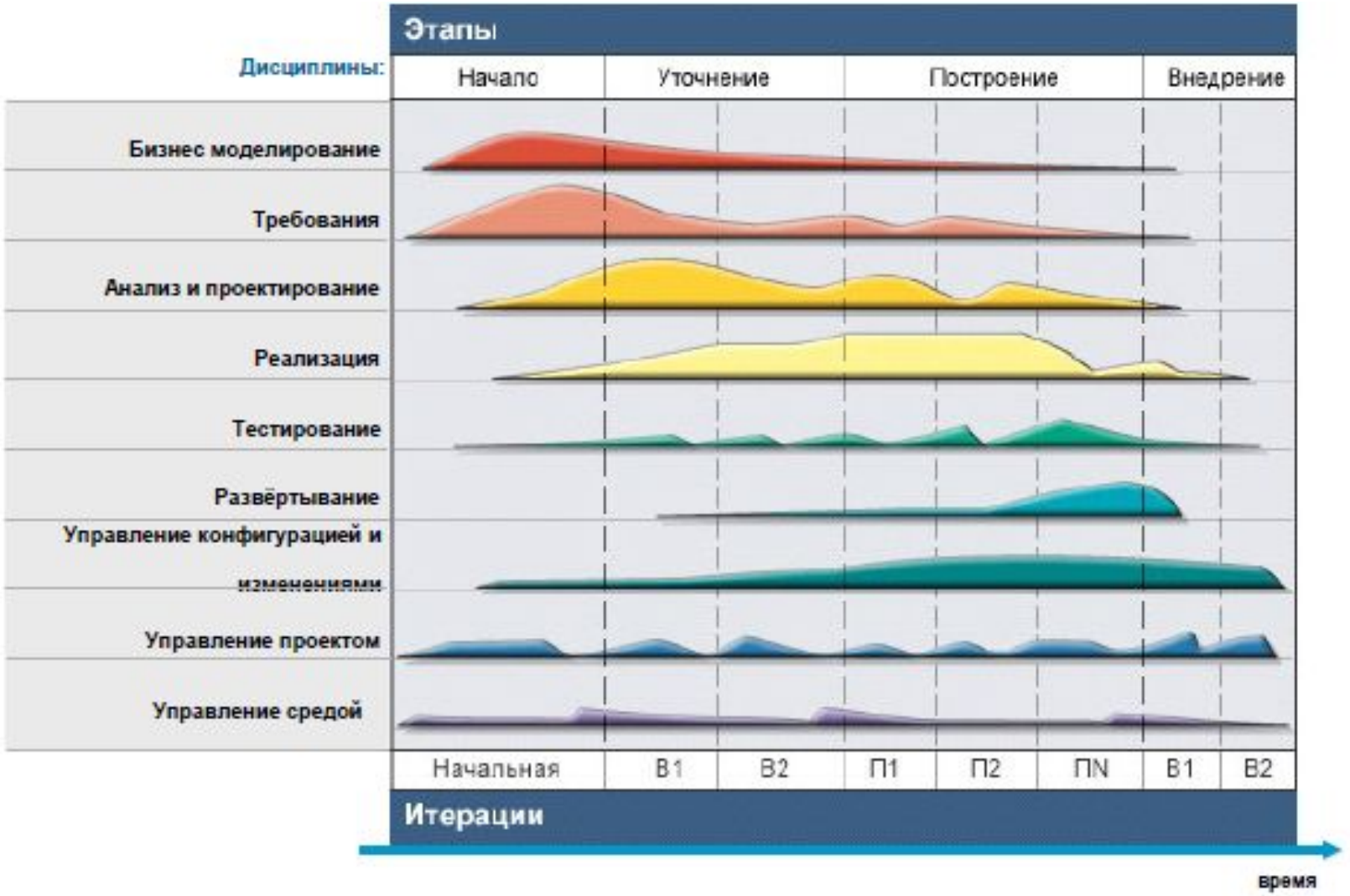
Уточнение ИС (развитие, проектирование - Elaboration) в RUP

д.э.н., проф. Тельнов Ю.Ф.

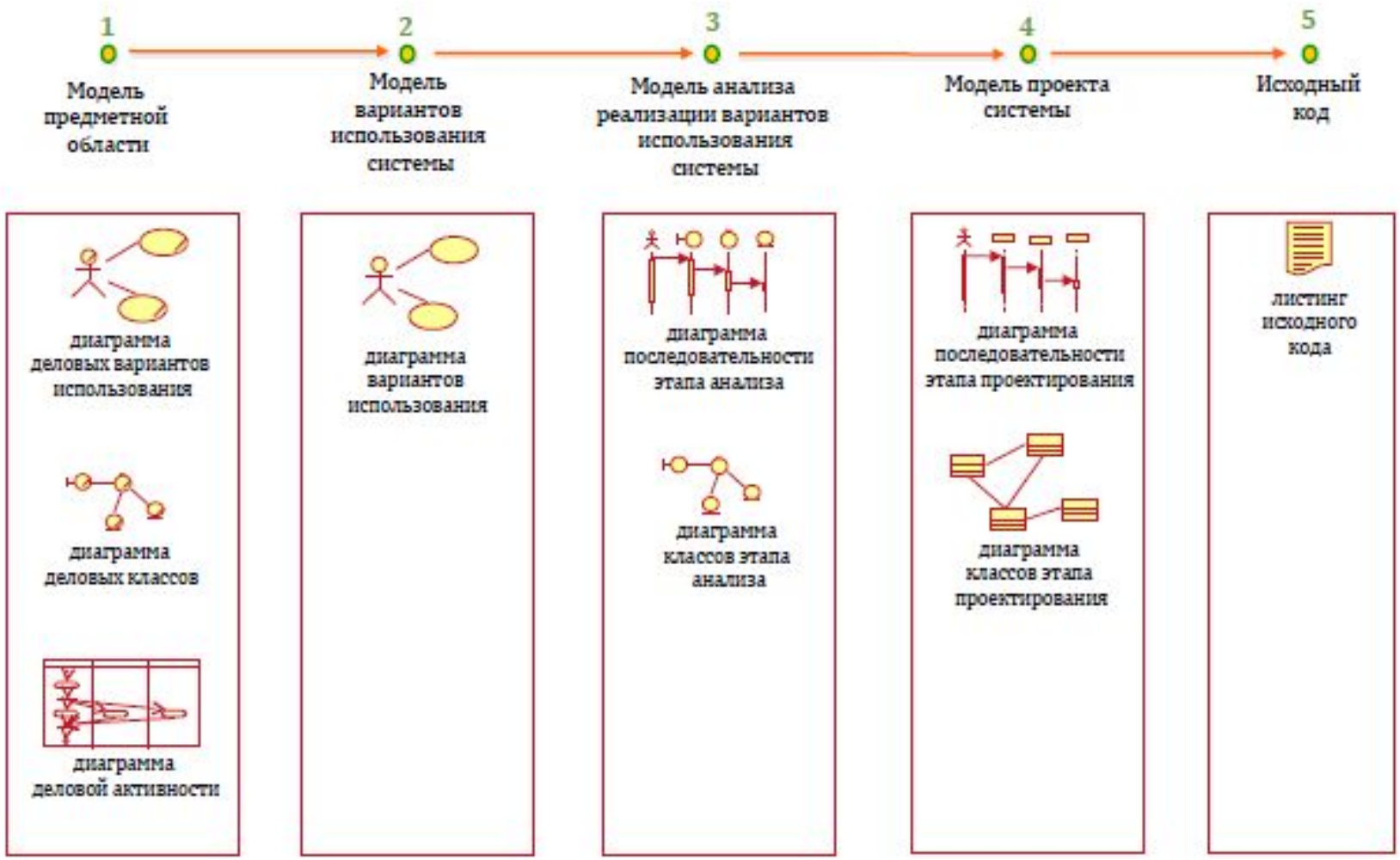
Архитектура RUP



Вопросы

1. Цели (задачи) фазы «Уточнение» (Проектирование - Elaboration)
2. Роль архитектора в фазе «Уточнение (проектирование)»
3. Технология проектирования Моделирование реализации вариантов (прецедентов) использования в RSA:
   - Диаграммы взаимодействий (interaction):
     - Последовательностей (sequence)
     - Коммуникаций (communication)
   - Диаграммы классов (class)
   - Диаграммы активностей (activity)
   - Диаграммы жизненного цикла объектов (life cycle)

Цепочка моделей при разработке ПО



1. Цели фазы «Уточнение – (Elaboration – Развитие, Проектирование)

1. **Цель** – выбор и создание основы архитектуры системы, которая должна стать фундаментом для разработки всей системы в следующей фазе
2. **Программная архитектура** – это набор существенных решений, касающихся организации программной системы:
   - Выбор структурных элементов и их интерфейсов (классов)
   - Поведение, определяемое взаимодействием этих элементов (методов)
   - Объединение этих структурных и функциональных элементов в более крупные подсистемы (компоненты)
   - Архитектурный стиль организации (принципы, методы, инструментальные средства, интерфейсы в соответствии с ИТ-стратегией – относится к встраиванию разрабатываемых компонентов в существующую систему)
3. **Архитектура определяется**, исходя из самых существенных требований (вариантов использования) и оценки рисков:
   - Риски, связанные с требованиями (те ли приложения?)
   - Риски, связанные с архитектурой (те ли решения?)
   - Риски, связанные со стоимостью и сроками (ресурсами)
   - Риски, связанные с процессом и инструментальной средой (технологии разработки)

Архитектурно значимые решения оказывают длительный эффект на

Задачи фазы уточнения (проектирования):

1. Более глубоко понять требования
2. Спроектировать, реализовать и проверить базовую архитектуру
3. Снизить существенные риски и дать более точную оценку сроков и стоимости
4. Уточнить прецедент разработки и установить среду разработки

Рецензирование проекта. Веха архитектуры жизненного цикла

1.1. Более глубокое понимание требований

- Детальное описание большинства прецедентов использования (оставшиеся 80 % use-case)
- Создание прототипа пользовательского интерфейса для главных прецедентов использования
- Выделение дополнительных прецедентов использования
- Обновление глоссария (тезауруса системы)

1.2. Спроектировать, реализовать и проверить базовую архитектуру

- Выбор наиболее важных строительных блоков (компонентов) системы (подсистем, пакетов, прецедентов использования) и их интерфейсов и выбор проектных решений по созданию, покупке или повторному использованию компонентов строительных блоков
- Описание, как эти строительные блоки (компоненты) будут взаимодействовать между собой в ходе выполнения наиболее важных сценариев
- Реализация и тестирование прототипа архитектуры с точки зрения оценки рисков и важнейших качественных характеристик – производительности, масштабируемости, стоимости и др. – **Создание исполняемой архитектуры (эволюционирующего прототипа),** могут быть «заглушки», строится каркас системы.

Вопросы для принятия проектных решений:

- Соответствие компонентов требованиям
- Стоимость и юридические условия приобретения компонентов
- Поддержка компонентов по мере развития системы
- Доступ к исходному коду для внесения изменений
- Документирование компонентов: описание проектных решений, способы использования и тестирования

1.3. Снизить существенные риски и дать более точную оценку сроков и стоимости

- Менеджер проекта проводит обновление Экономического обоснования и Плана разработки проекта, добавляя дополнительные планы, списки рисков, ключевые управленческие артефакты
- Менеджер проекта проводит совещание с ключевыми заинтересованными лицами с целью рассмотрения экономического обоснования, списка рисков, концепции и плана разработки системы

1.4. Уточнить прецедент разработки и установить среду разработки

- Внесение изменений в прецедент разработки: какие артефакты нужно дополнительно создать, какие использовать шаблоны, как документировать информацию
- При необходимости проводится обучение членов команды

Рецензирование проекта. Веха архитектуры жизненного цикла.

- Являются ли Концепция и требования проекта устойчивыми?
- Является ли архитектура устойчивой?
- Проверены ли основные подходы, которые будут использоваться при тестировании и оценке системы?
- Устранены ли основные риски в процессе тестирования и оценки системы?
- Являются ли планы итераций на фазу Построение достаточно подробными и точными?
- Проведено ли успешное согласование текущего плана и текущей архитектуры с заинтересованными лицами?
- Приемлемо отклонение текущих расходов от запланированных?

2. Роли участников проекта создания (модернизации ИС)

Требования	 Аналитик  Заинтересованные лица
Архитектура	 Архитектор
Разработка	 Разработчик
Тестирование	 Тестировщик
Управление проектом	 Руководитель проекта
Управление конфигурацией и изменениями	 Разработчик

2. Роль архитектора в фазе проектирования

- Архитектор программного обеспечения направляет и координирует решение технических задач и разработку артефактов (программ, баз данных, документации и т.д.), а также координирует принятие ряда ключевых решений, касающихся технологий, структуры и организации программной системы (ИС)
- Архитектор должен быть широко образован, обладать зрелостью, видением и глубоким опытом, который позволяет быстро решать проблемы и выносить грамотные суждения при отсутствии полной информации

Требования к архитектору ПО

- Лидерство, ответственность за решение технических вопросов, взаимодействие с менеджером проектов
- Коммуникабельность, доверие проектной команды, менеджера проекта, заказчика, сообщества пользователей и менеджмента
- Ориентация на цели и предусмотрительность с непреклонной ориентацией на результат, прагматизм, а не перфекционизм.
- Опыт в предметной области и в области разработки ПО, глобальное видение проекта

Список активностей (задач) архитектора ПО

1) Работа с требованиями
2) Совершенствование архитектуры
3) Поддержание архитектурной целостности

1).Работа с требованиями (работа с аналитической моделью)

- Расстановка прецедентов использования в порядке приоритета при планировании итерации (совместно с менеджером проекта) – на основе анализа рисков, архитектурной важности, наличия ресурсов)
- Архитектурный анализ – построение логической структуры моделей и классов, устраняя избыточность
- Конструирование архитектурного прототипа «подтверждения концепции» в целях уяснения границ системы, проверки её осуществимости и оценки реальности риска и эффективности решения

2). Совершенствование архитектуры

- Определение механизмов проектной модели – выбор программной инфраструктуры (промежуточного ПО: СУБД, программных сред,…), выбор конкретных механизмов реализации
- Определение элементов проектной модели – выбор ключевых элементов проектной модели
- Объединение имеющихся элементов проектной модели – повторное использование программного кода (других платформ, проектов, систем). Переработка имеющихся элементов и интеграция в новую систему
- Структурирование модели реализации – размещение повторно используемых элементов, разрабатываемых артефактов: исходных и исполняемых кодов, БД, документации в проектном репозитории; организация системы контроля конфигураций и версий
- Описание распределения и архитектуры времени выполнения (календарный план-график работ) с учетом:
  - Анализа требований к параллелизму
  - Определения процессов и потоков и механизмов взаимодействия между ними
  - Распределения ресурсов для координации между процессами
  - Связывания процессов с программной средой
  - Распределения элементов моделей среди проектов

3). Поддержание архитектурной целостности

- Разработка руководств по проектированию (как использовать элементы архитектуры)
- Разработка руководств по программированию с учетом свойств выбранного языка программирования
- Рецензирование архитектуры (review):
  - выявить все неизвестные риски для сроков и бюджета
  - определить архитектурные недостатки
  - выявить несоответствия требованиям
  - оценить одно или несколько качественных характеристик ПО
  - выявить возможности повторного использования кода

3. Технология проектирования архитектуры (подсистем, ключевых компонентов и их интерфейсов)

- Выбор архитектурно-значимых прецедентов использования (важность, сложность, риски, основание для других прецедентов)
- Проектирование критичных прецедентов использования (итерации от аналитической модели к проектной)
- Объединение и сборка готовых классов в пакеты
- Проверка архитектурного охвата всех основных артефактов (классов и интерфейсов) - применяемости
- Проектирование базы данных
- Определение и применение архитектурных механизмов (типовых решений, шаблонов)
- Интеграция компонентов – выпуски версий (релизов)
- Тестирование критических сценариев (по производительности, нагрузке, интерфейсам, нефункциональным требованиям)

Критерии выбора архитектурно-значимых компонентов (прецедентов использования)

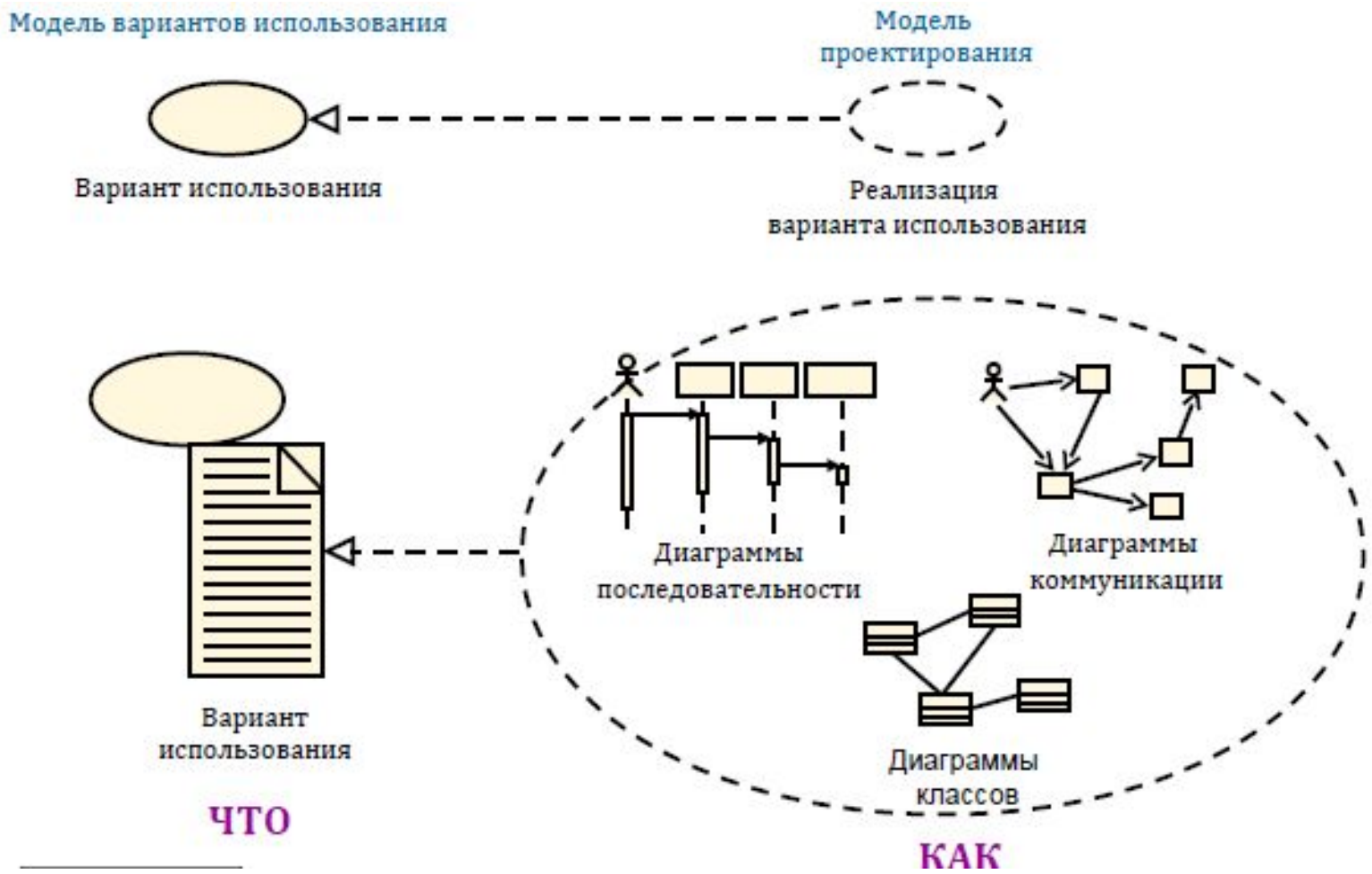
- Сложность, рискованность реализации
- Реализация критически важных параметров: производительность системы, время реакции на запрос.
- Проверка использования бизнес-сущностей (объектов) – принцип архитектурного охвата

Проектируя, реализуя, тестируя и документируя архитектурные механизмы – можно решить самые общие и трудные проблемы, которые потом будут использоваться как готовые решения для реализации всех прецедентов.

Проектирование критично важных прецедентов использования (от аналитической модели к проектной модели)

1. Создать предварительную схему объектов аналитической модели на основе диаграммы взаимодействий
2. Распределить действия по классам объектов аналитической модели для обеспечения функциональности прецедента использований
3. Прорецензировать классы объектов на предмет дублирования функций и взаимодействия
4. Спроектировать классы объектов проектной модели (путем объединения/разъединения классов объектов аналитической модели
5. Детализировать классы объектов проектной модели путем определения атрибутов и методов.

Реализация варианта использования



Декомпозиция функциональных требований (бизнес-функций) на функции компьютерной обработки данных в сценарии прецедента

Функциональное требование (бизнес-функция)	Функция компьютерной обработки данных
Прием документа (заявки, требования, ….)	Заполнение экранной формы (ввод данных)
	Контроль (Проверка) данных
	Запись в базу данных
	Вывод сообщений об ошибках
Формирование документа (договора, заказа, платежного документа, счета ….)	Выбор первичного документа (например, заявки)
	Отбор нормативно-справочной информации (например, ценник)
	Расчет показателей (стоимость=количество*цена)
	Запись документа в БД (распечатка, вывод на экран)

Декомпозиция функциональных требований (бизнес-функций) на функции компьютерной обработки данных

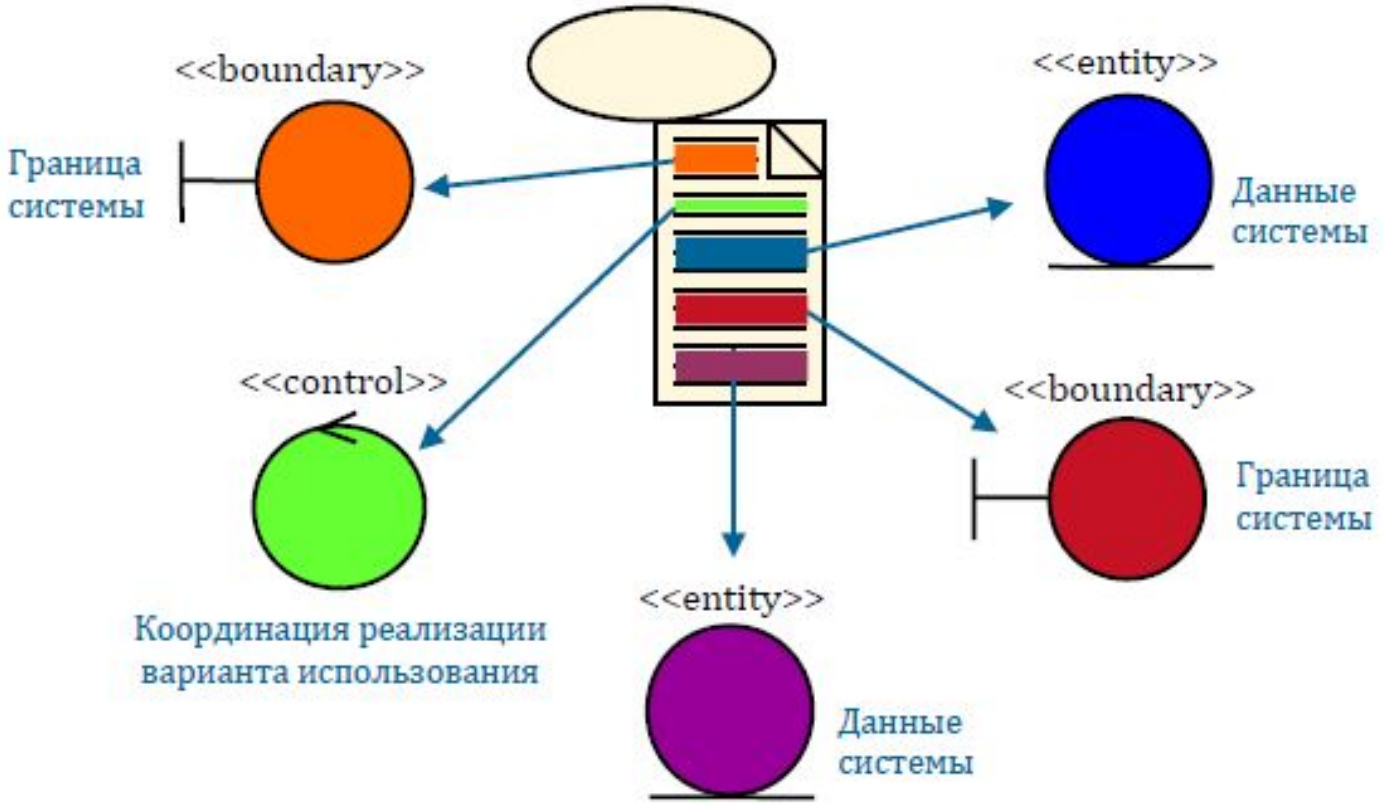
Функциональное требование (бизнес-функция)	Функция компьютерной обработки данных
Формирование отчета	Выбор ключевых признаков из справочников
	Группировка (сортировка) по ключевым признакам
	Подсчет итогов
	Формирование итогового отчета
	Запись в базу данных*
	Печать (вывод на экран)

Диаграммы классов

- **Диаграмма** классов определяет типы классов системы и различного рода статические связи, которые существуют между ними.
- На диаграммах классов изображаются атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами.
- Диаграммы классов отражают взаимодействие между классами системы. Классы можно рассматривать как типы объектов.
- Классы содержат данные и поведение (действия), влияющее на эти данные.

Классы анализа архитектуры системы

- Поведение системы, описанное в варианте использования, может быть распределено среди нескольких классов анализа

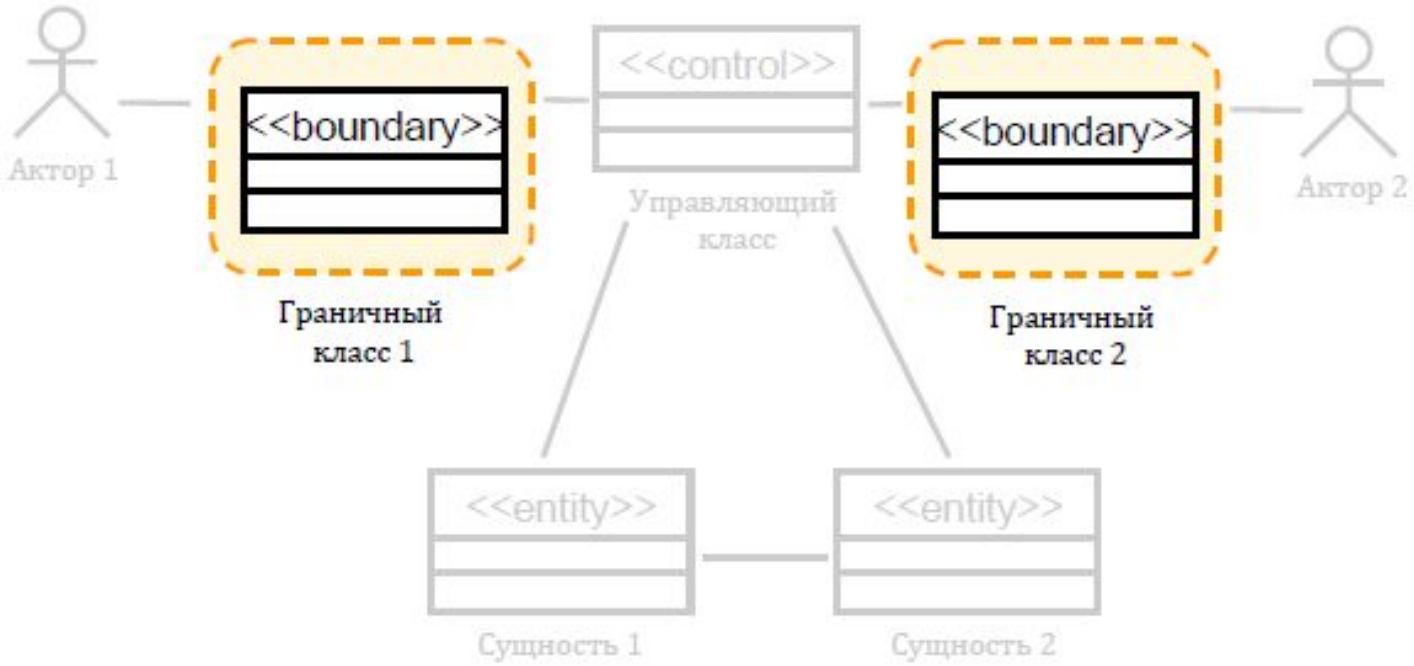


Понятие граничного класса -boundary

- Промежуточное звено между интерфейсом и чем-либо внутри системы
- Возможны несколько типов:
  - Классы интерфейса пользователя
  - Классы интерфейса системы
  - Классы интерфейса устройства

Стереотип граничного класса

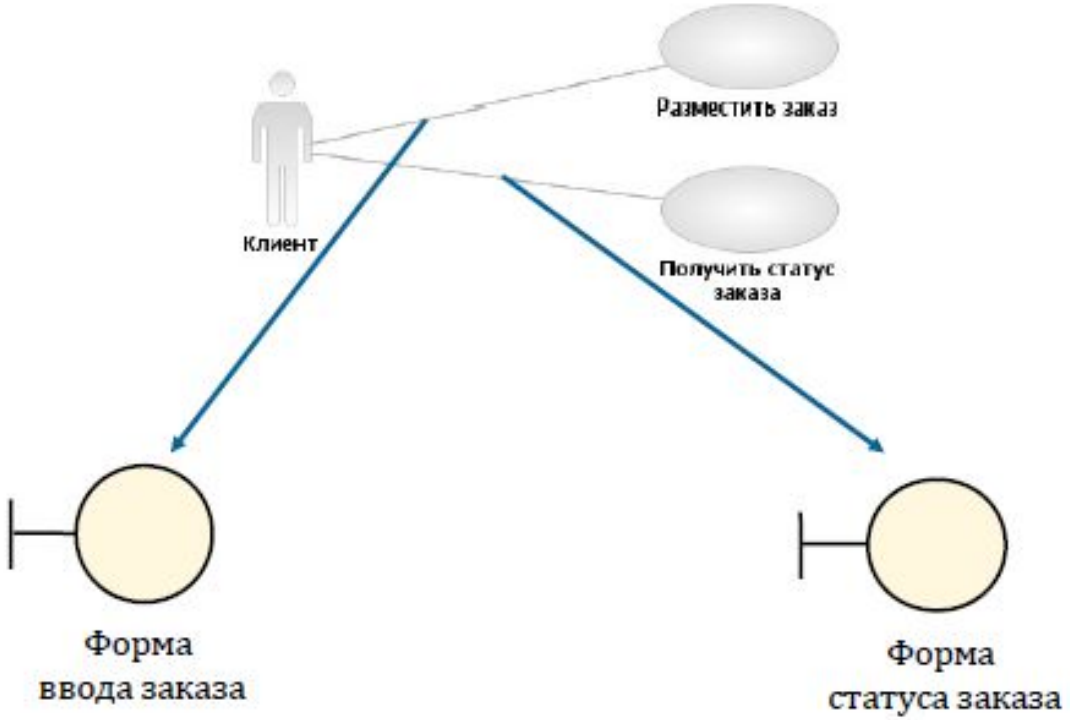
Роль граничных классов



Модель взаимодействия между системой и её средой

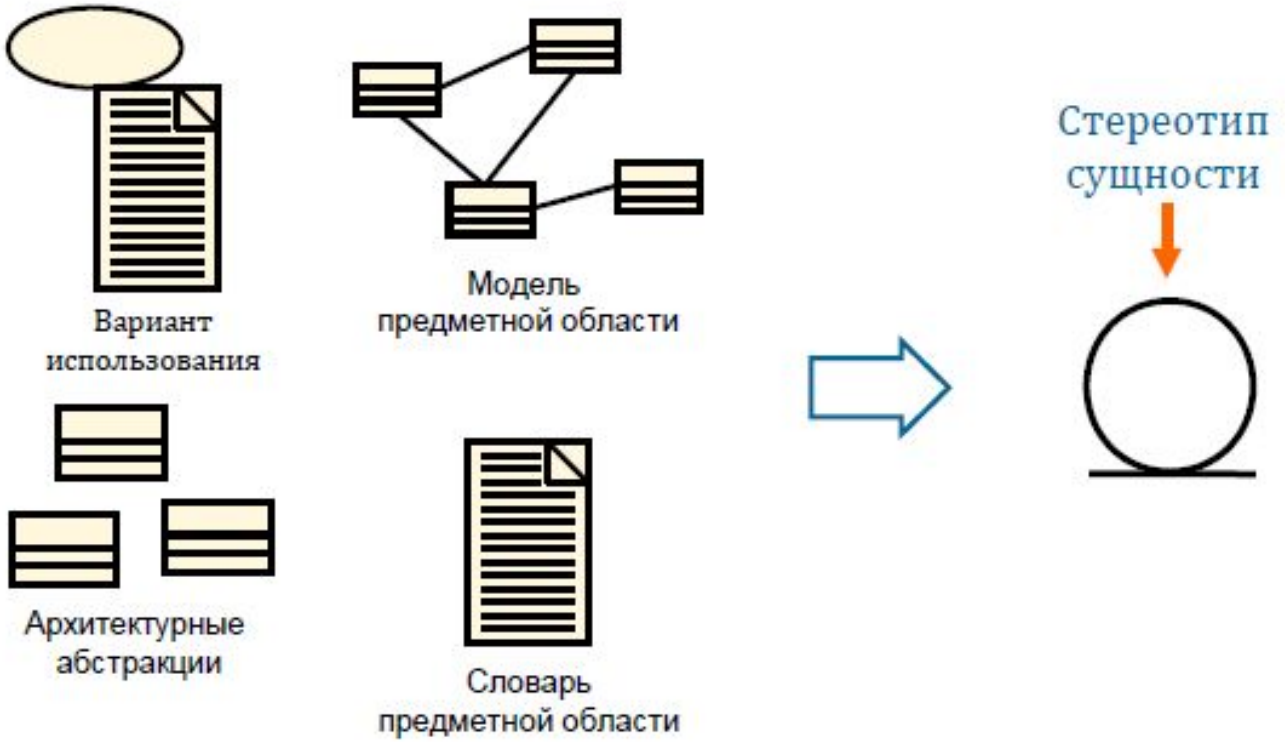
Определение граничных классов

- Один граничный класс для одной пары «вариант использования -актор»

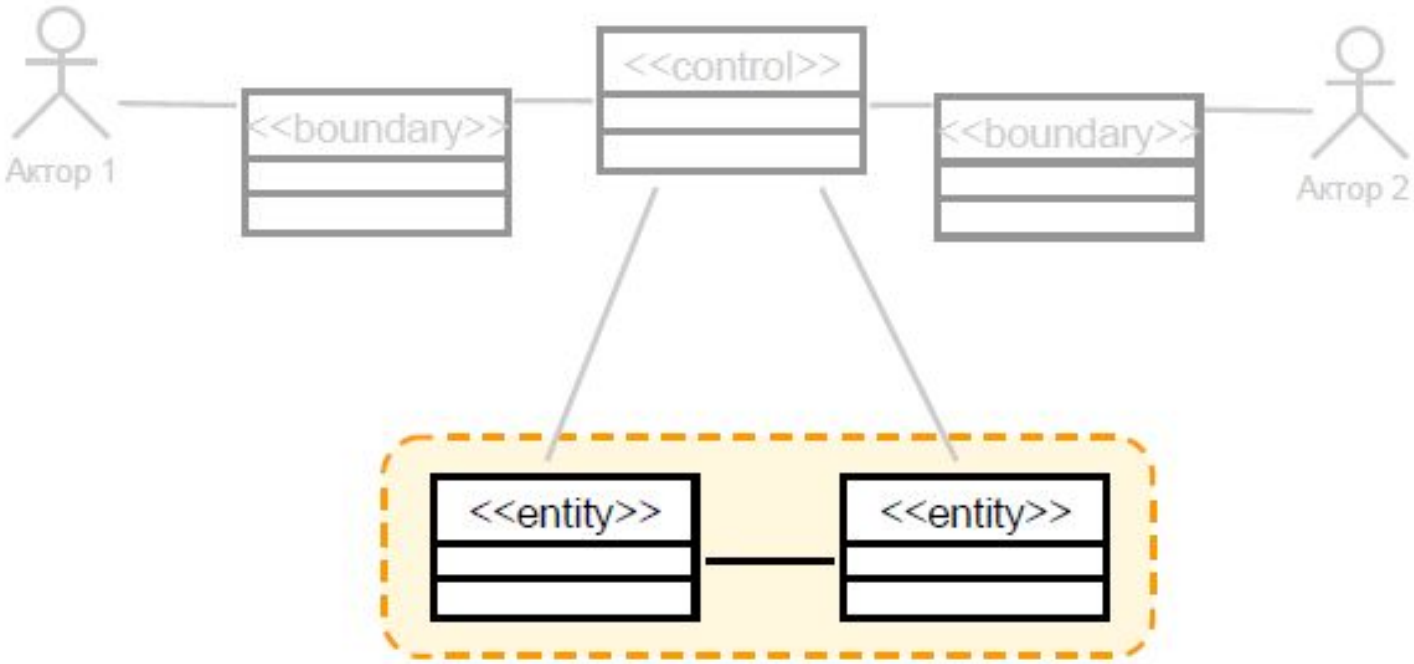


Понятие сущности

- Ключевые абстракции системы

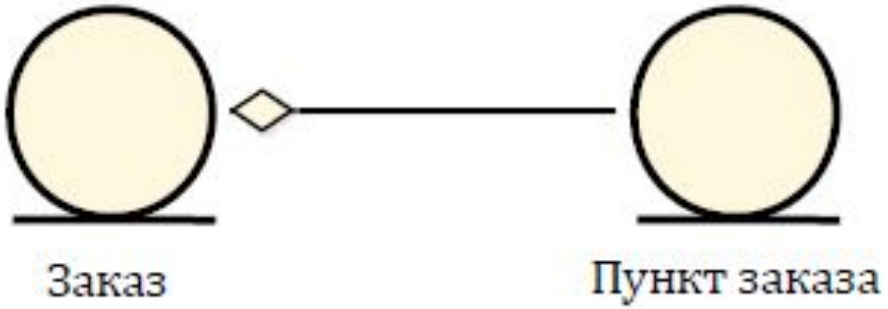


Роль сущности в модели анализа



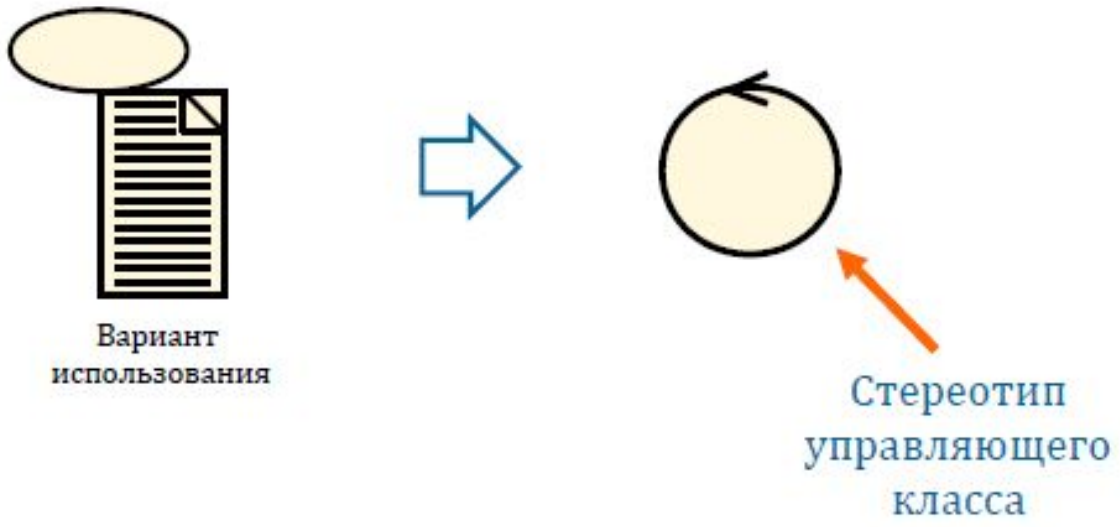
Определение сущностей

- Варианты использования:
  - «Разместить заказ»
  - «Получить статус заказа»
  - «Проверить заказ»

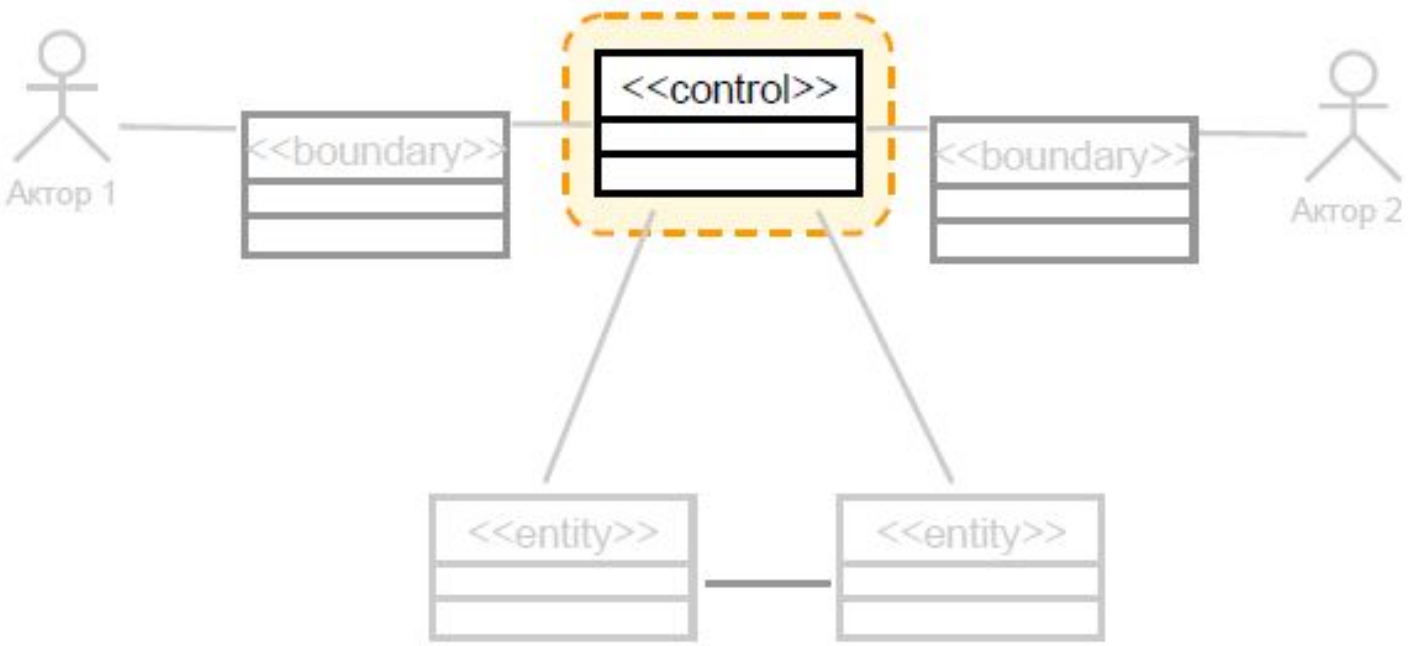


Понятие управляющего класса

- Координатор поведения системы при реализации варианта использования
  - Сложные варианты использования требуют использования нескольких управляющих классов

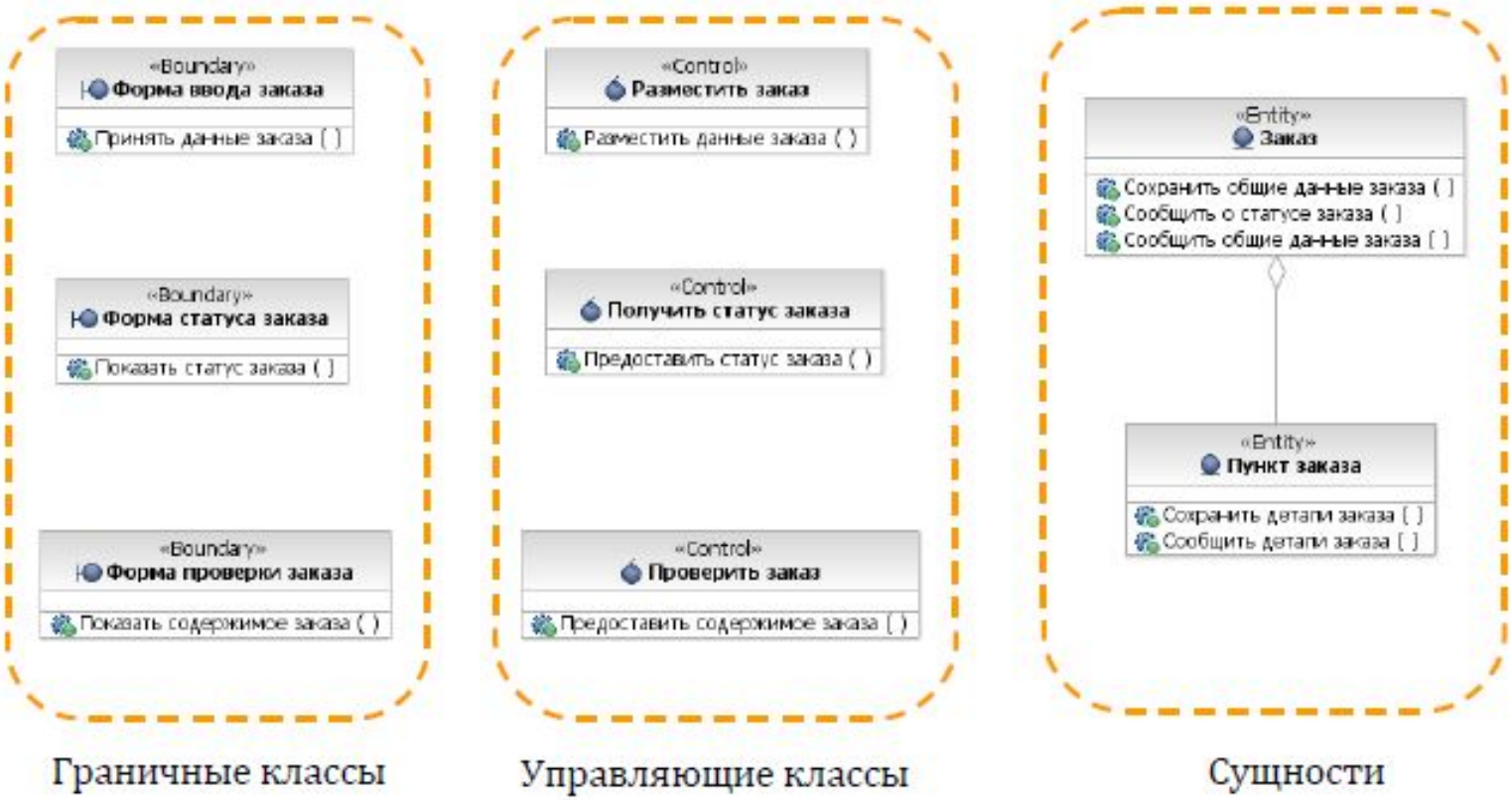


Роль управляющего класса



Координирует поведение варианта использования

Сводный перечень классов



Граничные классы

Управляющие классы

Сущности

Диаграммы взаимодействия

Диаграммы взаимодействия (interaction diagrams) являются моделями, описывающими поведение взаимодействующих групп объектов.

Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного варианта использования. На такой диаграмме отображается ряд объектов и те сообщения, которыми они обмениваются между собой.

Сообщение (message) - средство, с помощью которого объект-отправитель запрашивает у объекта получателя выполнение одной из его операций.

Информационное (informative) сообщение - сообщение, снабжающее объект-получатель некоторой информацией для обновления его состояния.

Сообщение-запрос (interrogative) - сообщение, запрашивающее выдачу некоторой информации об объекте-получателе.

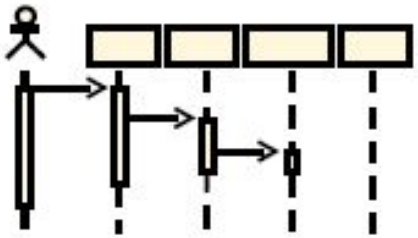
Императивное (imperative) сообщение - сообщение, запрашивающее у объекта-получателя выполнение некоторых действий.

Существует два вида диаграмм взаимодействия: диаграммы последовательности (**sequence diagrams**) и кооперативные диаграммы (**collaboration diagrams**).

Диаграммы Последовательности отражают поток событий, происходящих в рамках варианта использования.

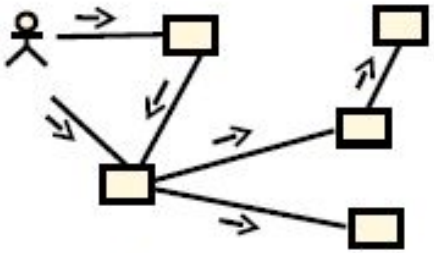
Диаграммы взаимодействия

- Диаграмма последовательности
  - Взгляд на взаимодействие объектов с акцентом на последовательности обмена сообщениями



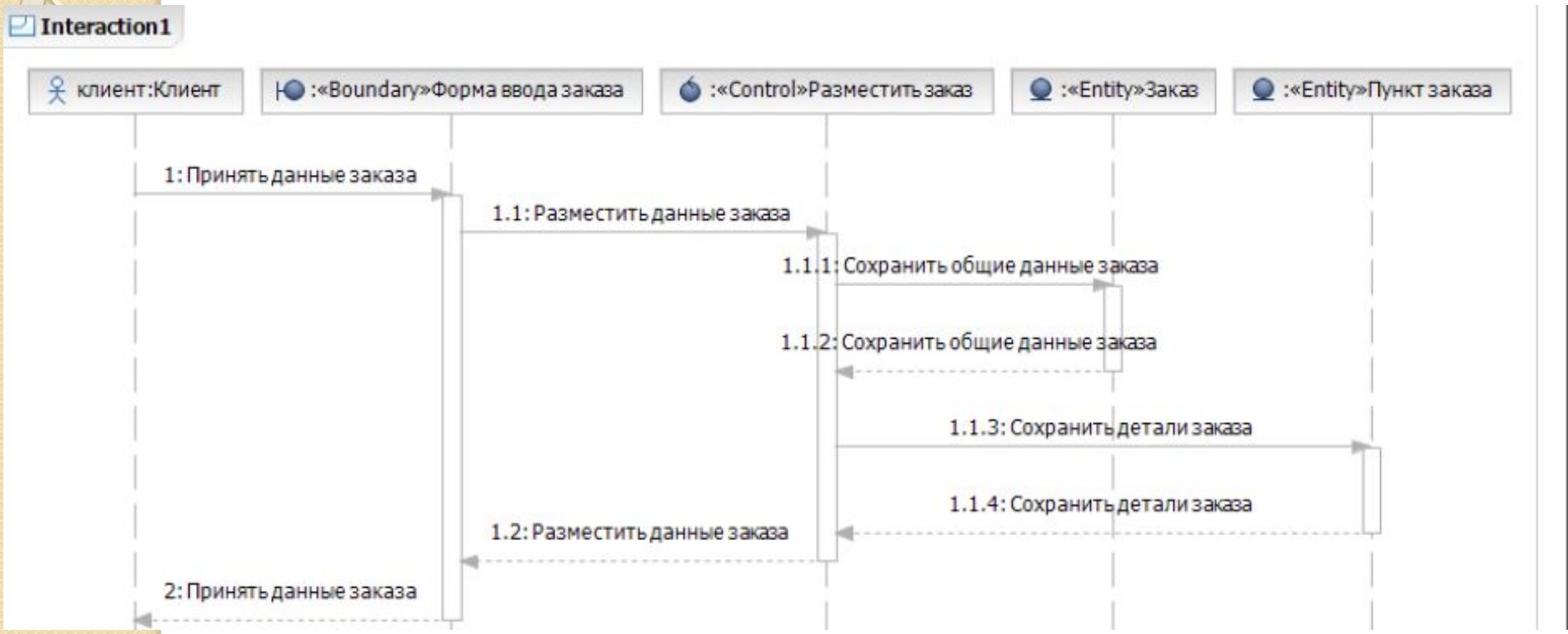
Диаграммы последовательности

- Диаграмма коммуникации
  - Взгляд на взаимодействие объектов с акцентом на структурную организацию взаимодействующих объектов

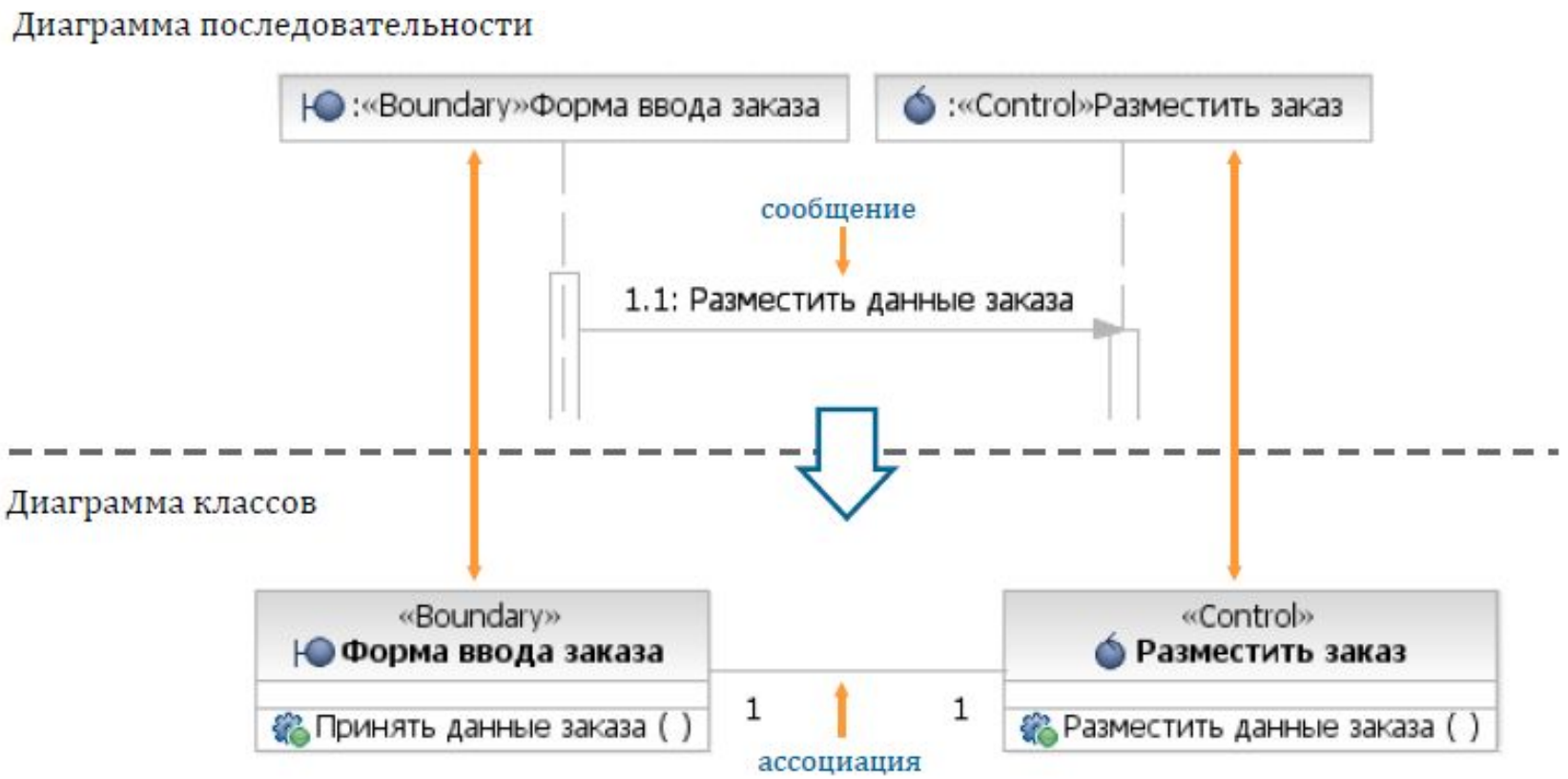


Диаграммы коммуникации

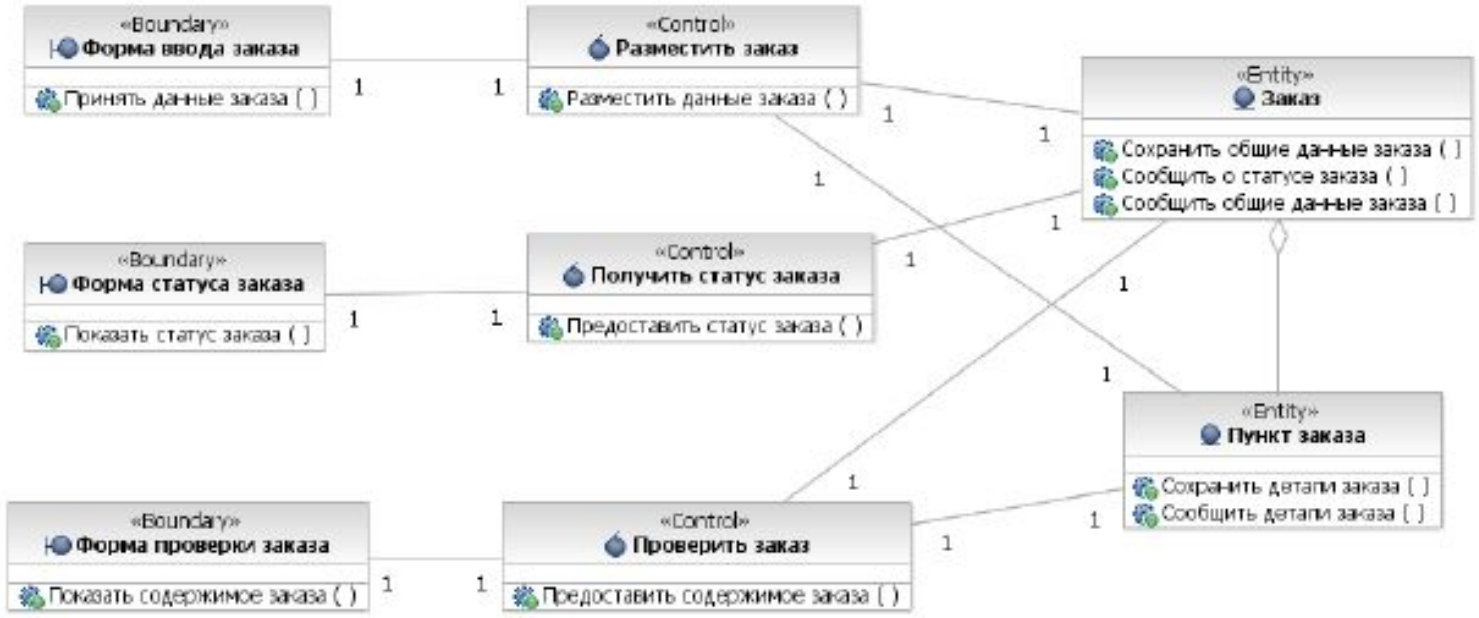
Диаграмма последовательностей (sequence)



Выявление связей между классами – 1 сообщение -> 1 операция



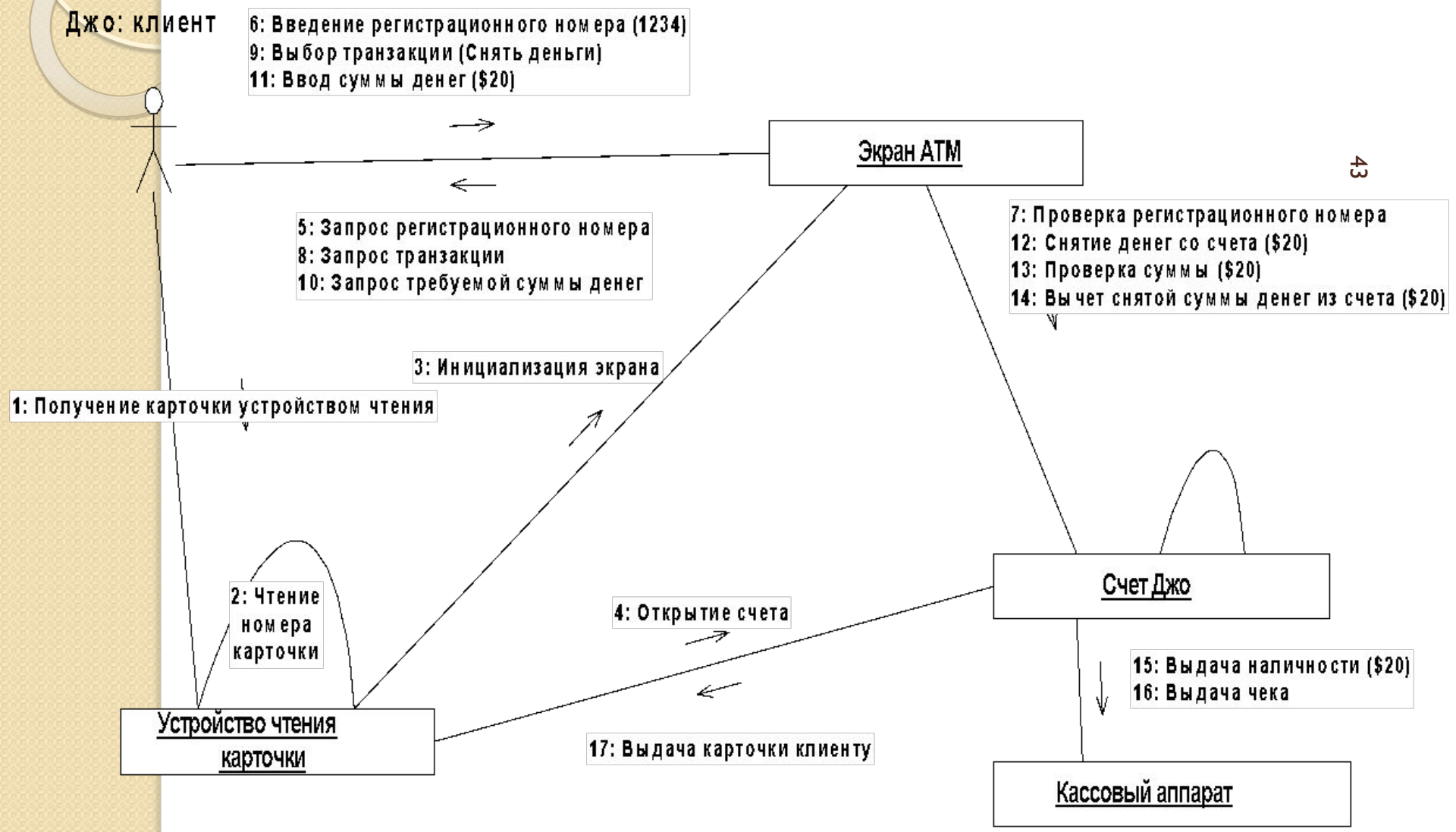
Диаграммы классов (ассоциации классов)



Кооперативные диаграммы (collaboration)

- Подобно диаграммам последовательности, кооперативные диаграммы (collaborations) отображают поток событий через конкретный сценарий варианта использования.
- Диаграммы последовательности упорядочены по времени, а кооперативные диаграммы больше внимание заостряют на связях между объектами.
- У разных разработчиков имеются различные предпочтения по поводу выбора вида диаграммы взаимодействия.
- В диаграмме последовательности делается акцент именно на последовательность сообщений: легче наблюдать порядок, в котором происходят различные события. На кооперативной диаграмме можно использовать пространственное расположение объектов для того, чтобы показать их статическое взаимодействие.

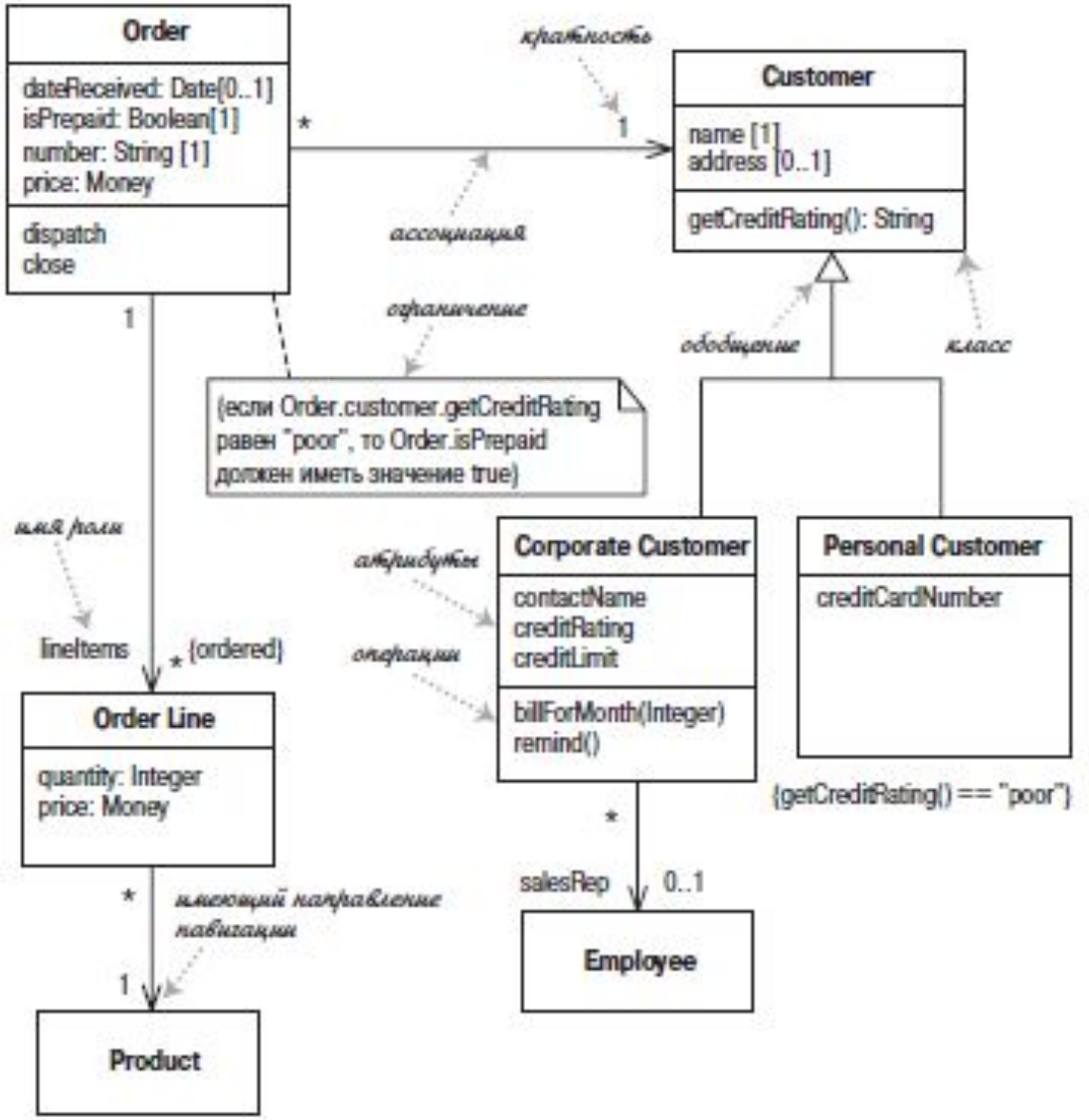
Кооперативная диаграмма (вариант использования «Снять деньги со счета»)



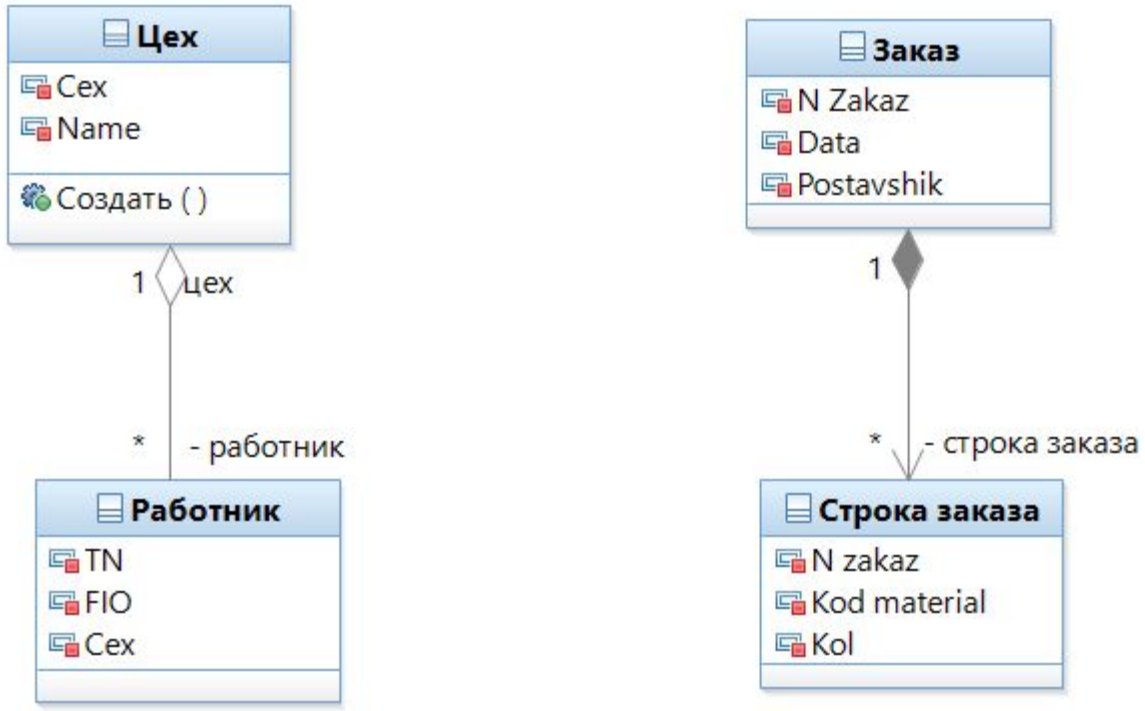
Диаграммы классов (class)

- Структура класса: атрибуты и операции (методы)
- Отношения:
  - Композиция: Заказ 1 <>--- * Пункт заказа
  - Ассоциации: Предприятие 0..1 --->* Клиент
  - Обобщение: Частный клиент --- |> Клиент

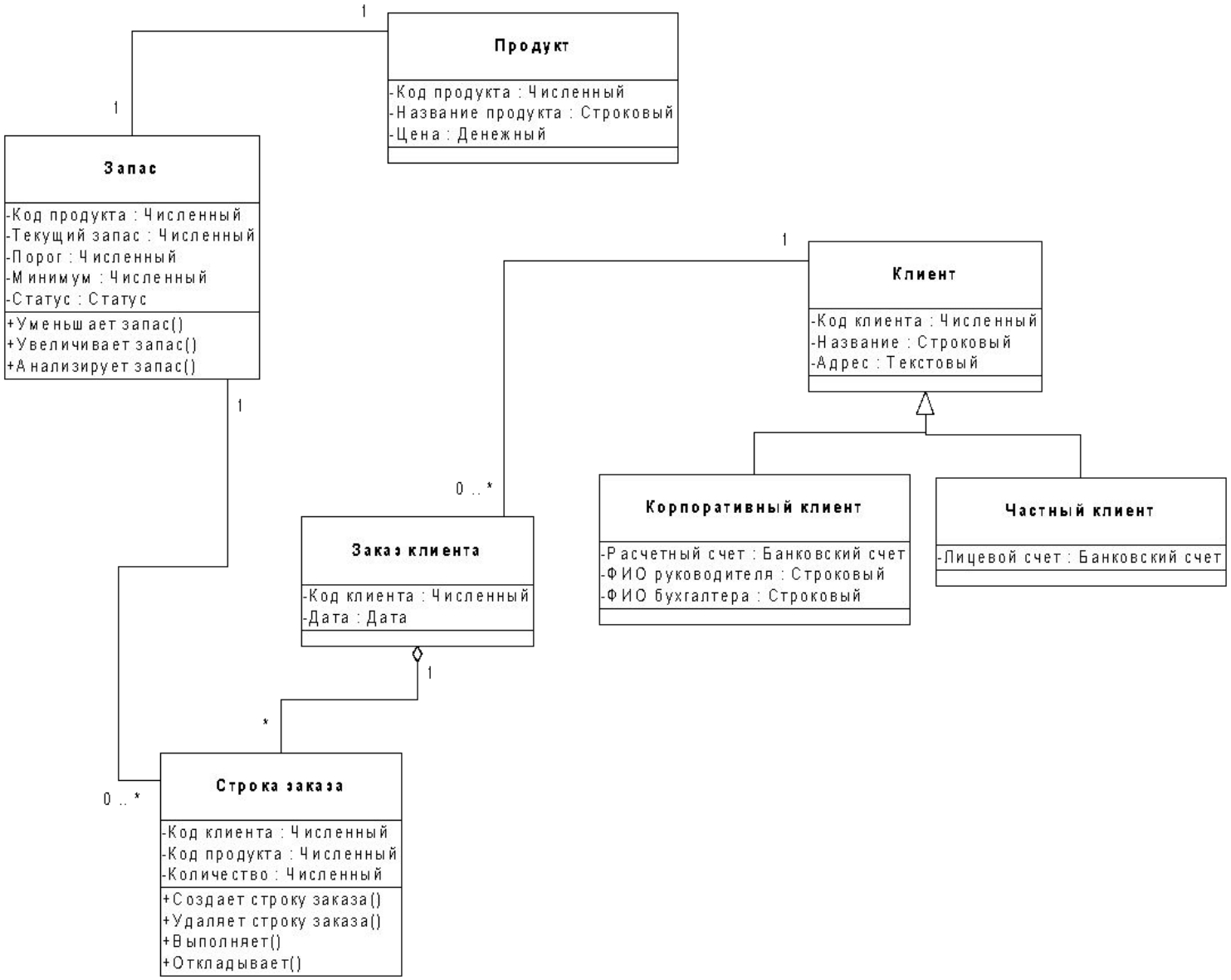
Пример диаграммы классов



Агрегация/композиция



Пример диаграммы классов



Диаграммы классов (ассоциации классов)

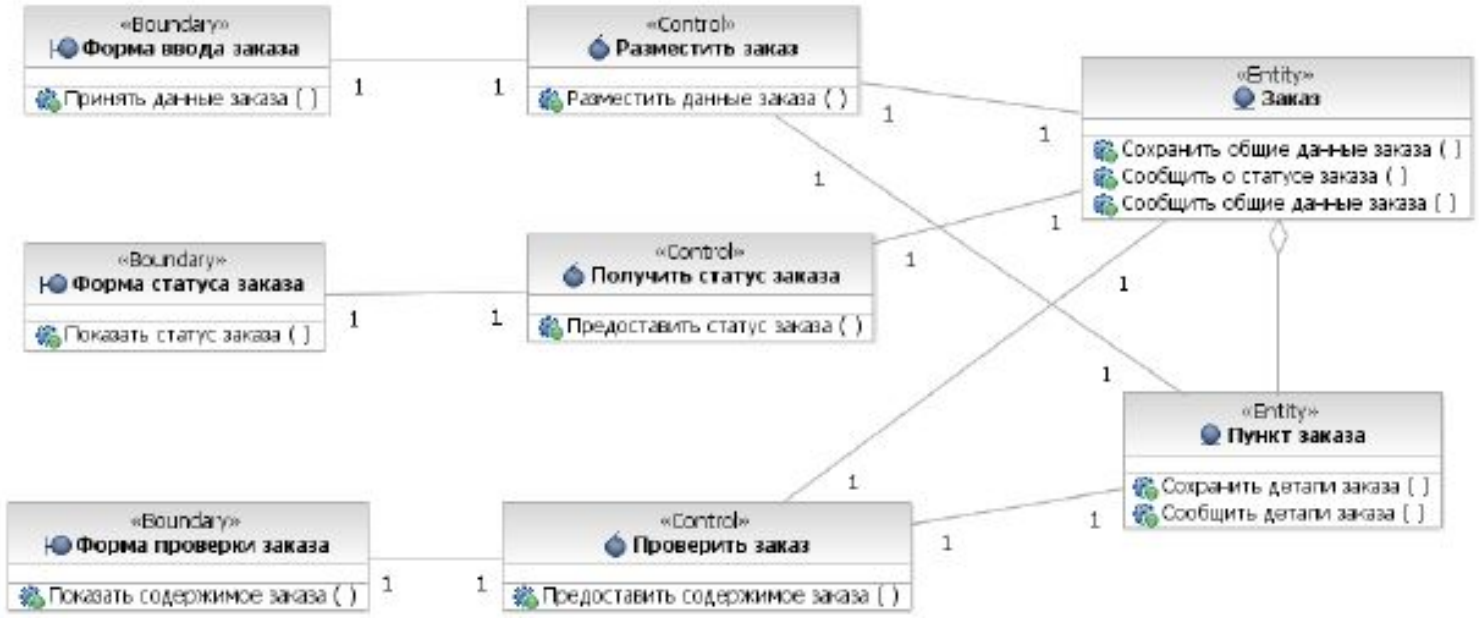
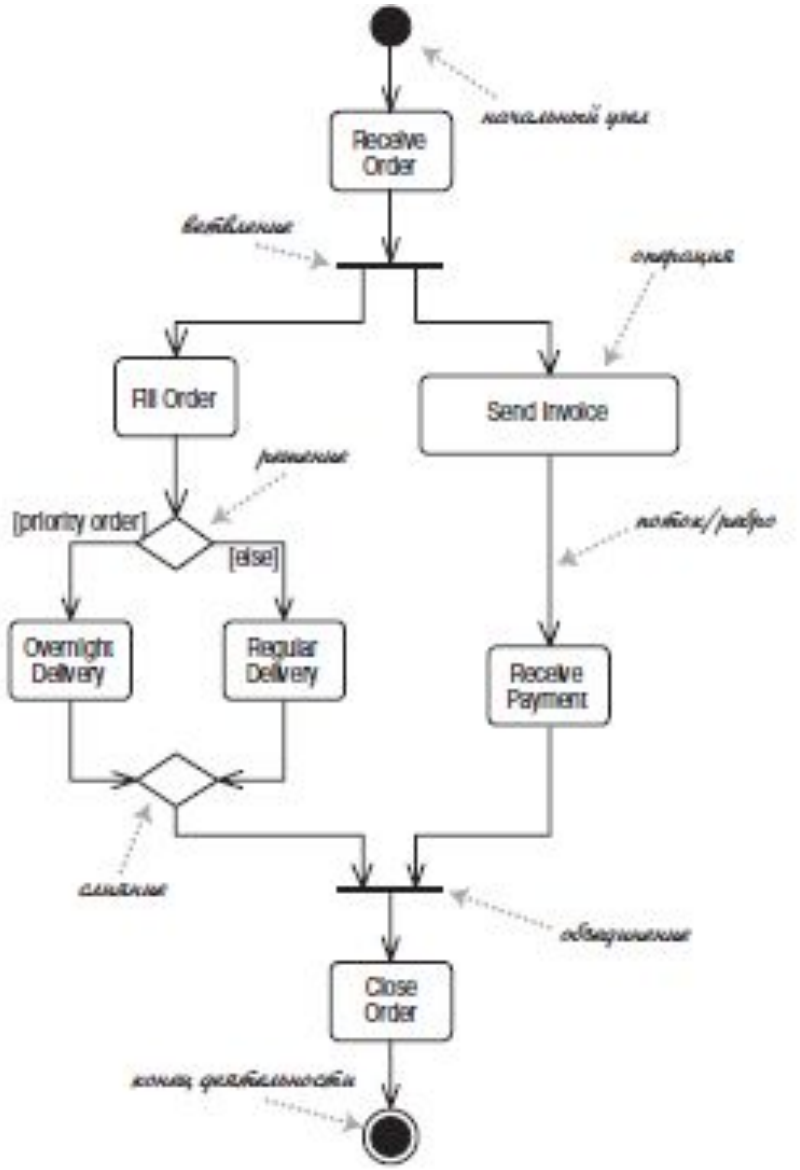


Диаграмма деятельностей (activity – активностей)

- Диаграммы деятельности – определяет технологию исполнения деятельности, описывающую логику процедур, бизнес-процессов и потоков работ.
- Во многих случаях DA напоминают блок-схемы, но принципиальная разница заключается в том, что они поддерживают параллельное исполнение процесса (асинхронное/синхронное).

Пример диаграммы деятельностей



Типы элементов

- Действие – action
- Управляющий поток – control flow
- Принятие решений – Decision (Xor)
- Разделение потока - Fork (And)
- Объединение потоков – Join (And)
- Слияние потоков – Merge (Or)
- Начальная и конечная точка (Initial, End)
- Разбиение деятельности (Partition)

Пример диаграммы активностей

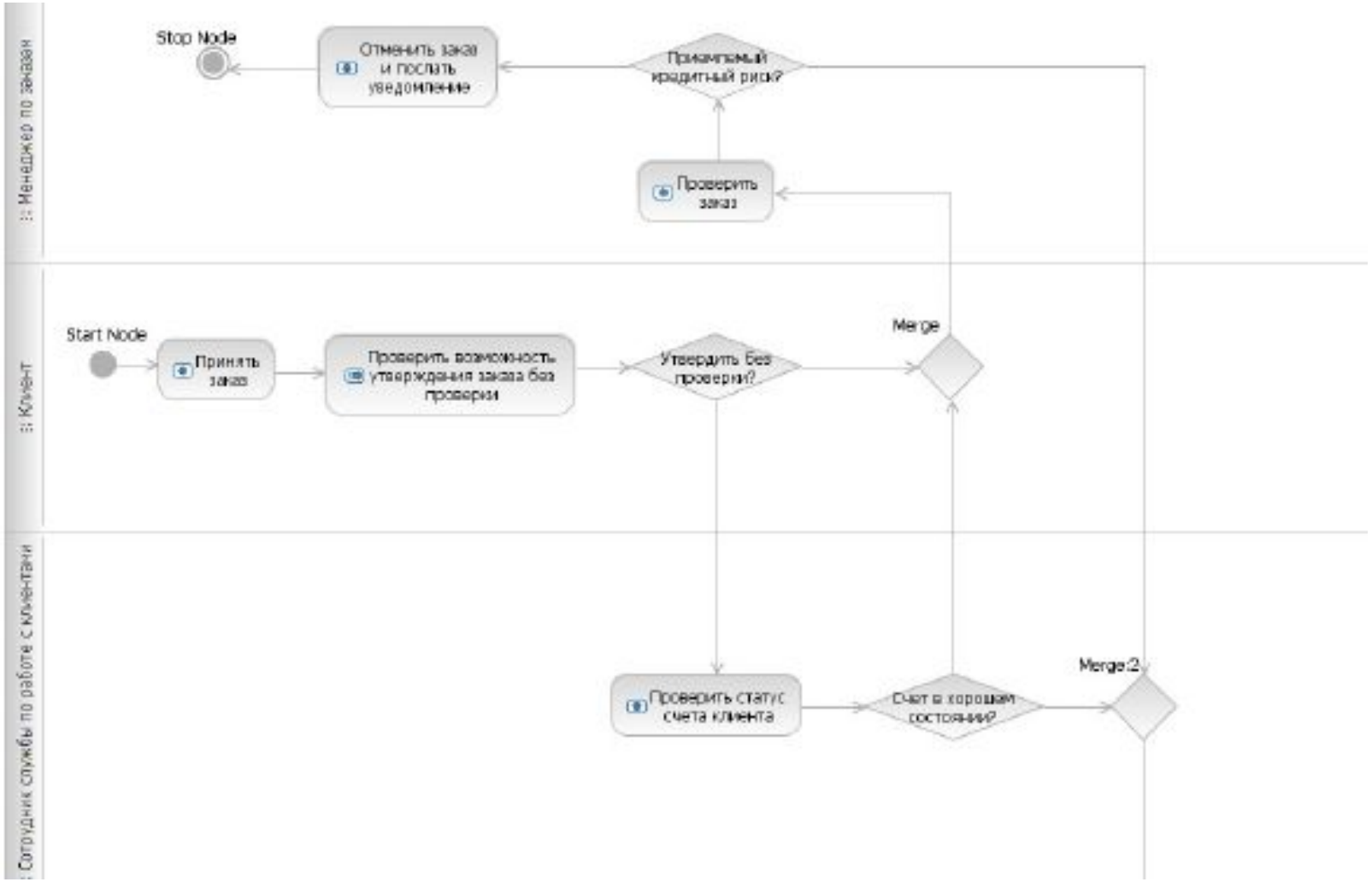


Диаграмма активностей

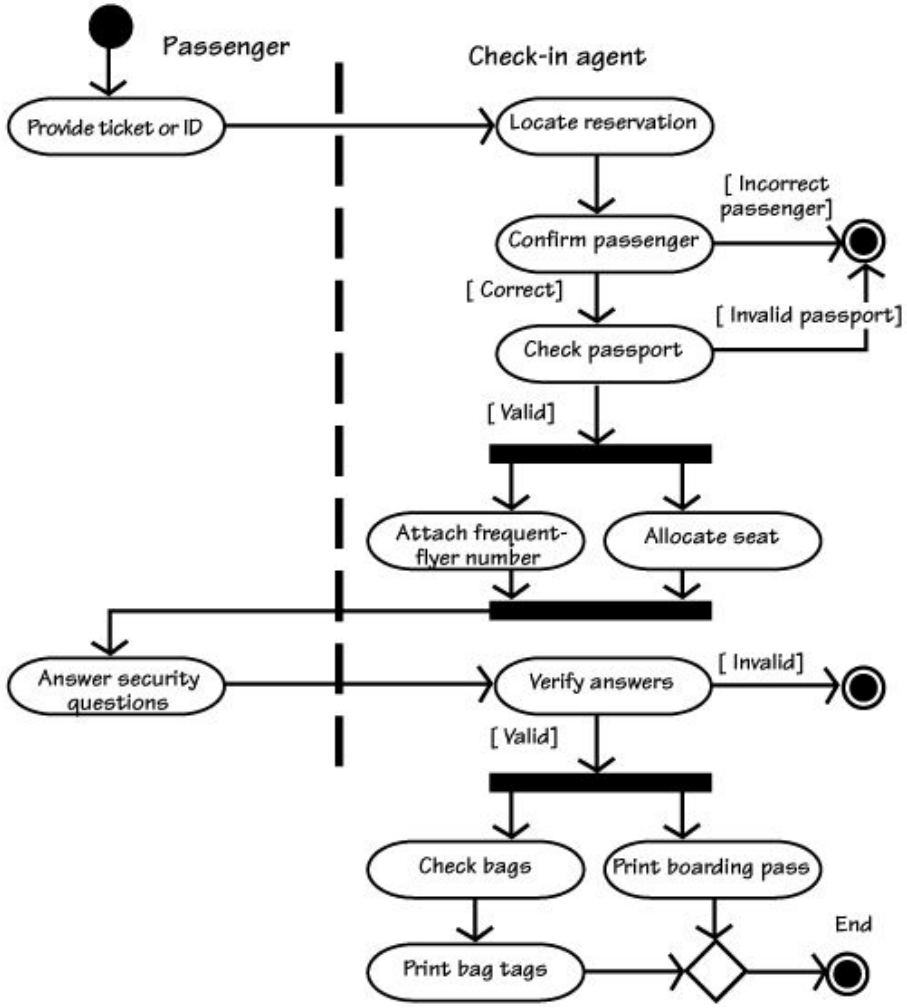


Диаграмма состояний (State chart diagram) или автомата (State Machine diagram)

- ***Диаграмма автомата*** (State Machine diagram, *диаграмма конечного автомата*, *диаграмма состояний*) — диаграмма, на которой представлен конечный автомат с простыми состояниями, переходами и композитными состояниями.
- ***Конечный автомат*** (*State machine*) — спецификация последовательности состояний, через которые проходит объект или взаимодействие (совокупности взаимодействующих объектов) в ответ на события своей жизни, а также ответные действия объекта на эти события. Конечный автомат прикреплён к одному исходному классу и служит для определения поведения его экземпляров

Пример диаграммы состояний

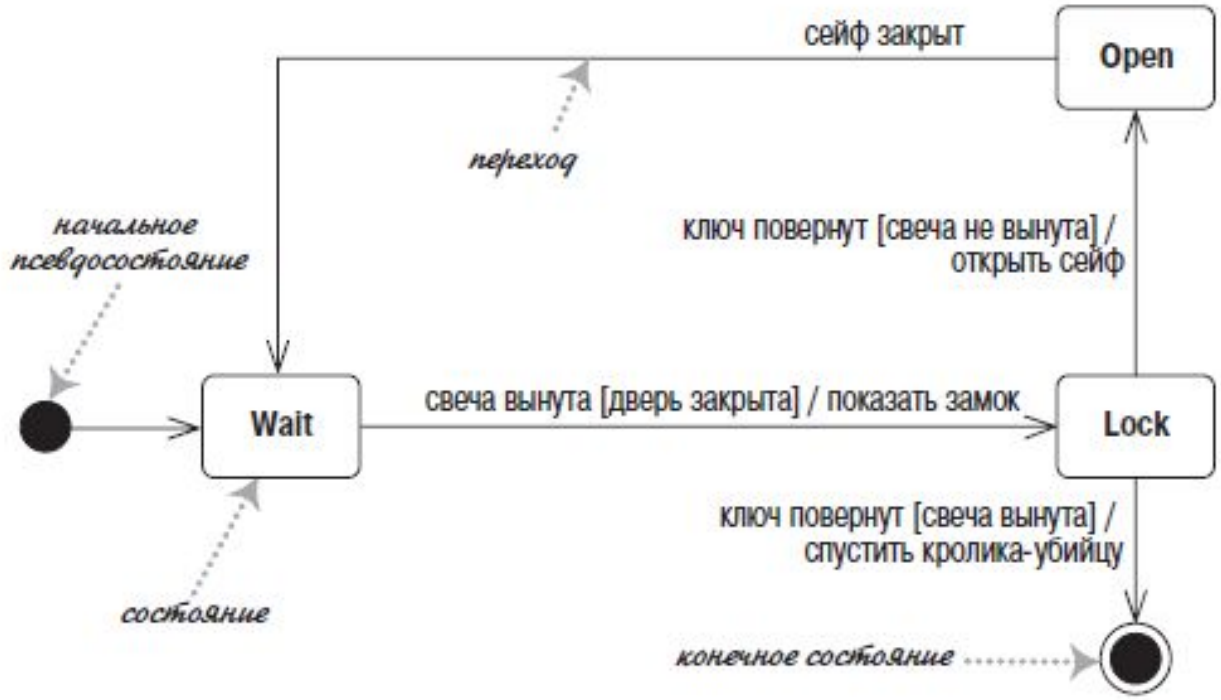


Диаграмма состояний

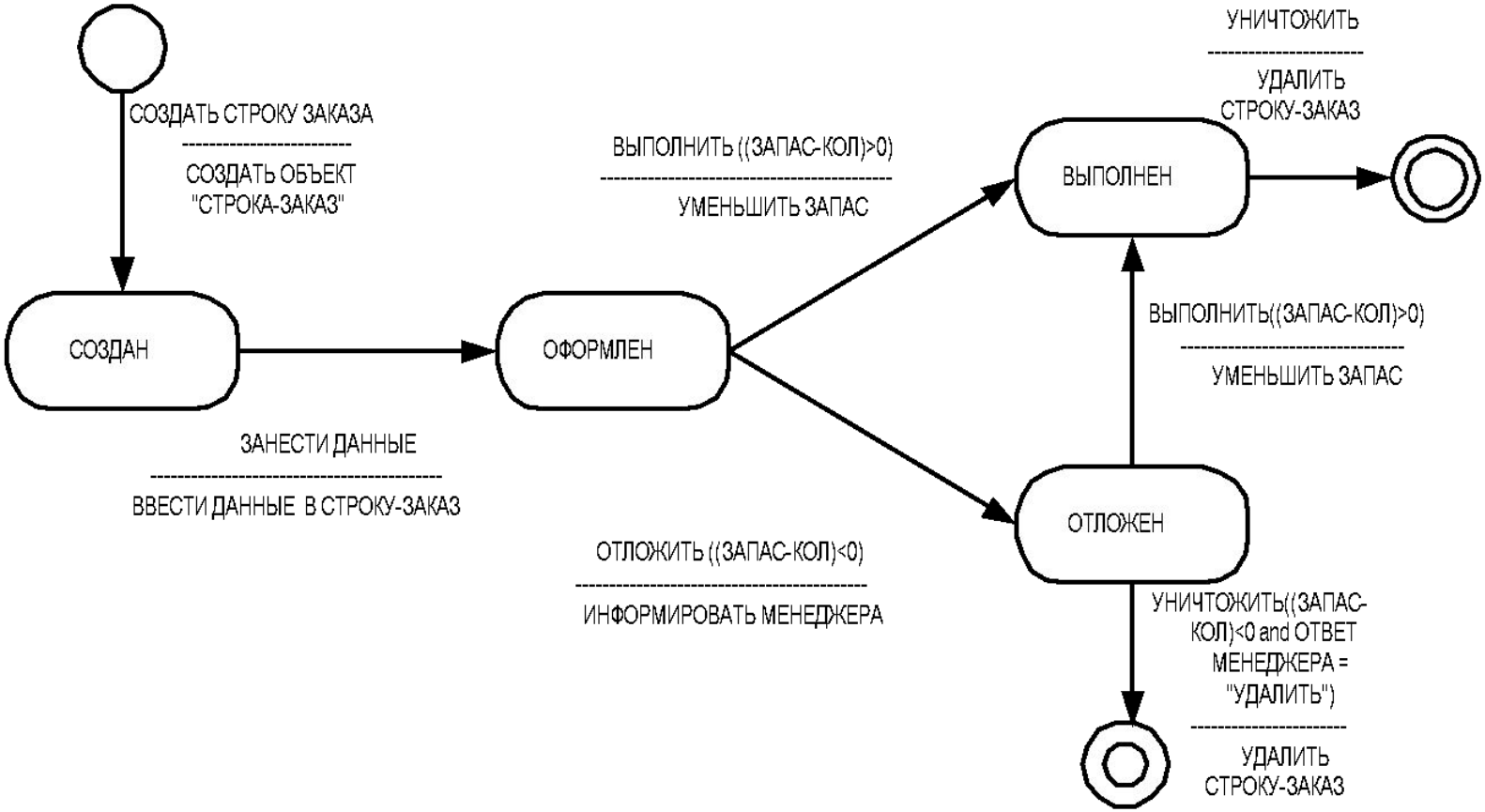


Диаграмма пакетов

Диаграмма пакетов (Package diagram) — структурная диаграмма, основным содержанием которой являются пакеты («контейнеры» диаграмм и элементов) и отношения между ними.

Жёсткого разделения между разными структурными диаграммами не проводится, поэтому данное название предлагается исключительно для удобства и не имеет семантического значения (пакеты и диаграммы пакетов могут присутствовать на других структурных диаграммах).

Пакеты (диаграммы пакетов) служат, в первую очередь, для логического объединения элементов в группы по какому-либо признаку с целью упрощения структуры и организации работы с моделью системы.

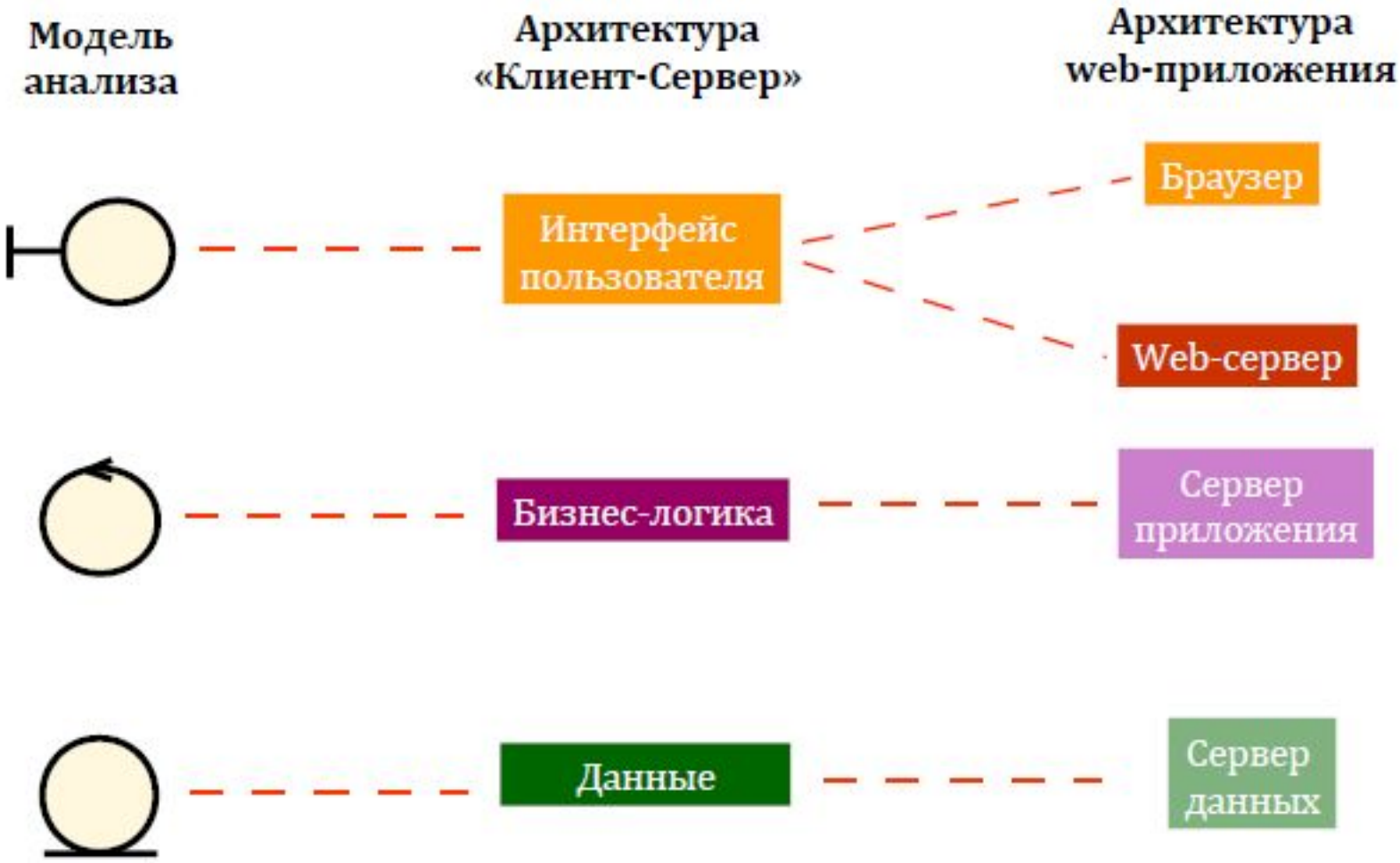
Принципы объединения в пакеты

- Общий принцип замыкания (Common Closure Principle) - причины изменения классов пакета должны быть одинаковые.
- Общий принцип повторного использования (Common Reuse Principle) утверждает, что классы должны использоваться повторно все вместе

Сборка классов объектов в пакеты

- Объединение классов объектов, реализующих интерфейсы одного актора (АРМы)
- По слоям клиент-серверной архитектуры
- Удобство конфигурации для конкретных пользователей

Переход к архитектурным моделям



Пример диаграммы пакетов

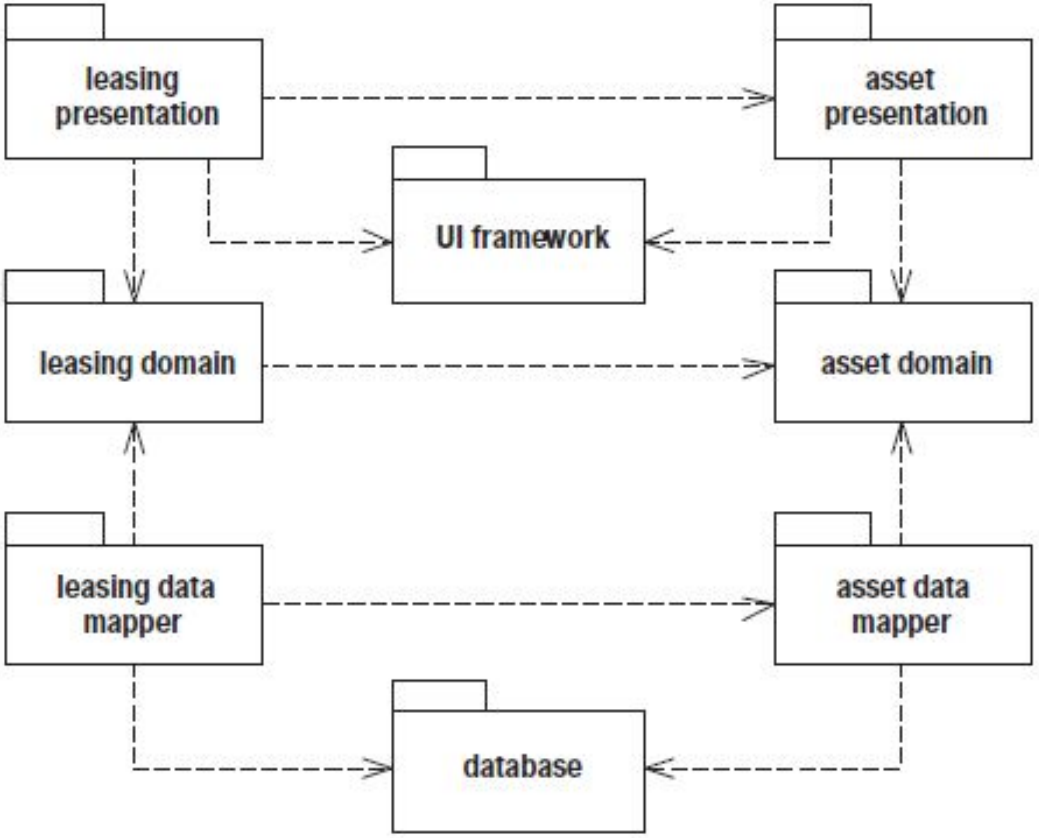
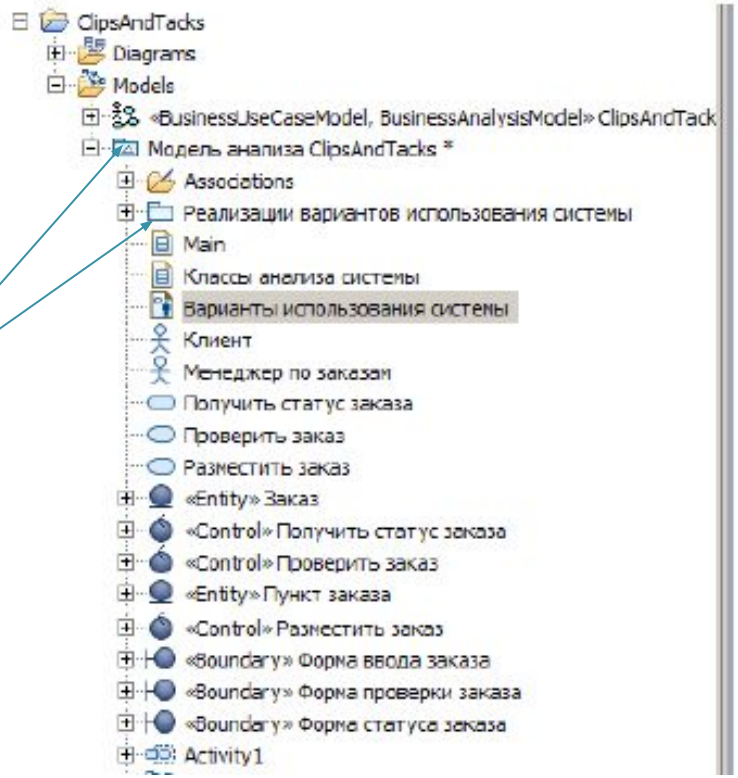
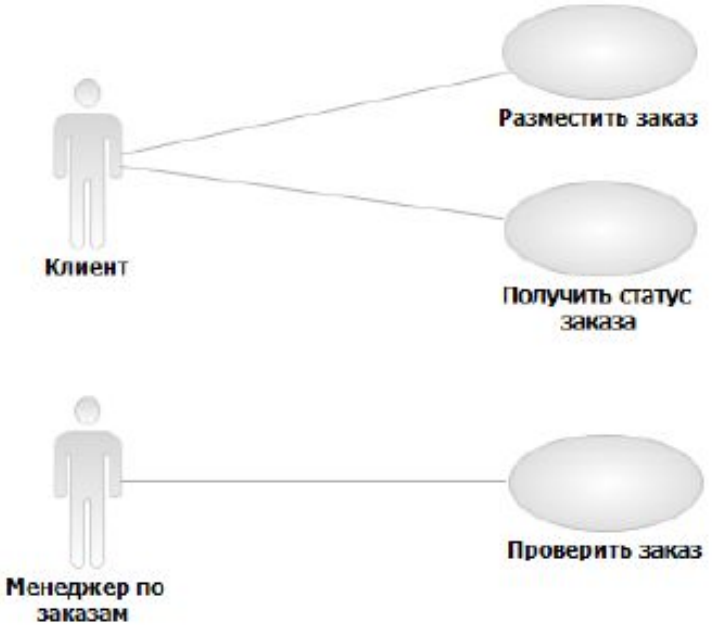


Диаграмма вариантов (прецедентов) использования

Пакеты





Пакеты вариантов использования

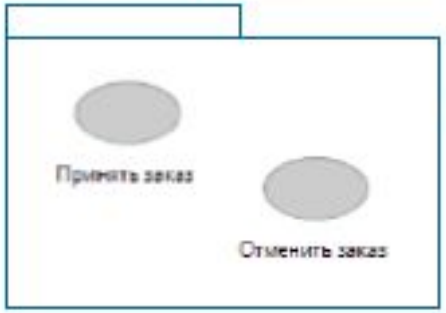
Пакет вариантов использования

- Используется для структурирования модели вариантов использования с целью упрощения анализа, коммуникации, навигации и планирования
- Примеры пакетов:

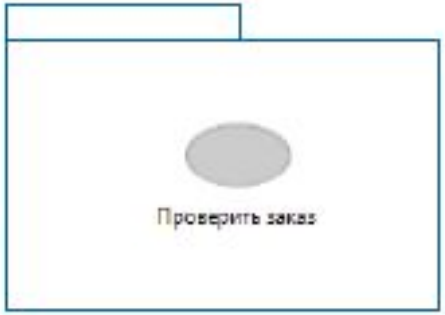
Делим задачу на подзадачи

Варианты использования для задачи обработки заказов

Упрощаем взаимодействие с заинтересованными лицами



Варианты использования для менеджера по заказам



Варианты использования для кредитного специалиста

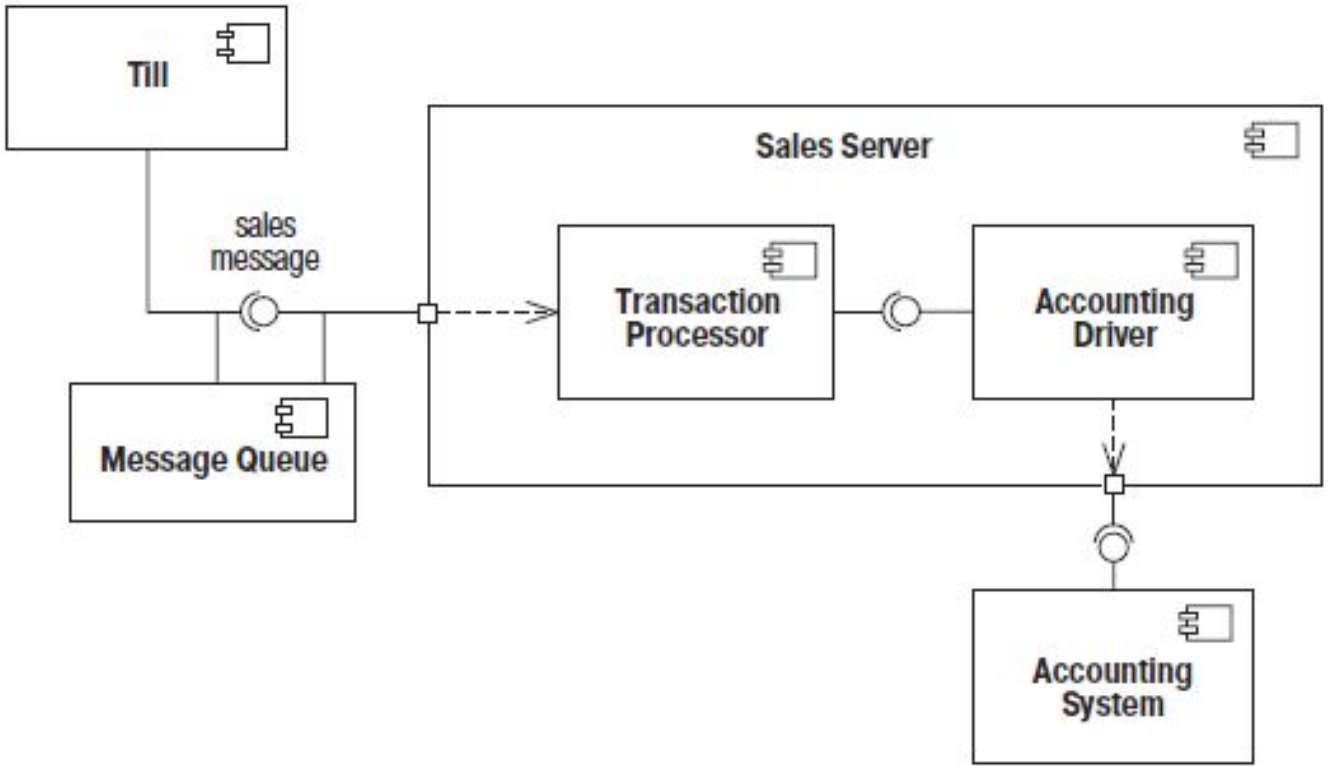
Диаграммы компонентов

- Диаграммы компонентов следует применять, когда система разделяется на компоненты и надо показать их взаимоотношения посредством интерфейсов или схему компонентов в низкоуровневой структуре системы.
- Компонент – это физически заменяемая часть системы, которая соответствует некоторому набору интерфейсов и обеспечивает его реализацию.
- Компонент представляет собой физическую упаковку логических элементов, таких как классы, интерфейсы, кооперации классов. Минимально компонент реализует один класс
- Компоненты имеют расширение: dll, exe, xml, db …
- Компоненты представляют элементы, которые можно независимо друг от друга создать, приобрести, повторно использовать и обновить.
- Компоненты обычно отображаются на диаграммах развертывания с префиксом «component»

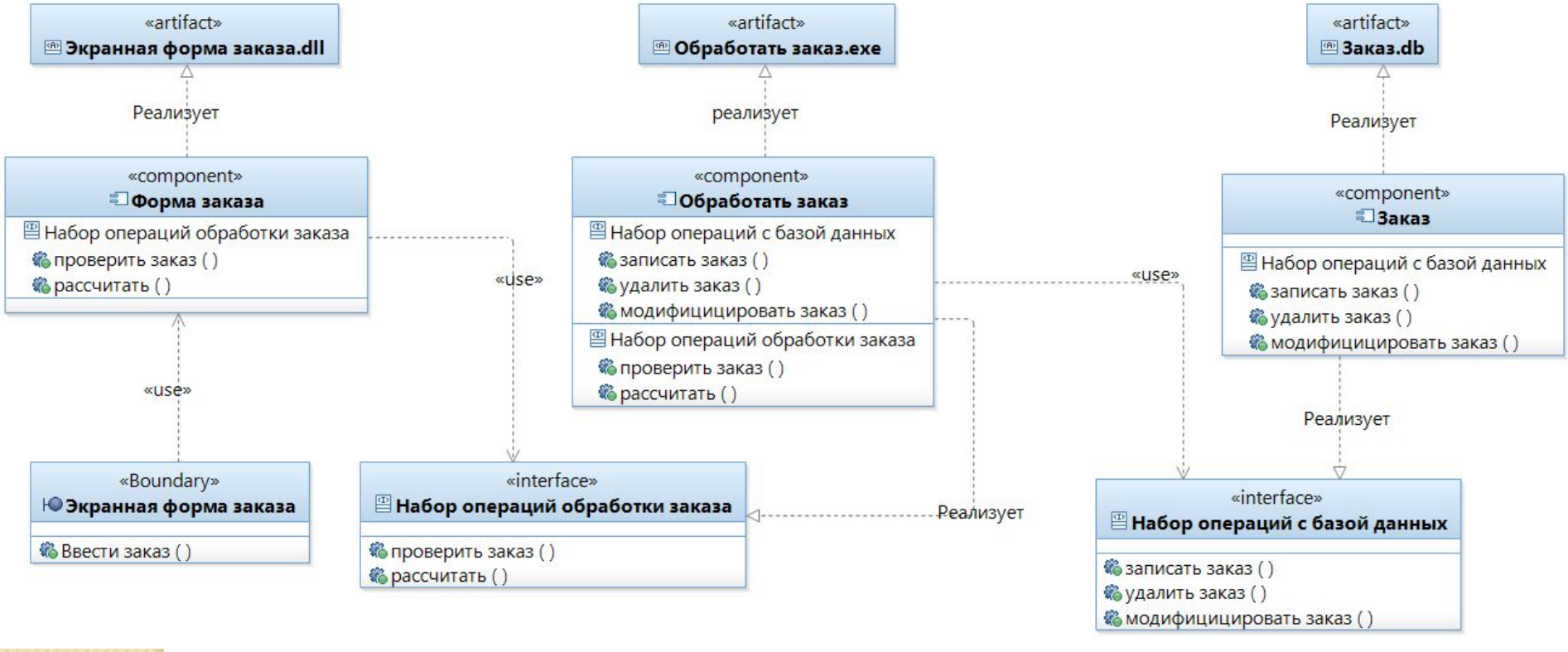
Интерфейсы компонентов

- Компоненты связываются между собой с помощью предоставляемых или требуемых интерфейсов, при этом применяется шарово-гнездовая нотация.
- Интерфейс – это набор операций, специфицирующих услуги (сервисы), предоставляемые компонентом (классом)
- Объявляя интерфейс, получается возможность постулировать желаемое поведение компонента, не зависящее от его реализации

Пример диаграммы компонентов



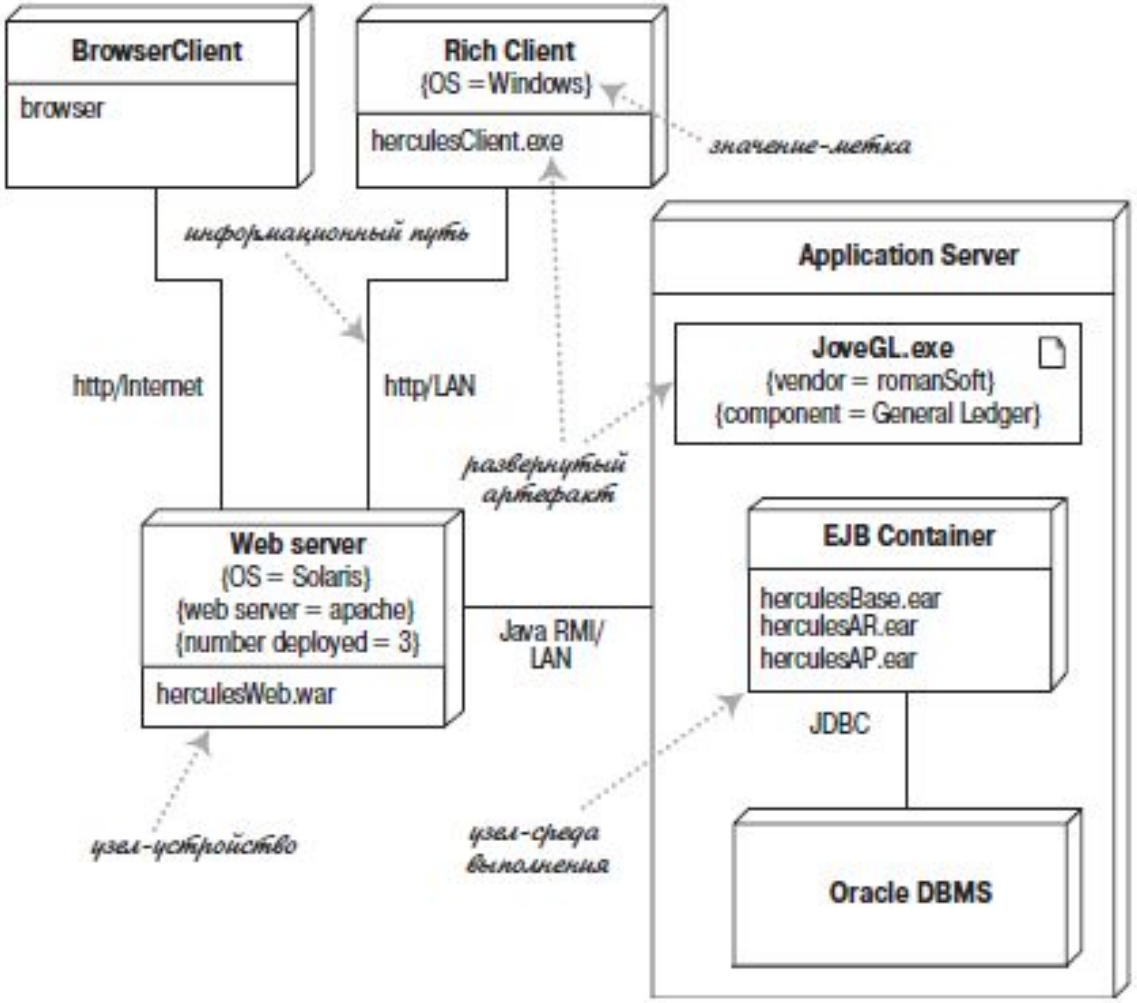
Пример диаграммы коспонентов



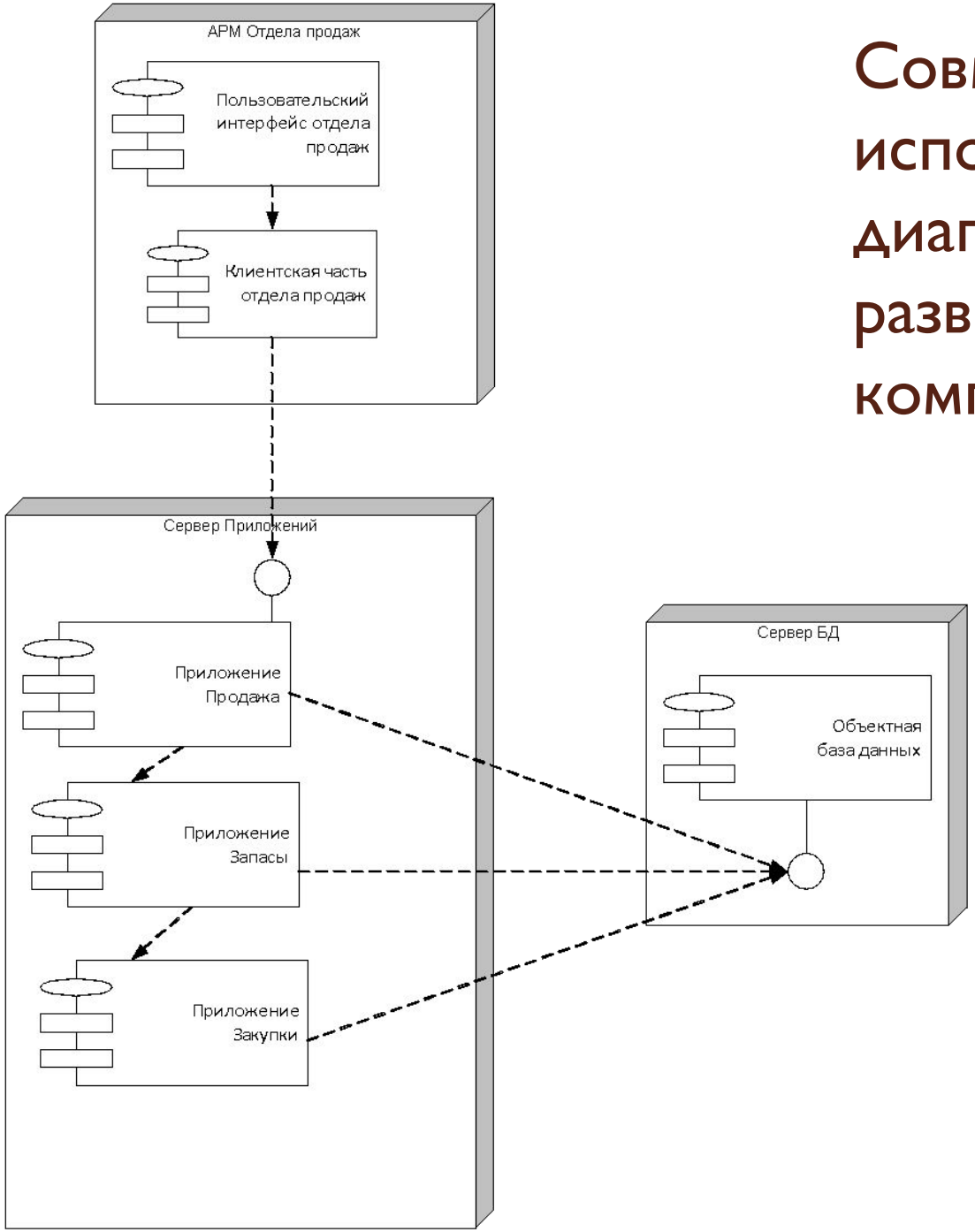
Диаграммы развертывания (Deployment)

- Диаграммы развертывания представляют физическое расположение системы, показывая, на каком физическом оборудовании запускается та или иная составляющая программного обеспечения
- Главными элементами диаграммы являются узлы, связанные информационными путями.
- Узлы бывают двух типов:
  - **Устройство** (device) – это физическое оборудование: компьютер или устройство, связанное с системой.
  - **Среда выполнения** (execution environment) – это программное обеспечение, которое само может включать другое программное обеспечение, например операционную систему или процесс-контейнер
- Узлы могут содержать **артефакты** (artifacts), обычно это исполняемые файлы (такие как файлы *.exe*, двоичные файлы, файлы DLL, файлы JAR, сборки или сценарии) или файлы данных, конфигурационные файлы, HTML документы и т. д.
- Артефакты часто являются реализацией компонентов, что можно показать, задав значения метки внутри прямоугольников артефактов.
- Информационные пути между узлами представляют обмен информацией в системе. Можно сопровождать эти пути информацией об используемых информационных протоколах

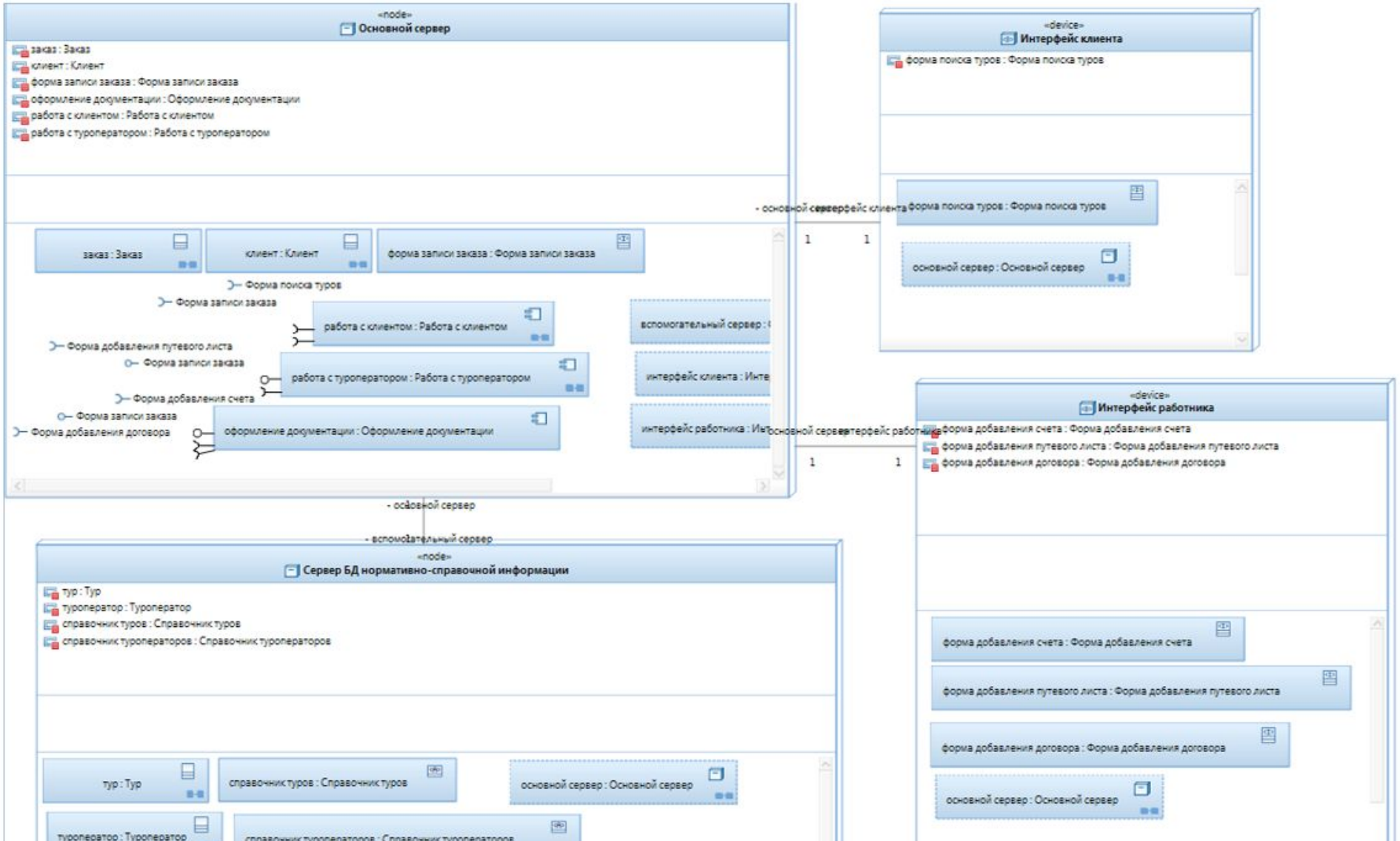
Пример диаграммы развертывания



Совместное использование диаграмм развертывания и компонентов



Пример диаграммы развертывания в RSA



Источники

- М. Фаулер. UML основы, третье издание. - СПб: Символ-Плюс, 2009. – 192 с.
- Смирнова Г.Н., Сорокин А.А., Тельнов Ю.Ф. Проектирование экономических информационных систем. – М.: Финансы и статистика, 2002.
- П. Кролл. Rational Unified Process – это легко. – М.: Кудиц-образ, 2004