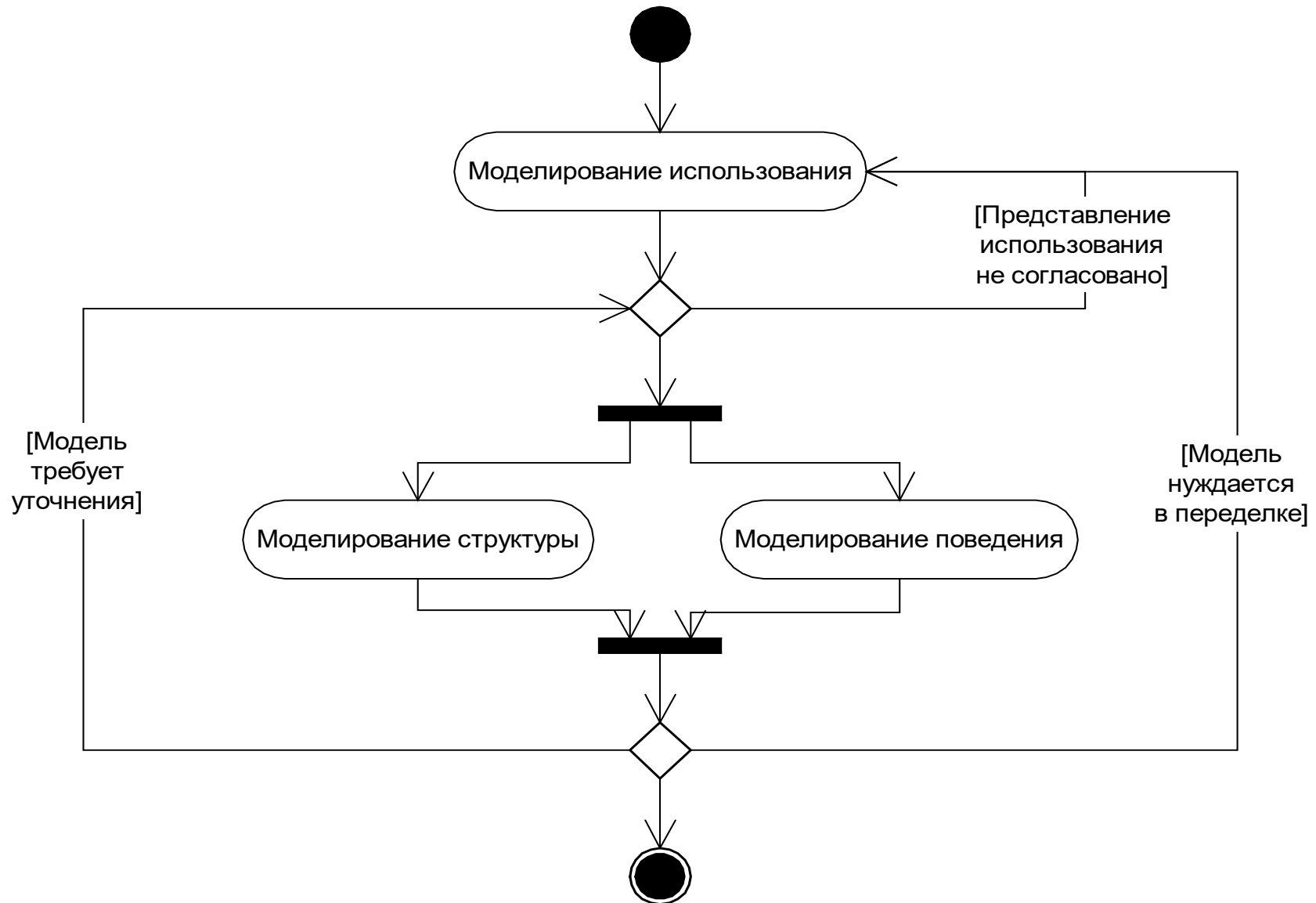
Диаграммы деятельности (*activity diagram*)



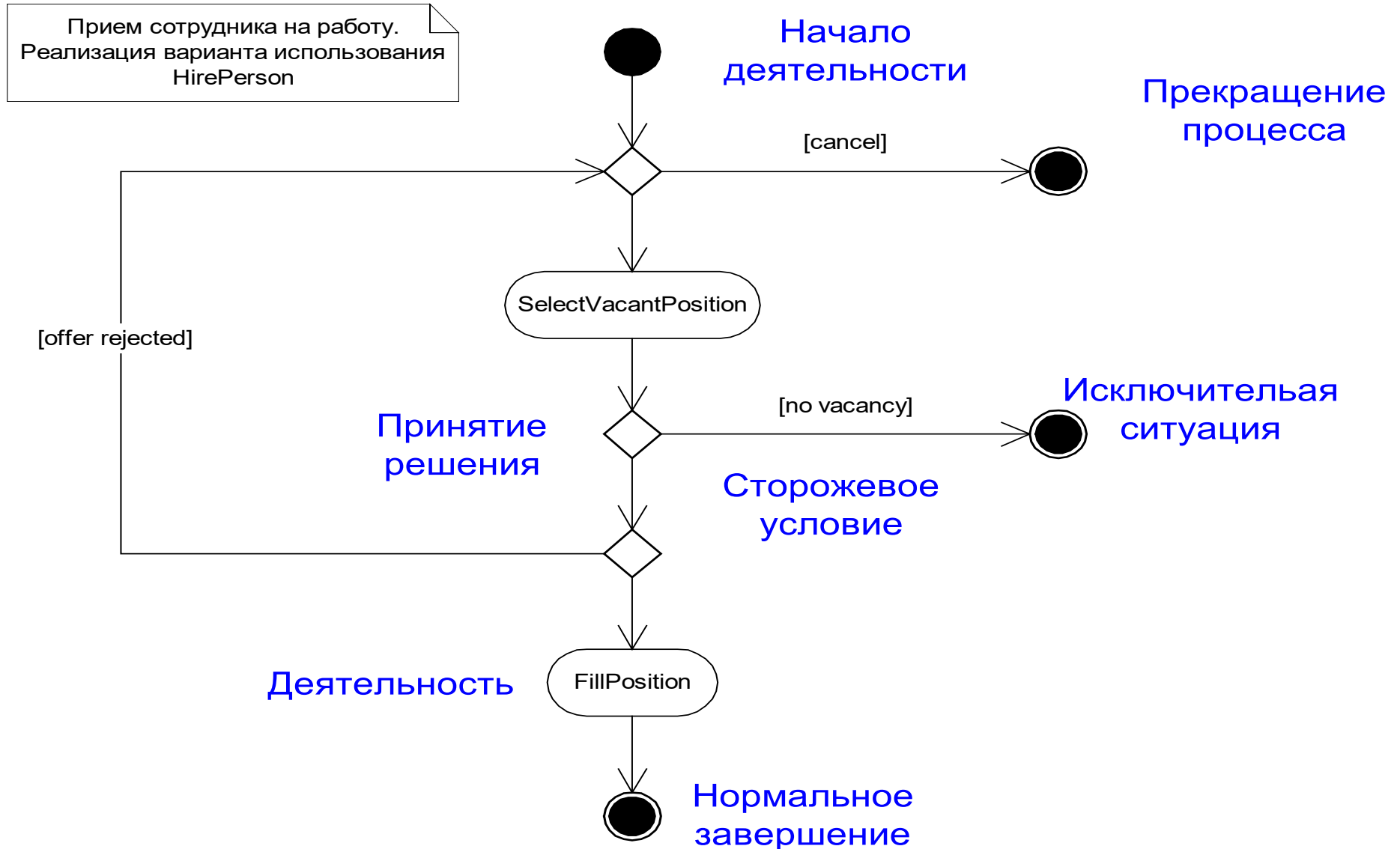
Баланс развилки и соединений



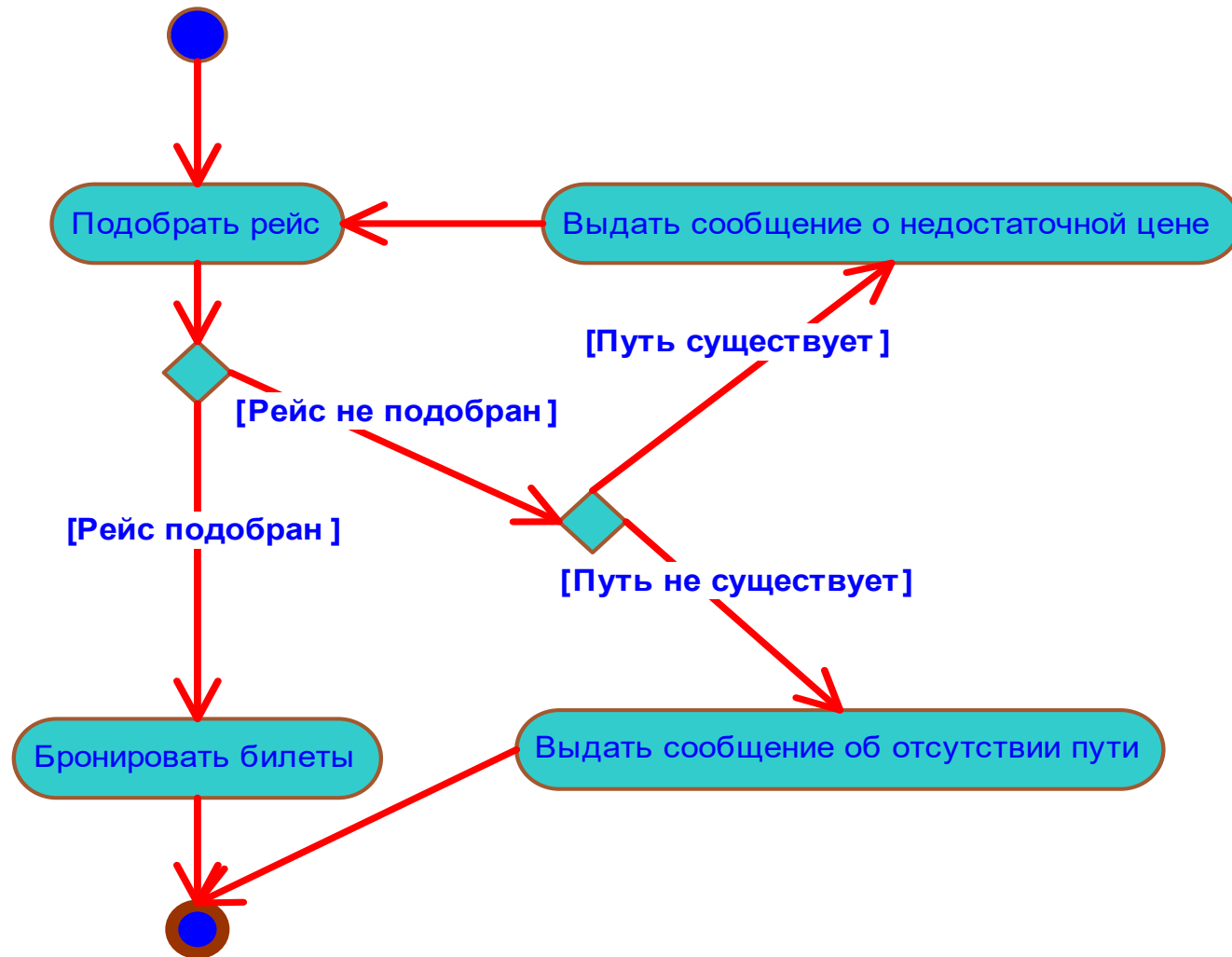
Итеративный процесс моделирования



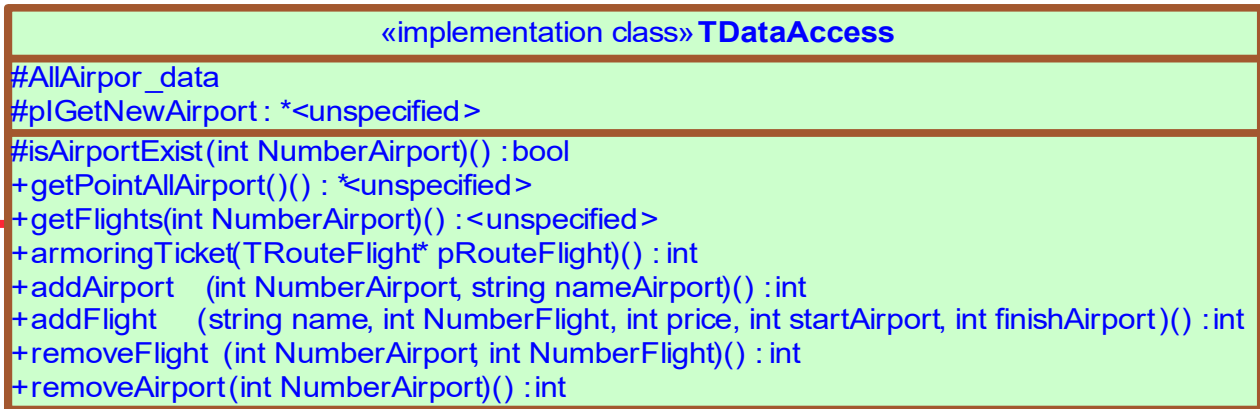
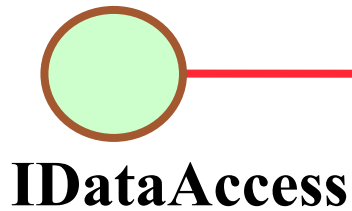
Реализация диаграммами деятельности



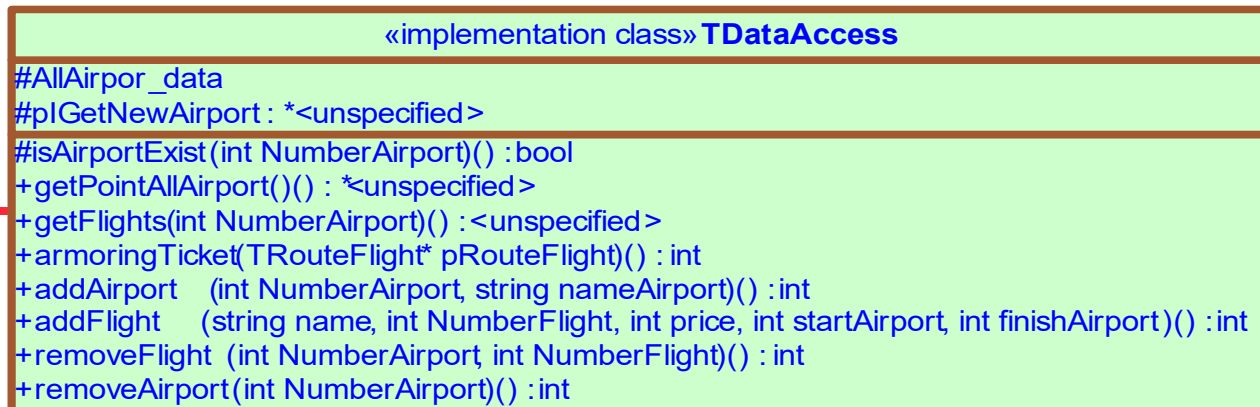
Диаграммы деятельности (*activity diagram*)



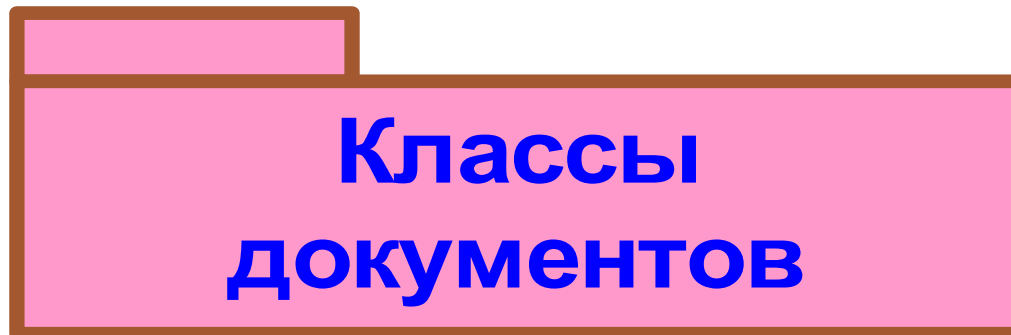
- **Интерфейс** определяет границу между спецификацией того, что делает абстракция, и реализацией того, как она это делает.
- **Интерфейс** — это набор операций, используемых для специфицирования услуг, предоставляемых классом или компонентом.
- Смысл использования: отделить детали реализации от функциональности. «Внешние» методы выносятся в **Интерфейс**.



«interface» **IDataAccess**



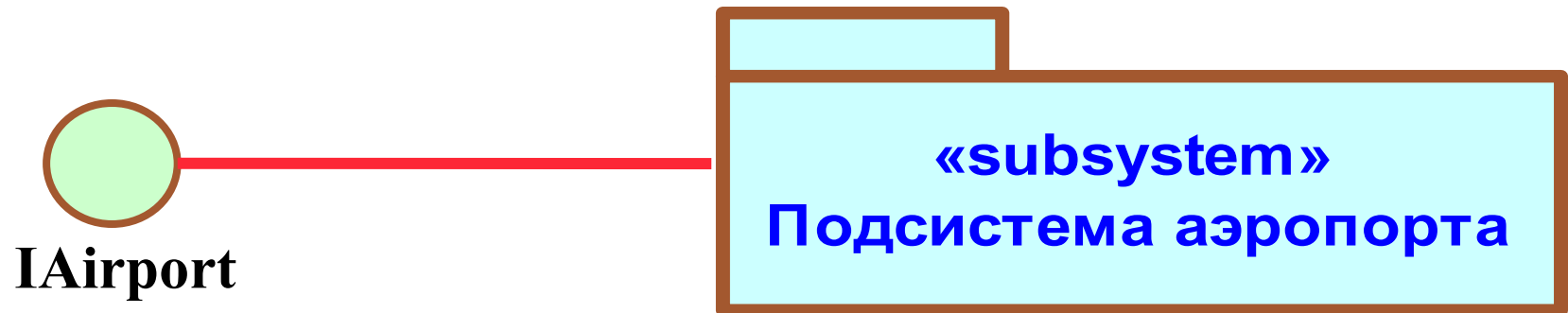
- **Пакет** – структурная единица для группировки элементов модели, в частности, классов.
- **Пакет** – это способ организации элементов модели в более крупные блоки, которыми впоследствии позволяется манипулировать как единым целым [3].
- Хорошо спроектированный пакет группирует семантически близкие элементы, которые имеют тенденцию изменяться совместно [3].



- **На этапе проектирования системы классы и пакеты могут объединяться в подсистемы.**
- Подсистема – структурная единица.
- Каждая подсистема имеют свою **область ответственности** и реализует некоторую **функциональность**.
- Подсистема реализует **Интерфейс**, который описывает ее поведение.
- Примеры: подсистема бронирования билетов; подсистема доступа к данным...



Подсистема реализует интерфейс



- **Компонент** — физическая заменяемая часть системы, совместимая с одним набором интерфейсов и обеспечивающая реализацию какого-либо другого.
- Компонент может разрабатываться и тестироваться независимо от системы.
- Виды компонентов:
 - Исходные файлы (.cpp, .h, .java...).
 - Бинарные файлы (.dll, .osx...).
 - Исполняемые файлы (.exe).

- Назначение
 - Перечисление и взаимосвязи артефактов системы
- Сущности
 - Компоненты
 - Интерфейсы
 - Классы
- Отношения
 - Зависимость
 - Ассоциация
 - Реализация

- По смыслу компонент – реализация подсистемы.
- На этапе проектирования – подсистемы. На этапе реализации – компоненты.

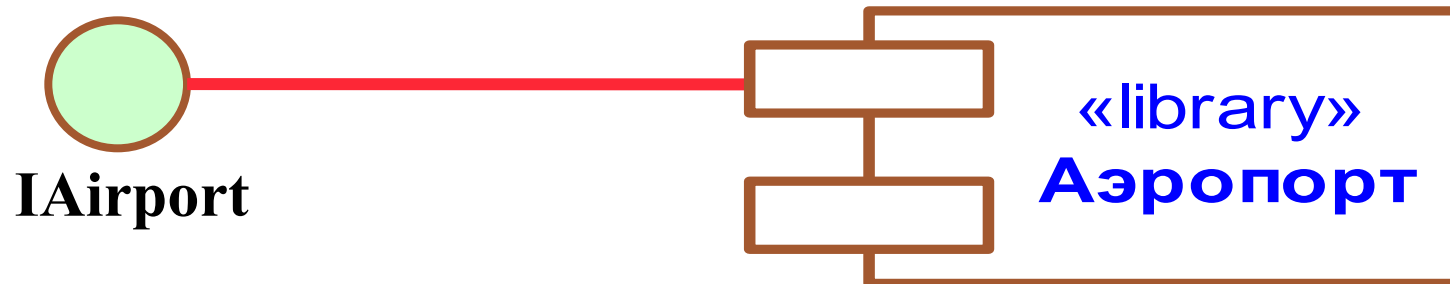
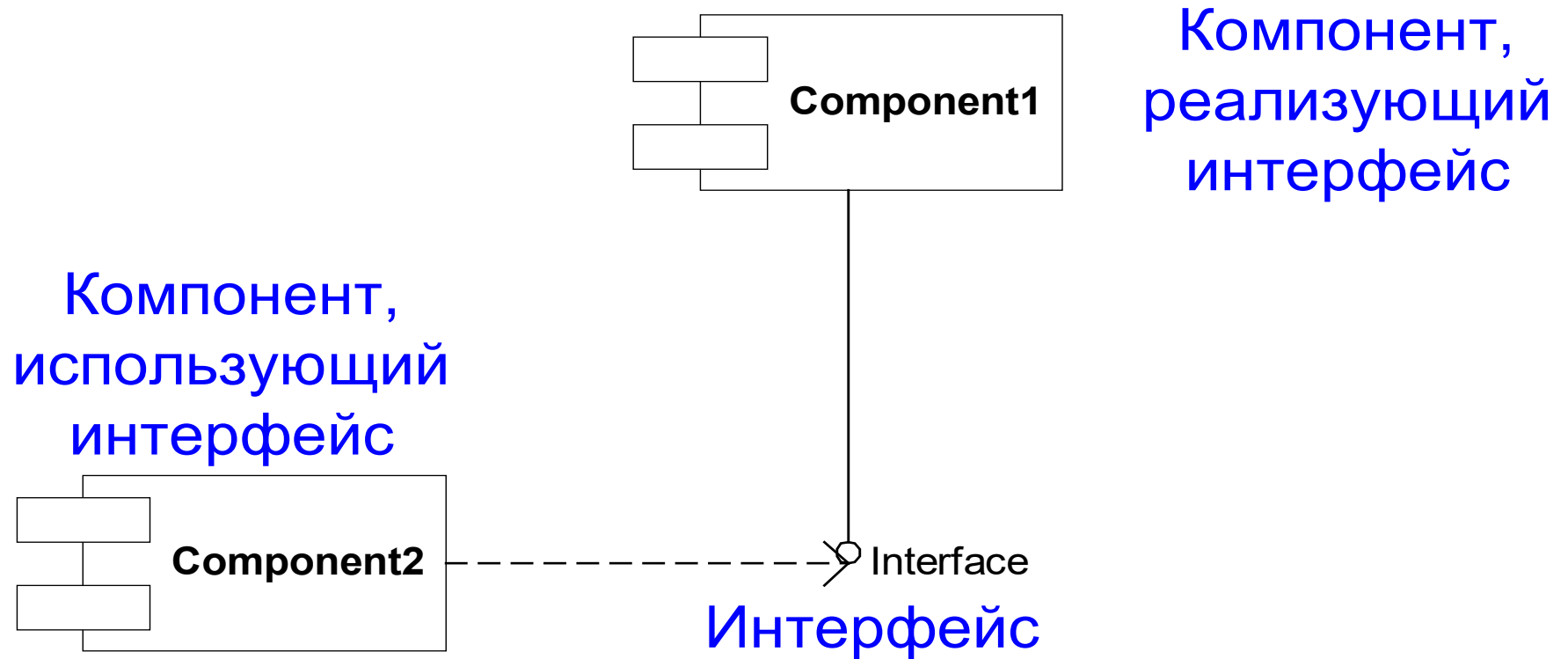


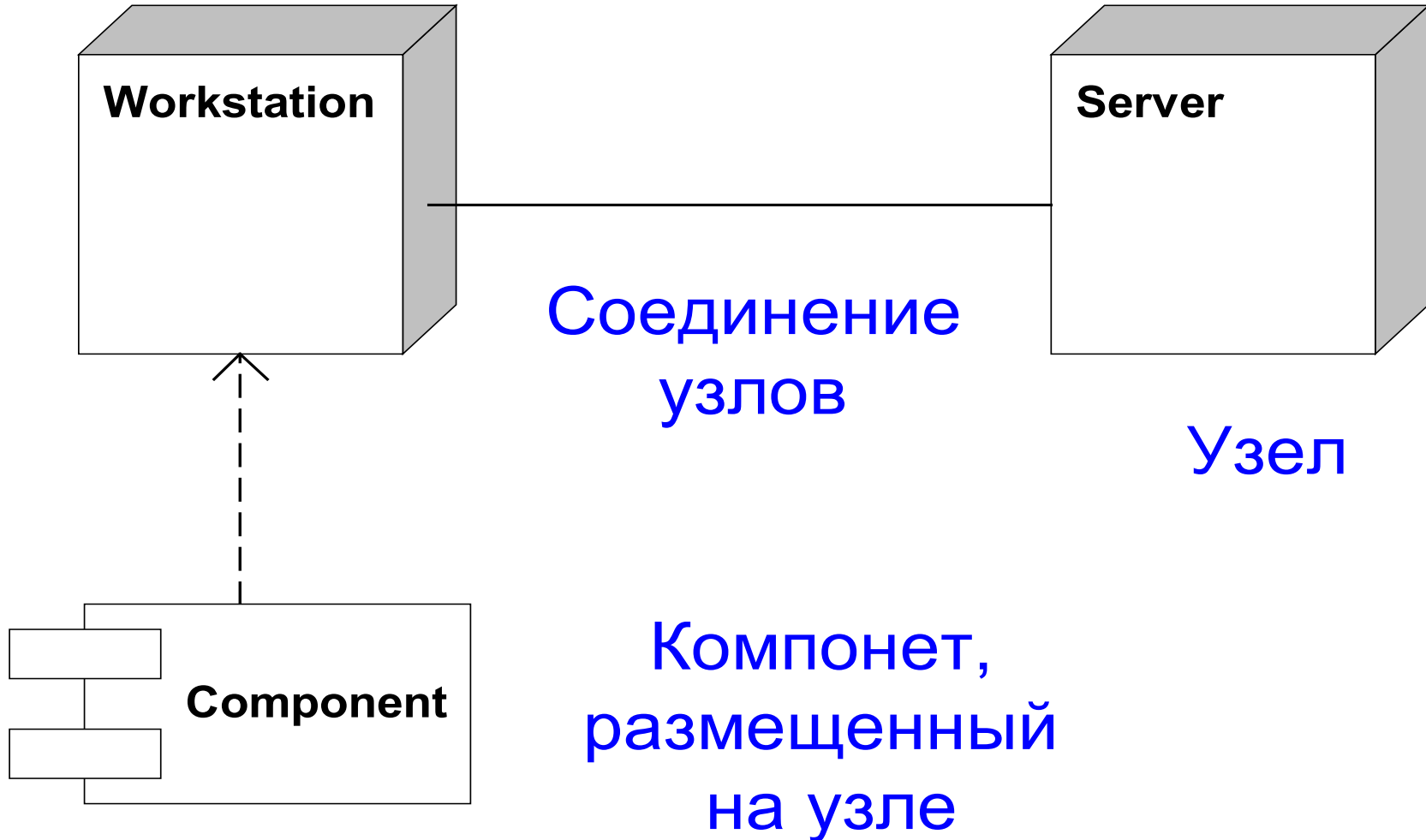
Диаграмма компонентов



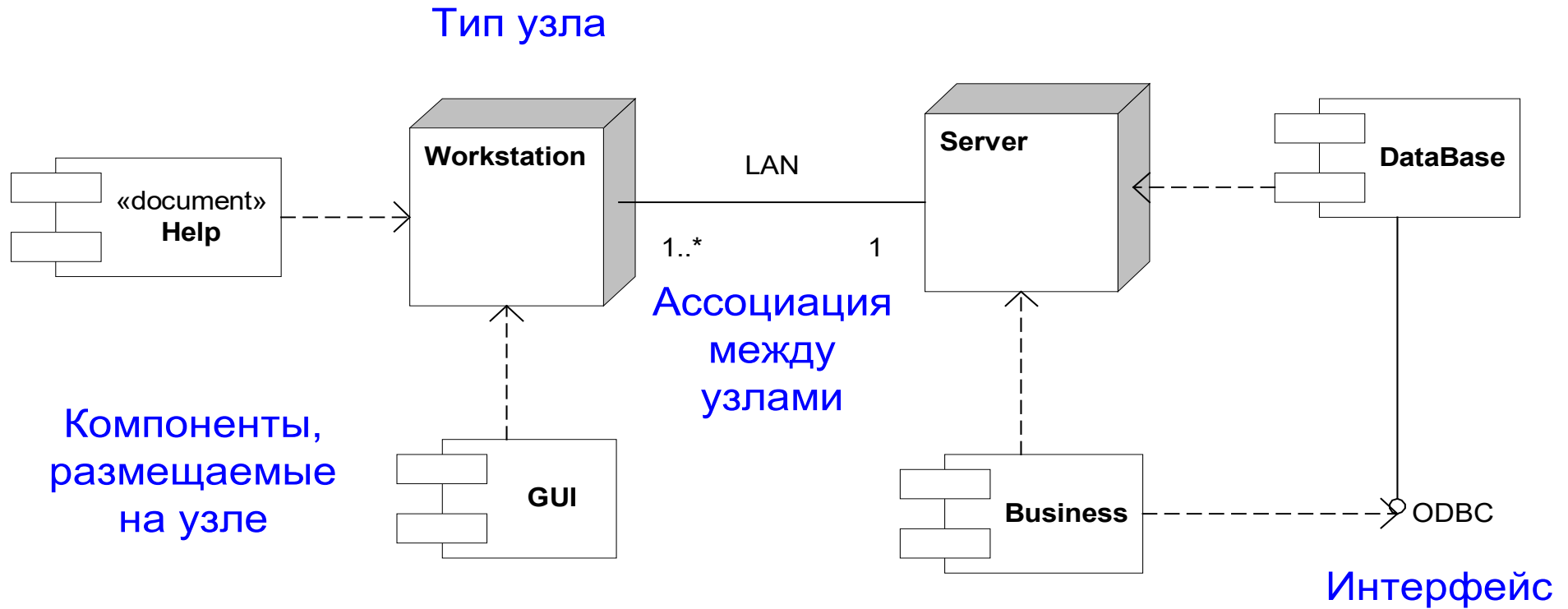
- Назначение
 - Описание топологии развернутой системы
- Сущности
 - Узлы
- Отношения
 - Зависимость
 - Ассоциация



Диаграмма размещения



Пример ИС ОК: тонкий клиент

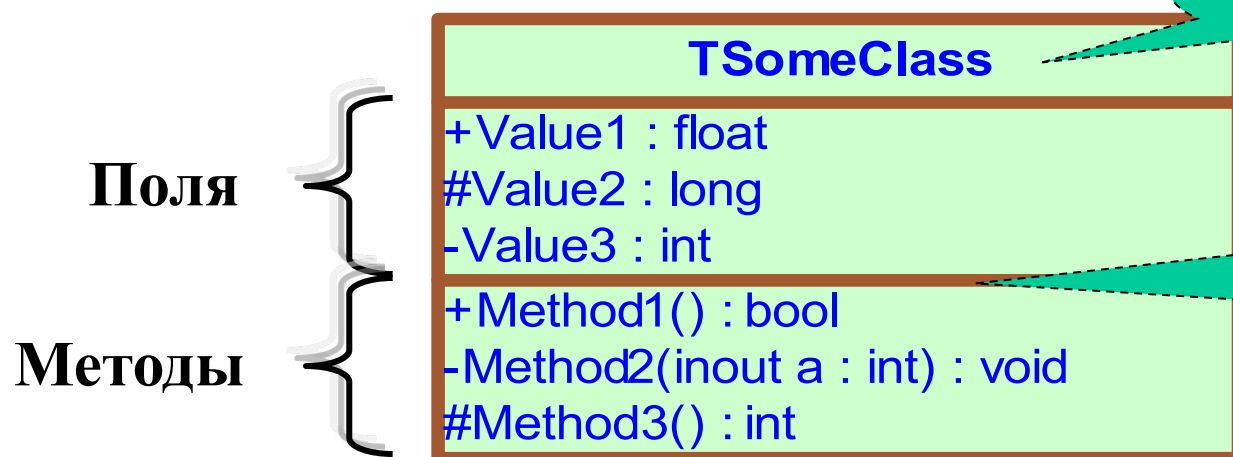
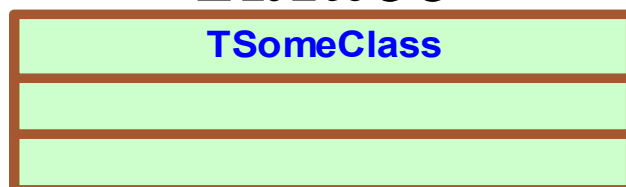


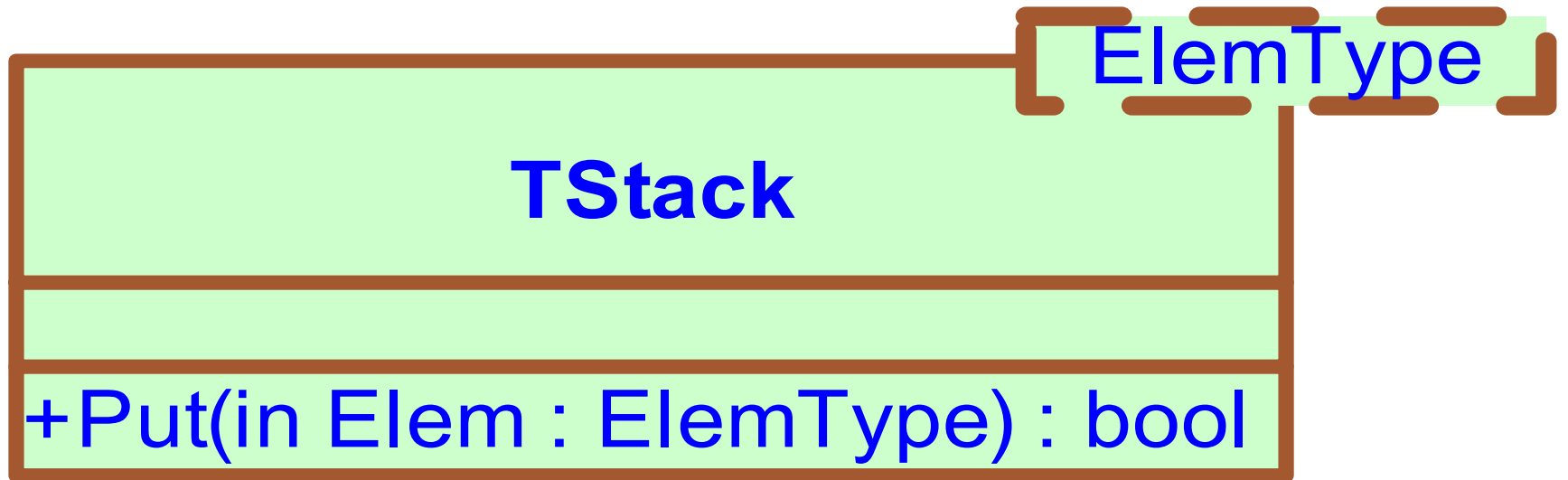
Отношения между элементами модели в UML

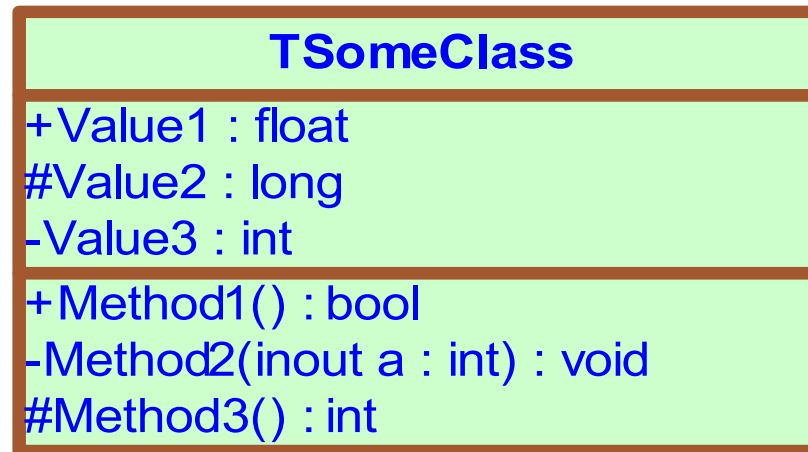
- **Отношения:**
 - **Зависимость;**
 - **Ассоциация;**
 - **Обобщение** (наследование);
 - **Реализация** (для Интерфейса).
- Отношения показывают наличие связей между элементами модели и семантику этих связей.



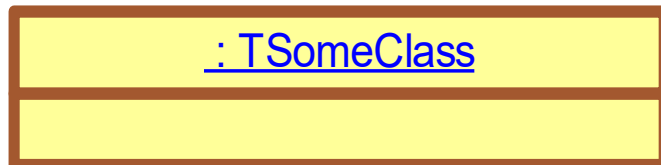
Класс







Объект



Именованный объект



Зависимость в UML

- **Зависимость** — связь между сущностями (классами, объектами).
- **Зависимость** показывает, что изменения в одной сущности могут повлиять на другую сущность.



TFirst зависит от TSecond

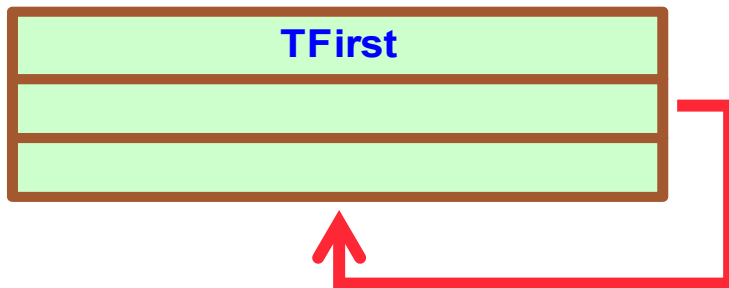
- **Зависимость** — не структурная связь. Возникает через локальную, глобальную переменные или параметр метода.

Ассоциация в UML

- **Ассоциация** – связь между сущностями (классами, объектами).
- **Ассоциация** показывает наличие структурной связи между экземплярами (объектами).
- Связь через поле класса. Направление может быть не указано (двусторонняя связь).



TFirst содержит поле, связанное с TSecond



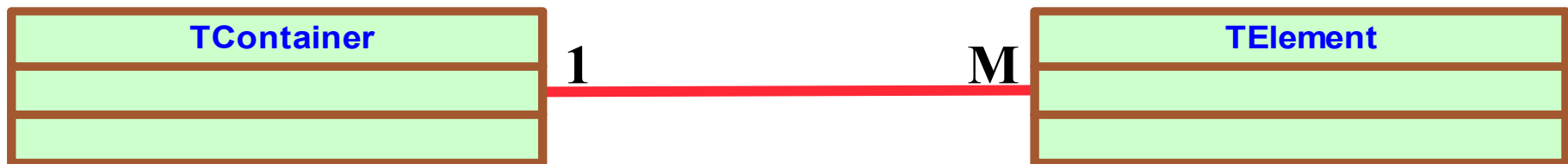
TFirst содержит поле, связанное с TFirst

- Заметим, что наличие направления связано с понятием **Навигация**.
- Навигация означает, что в направлении стрелки один объект **«видит»** другой, в то время как обратное не выполняется.



TFirst видит TSecond

Кратность – способ конкретизации характера отношения. Показывает тип отношения 1:1, 1:M, N:1, N:M.



*Каждому контейнеру соответствует M элементов.
Каждому элементу соответствует 1 контейнер.*

Таблица кратностей в UML

Вид кратности	Значение
* или 0..*	≥ 0
1..*	≥ 1
	обычно 0 или 1
1	Ровно 1
3,5..6	{3,5,6}

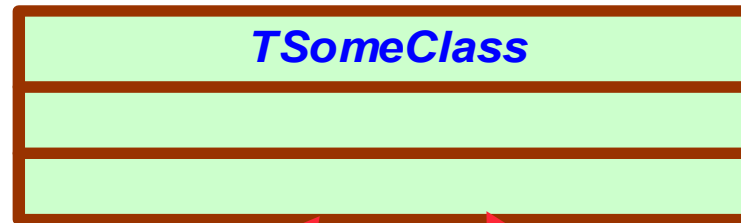


Частные случаи ассоциаций: агрегация и композиция

- **Агрегация** предполагает, что 0 или более объектов одного типа включены в 1 или более объектов другого типа.
- **Композиция** – вариант агрегации, в котором каждый объект второго типа может быть включен ровно в 1 объект первого типа.



Предок



Потомки

