# Унифицированный процесс разработки ПО

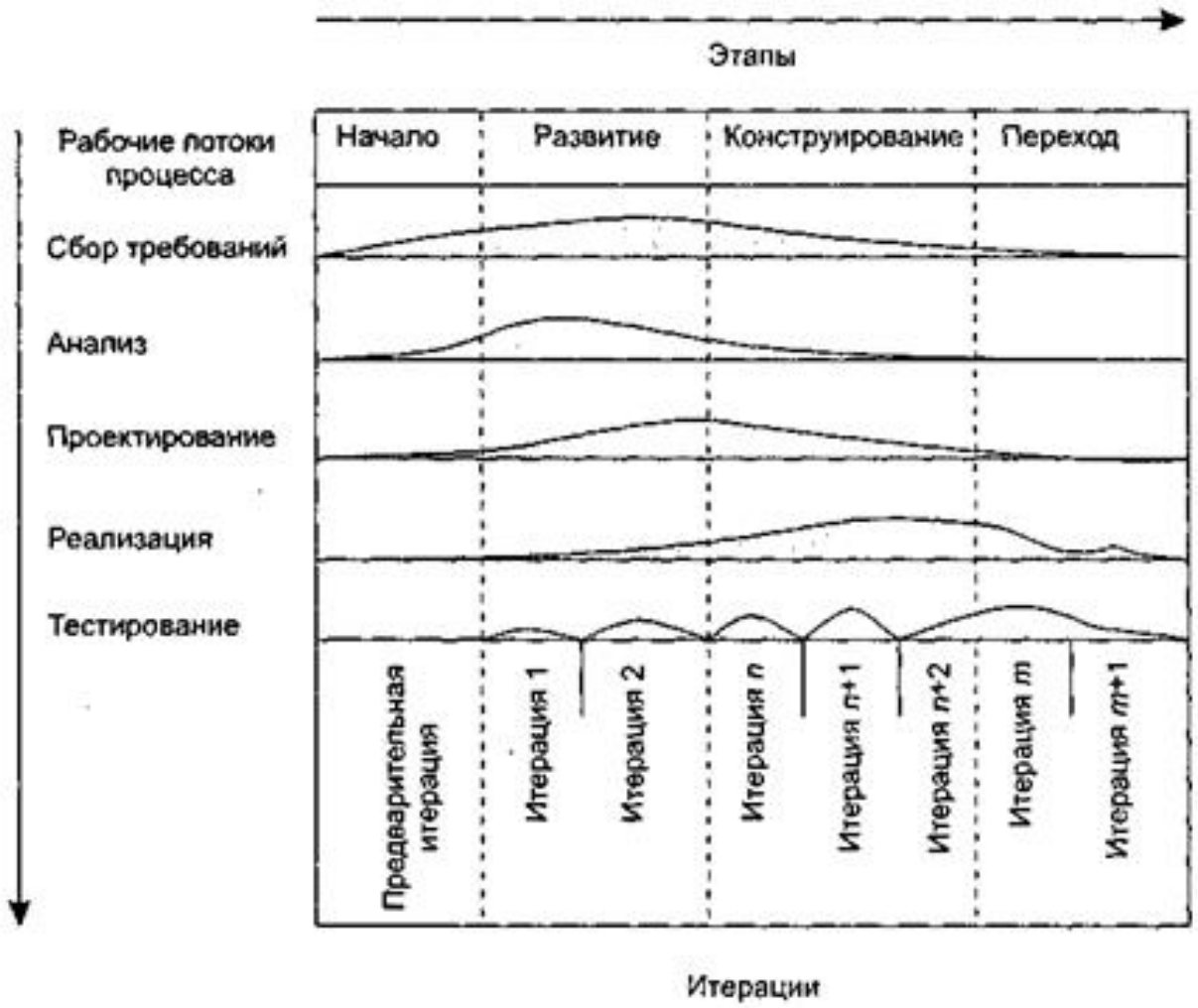
Технологии разработки программного обеспечения

Виноградова М.В.
МГТУ им. Н.Э. Баумана
Кафедра СОИУ (ИУ5)

# RUP - Rational Unified Process

- Использует UML (визуальное проектирование);
- Поддерживается Rational / IBM;
- Управляется прецедентами
- Ориентирован на архитектуру
- Итеративность и инкрементность

# График RUP


Этапы
Рабочие потоки процесса
Начало
Развитие
Конструирование
Переход
Сбор требований
Анализ
Проектирование
Реализация
Тестирование
Предварительная итерация
Итерация 1
Итерация 2
Итерация n
Итерация n+1
Итерация n+2
Итерация m
Итерация m+1
Итерации

# Этапы RUP

- **Начало** - спецификация представления продукта;
- **Развитие** - планирование действий и требуемых решений;
- **Конструирование** - построение ПО в виде серии инкрементных итераций;
- **Переход** - внедрение ПО в среду пользователя (промышленное производство, доставка и применение)

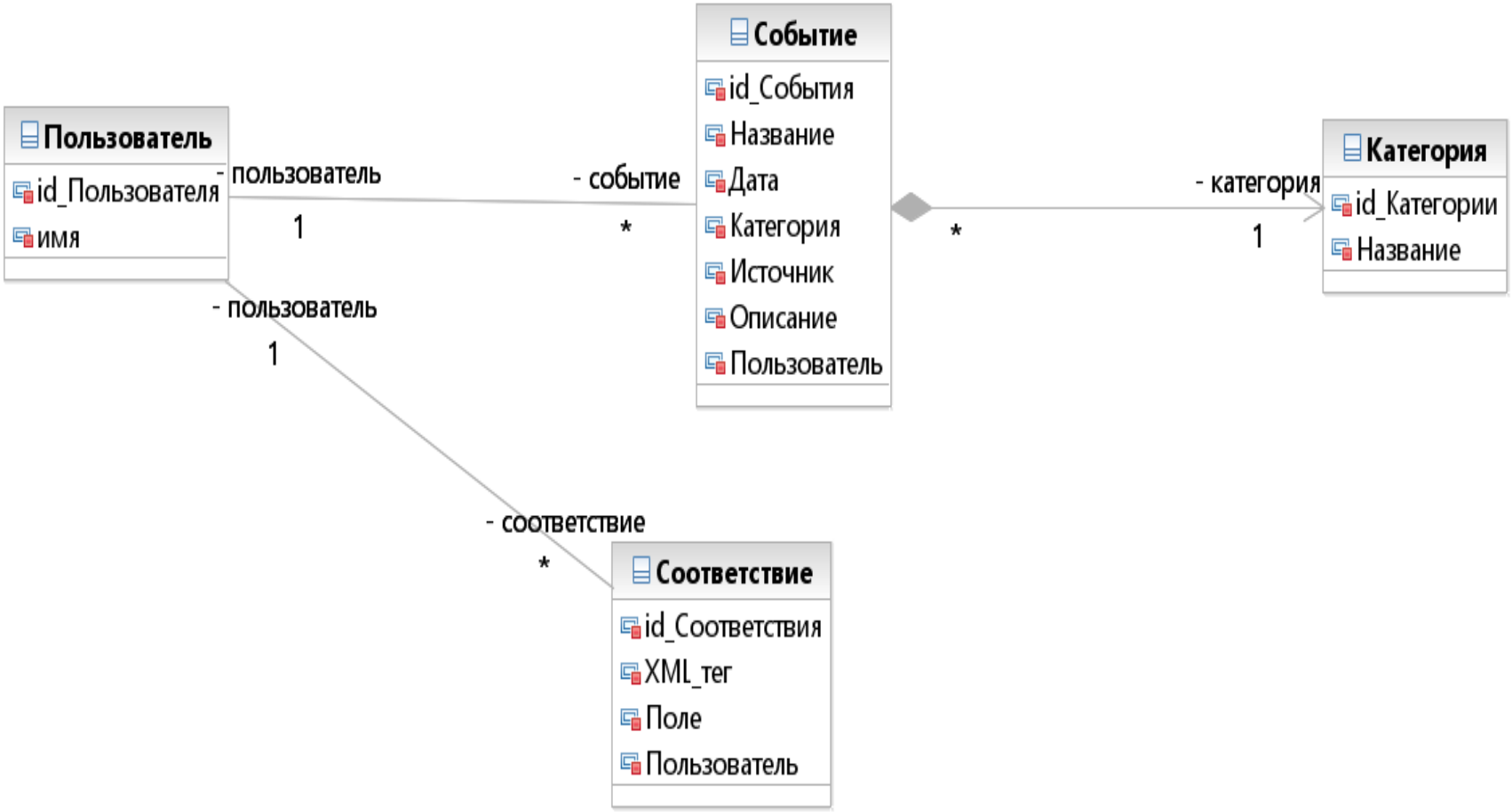
# Рабочие процессы RUP

- Сбор требований - что делает система;
- Анализ - преобразование требований в классы и объекты предметной области;
- Проектирование - создание статического и динамического представления системы для выполнения требований;
- Реализация - производство программного кода;
- Тестирование - проверка системы в целом.

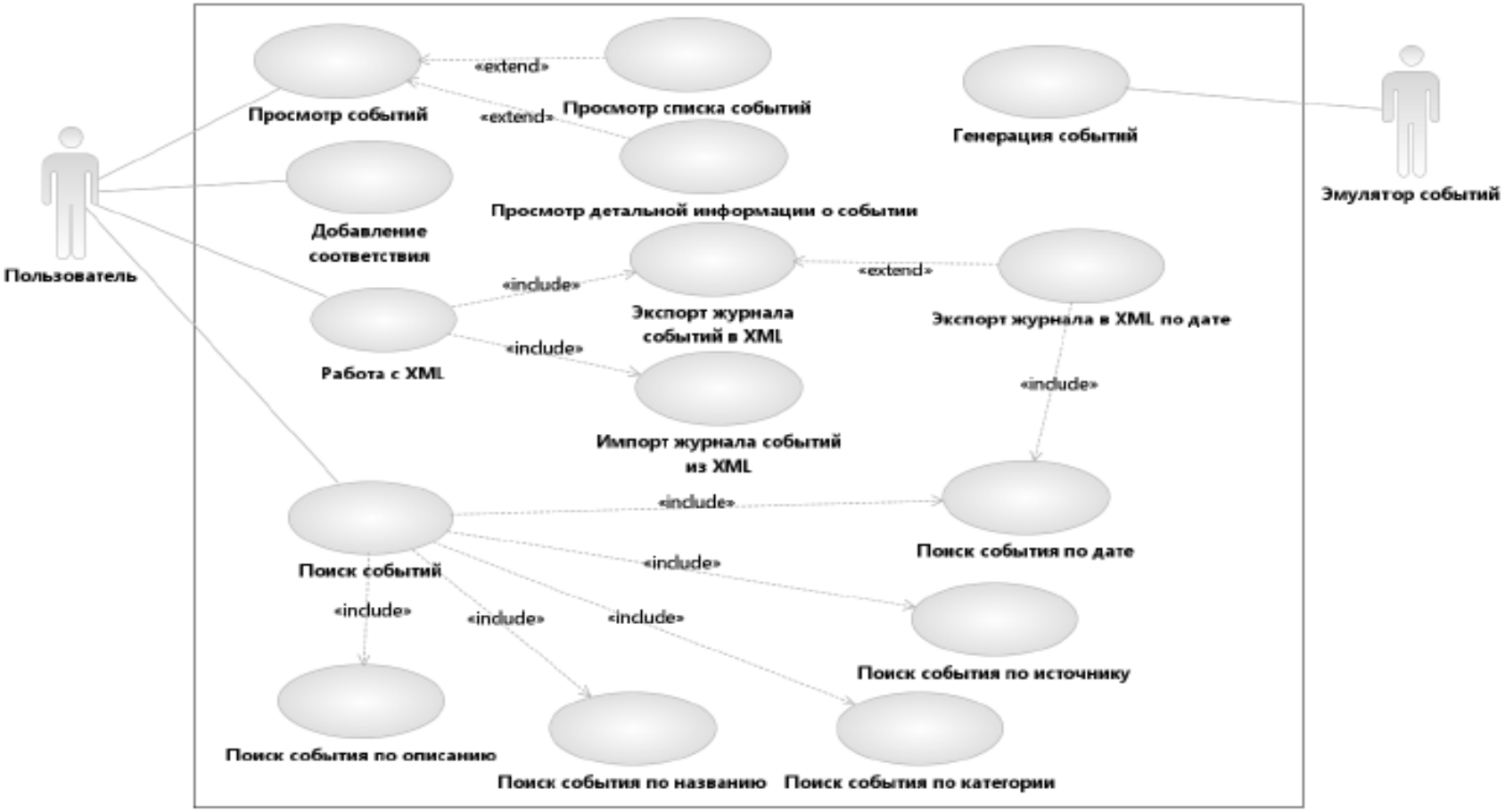
# Рабочий процесс Определение требований

- Перечисление кандидатов в требования
- Осознание контекста системы
- Определение функциональных требований (в виде прецедентов)
- Определение нефункциональных требований

# Модель предметной области – пример ЖСС



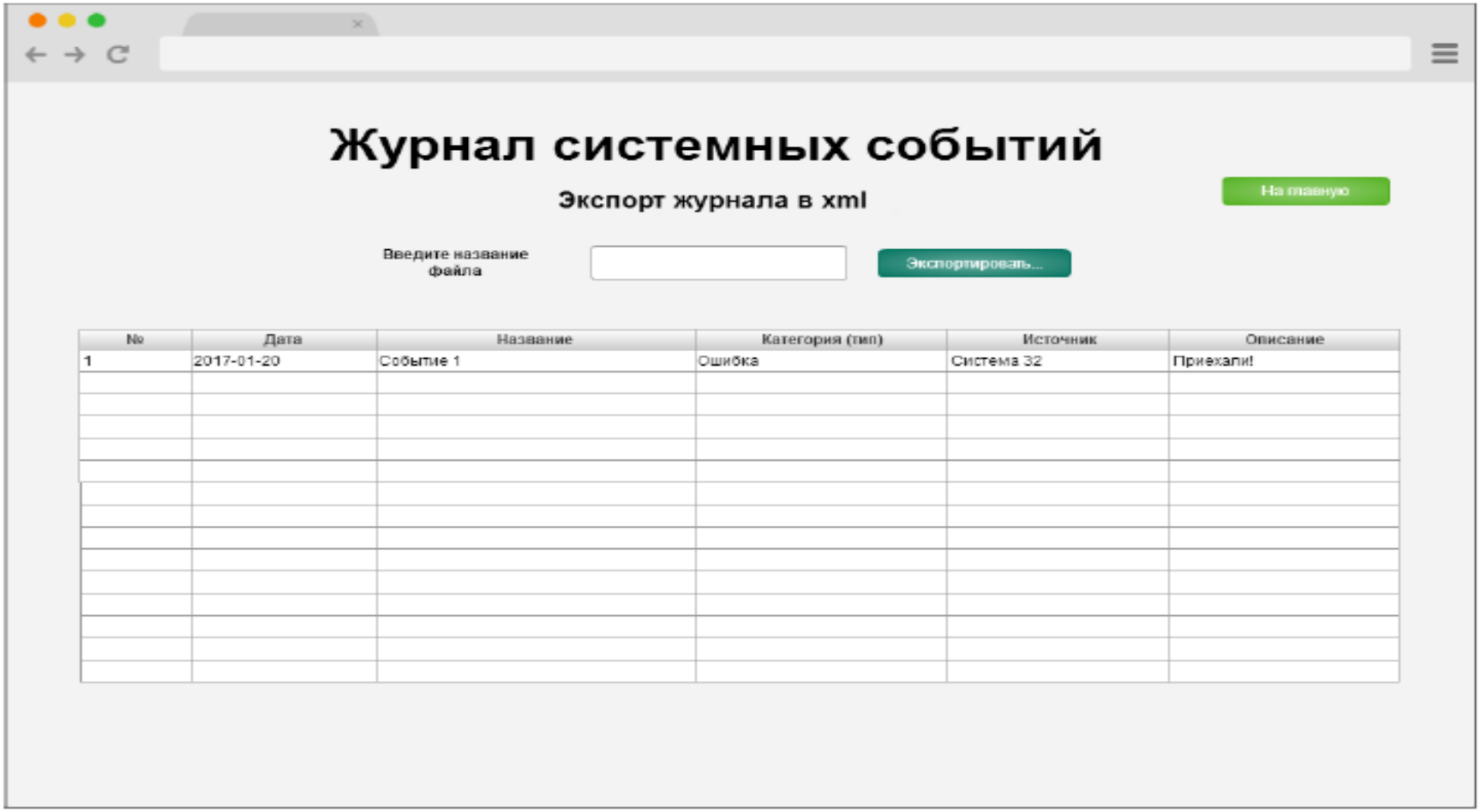
# Модель прецедентов – пример ЖСС



# Спецификация прецедента – пример ЖСС

| |
|---|
| Краткое описание:<br>Предназначен для экспорта журнала событий в документ с расширением XML |
| Главные актеры:<br>Пользователь |
| Второстепенные актеры: Нет |
| Предусловие:<br>1. Выполнен прецедент Работа с XML |
| Основной поток:<br>1. Прецедент начинается после выбора пользователем<br>2. Система проверяет введенные параметры пользователем<br>3. Система формирует пользователю документ в формате XML с информацией о событиях из БД<br>4. Система загружает документ в формате XML пользователю<br>5. Система отображает документ в формате XML пользователю |
| Постусловие: Нет |
| Альтернативные потоки:<br>1. Прецедент начинается после выбора пользователем<br>2. Система проверяет введенные параметры пользователем<br>3. Система выдает ошибку пользователю |
| Прецедент: Импорт журнала событий из XML |

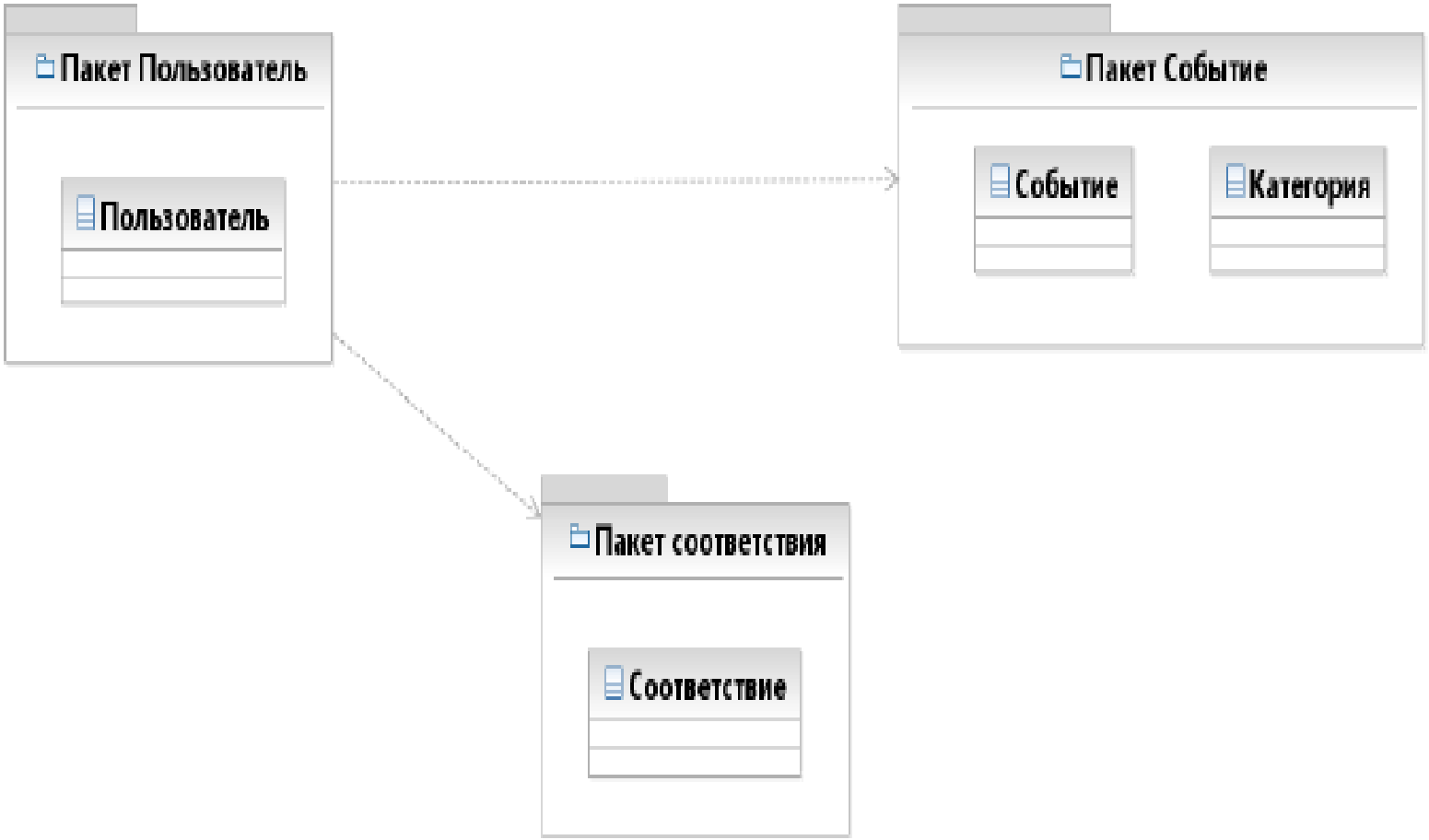
# Прототип ПИ – пример ЖСС



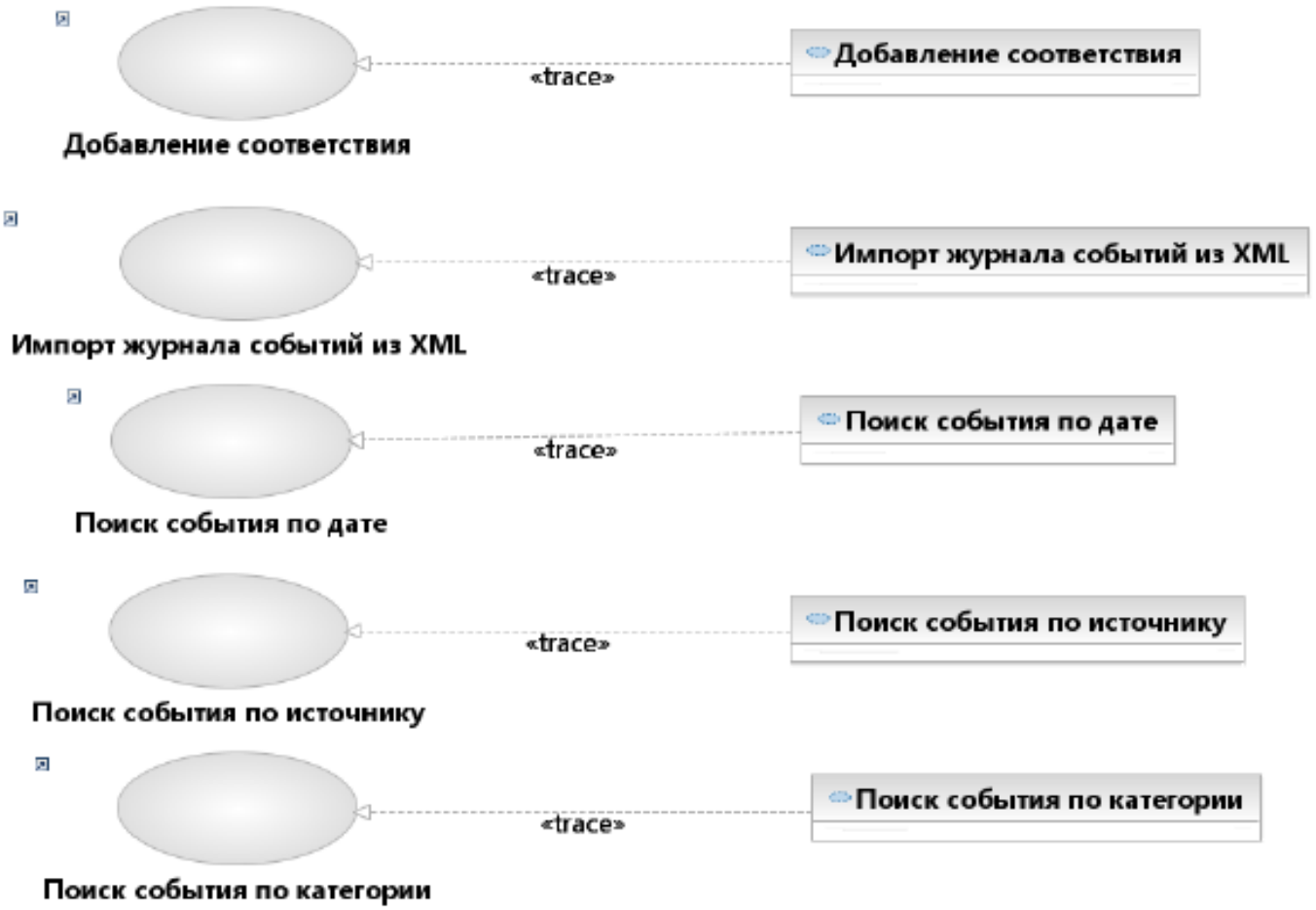
# Рабочий процесс Анализ (требований)

- Анализ архитектуры
- Анализ кооперацій
- Анализ классов
- Анализ пакетов

# Пакеты анализа – пример ЖСС

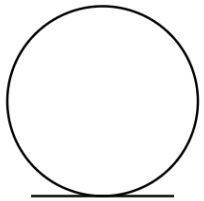


# Трассировка кооперацій от прецедентов – пример ЖСС

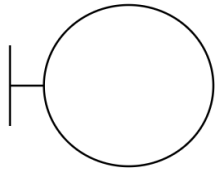


# Классы анализа

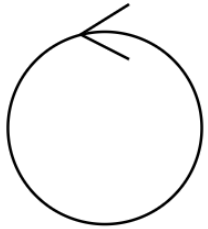
- Класс сущности - для моделирования долгоживущей системы, часто сохраняемой информацию о человеке / объекте / событии реального мира. Может иметь сложное поведение. Показывает логическую структуру данных.



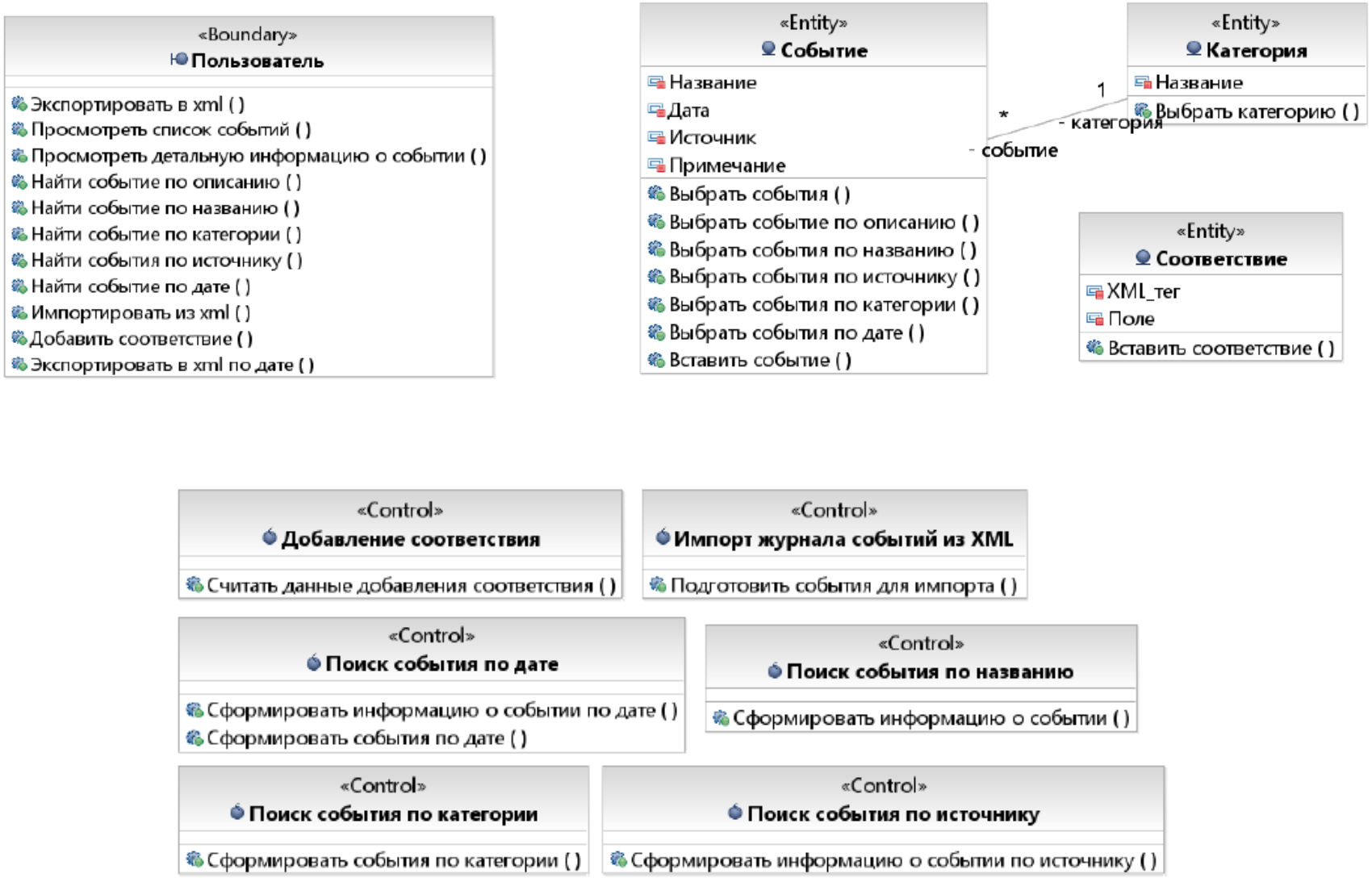
- Граничный класс - для моделирования взаимодействия между системами и актерами:получение / передача информации / запросов. Соответствуют пользовательскому интерфейсу / устройству/ коммуникации / API на внешнем уровне без реализации.



- Управляющий класс - координация / последовательность /взаимодействие / управление другими объектами для прецедента. Обрабатывают и координируют  действия и потоки управления; реализуют бизнес-логику.

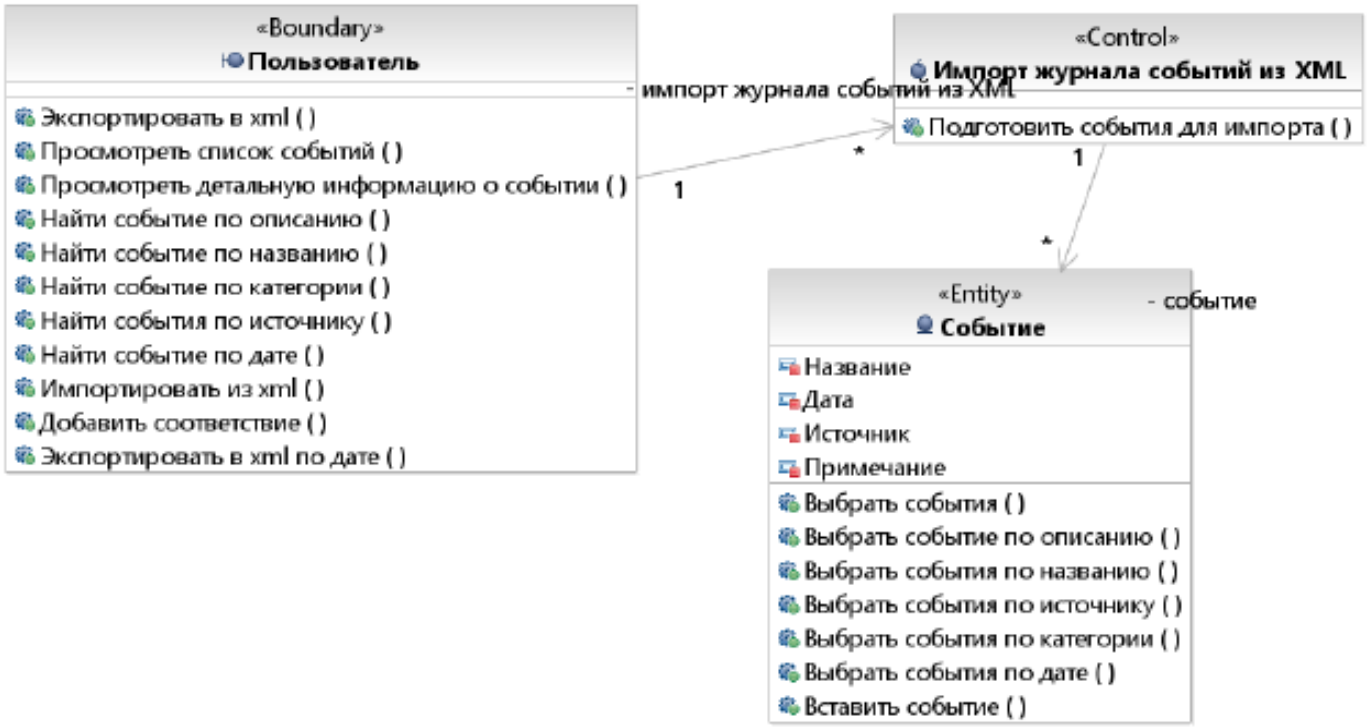


# Классы анализа – пример ЖСС

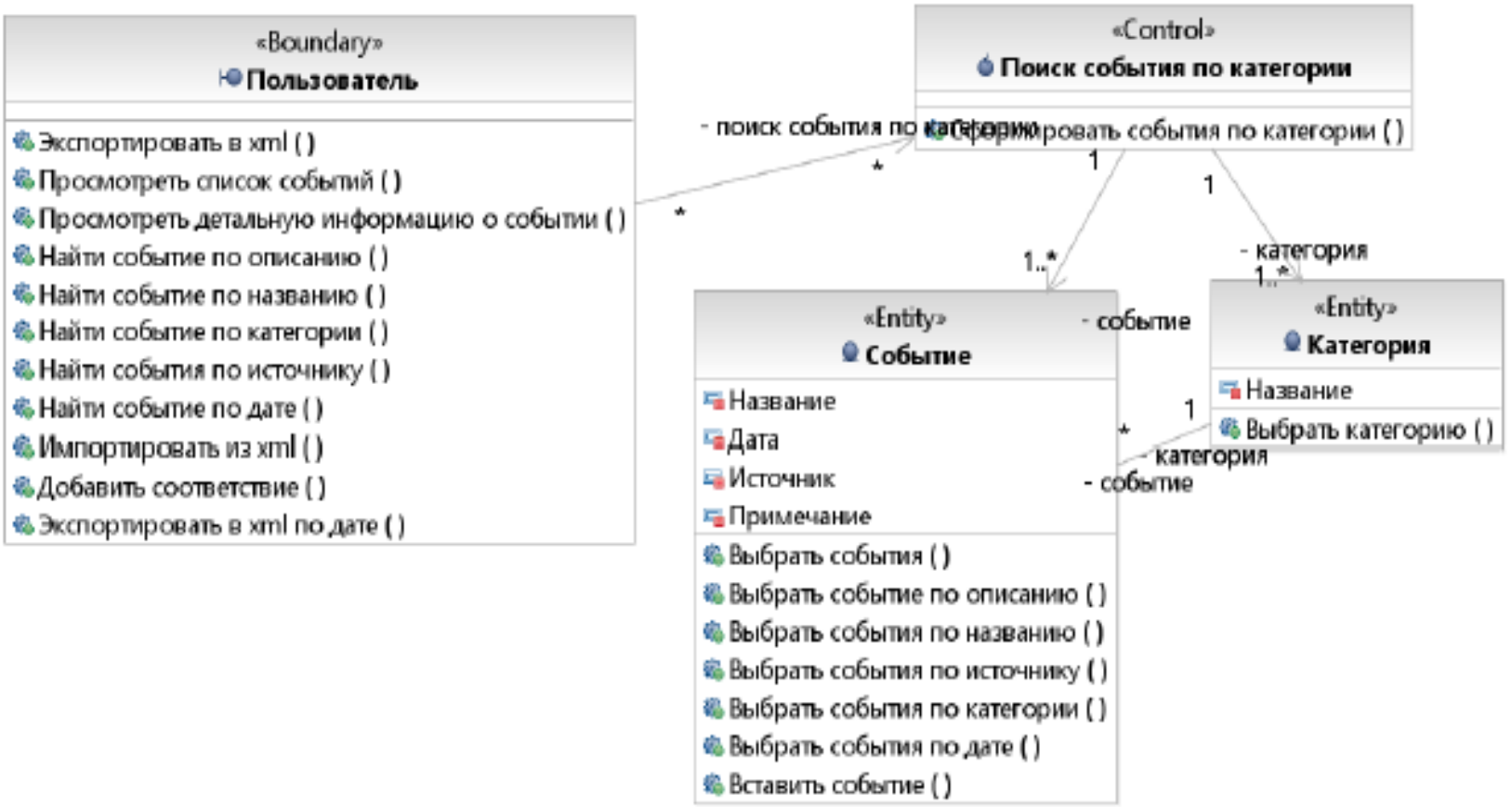


# Кооперация – пример ЖСС

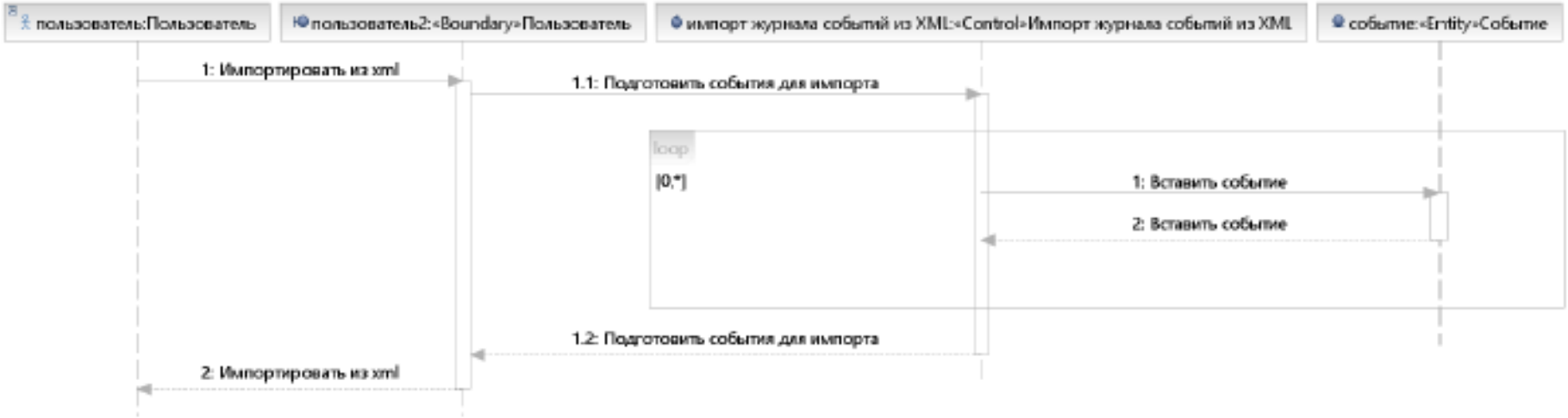
Импорт журнала событий из XML



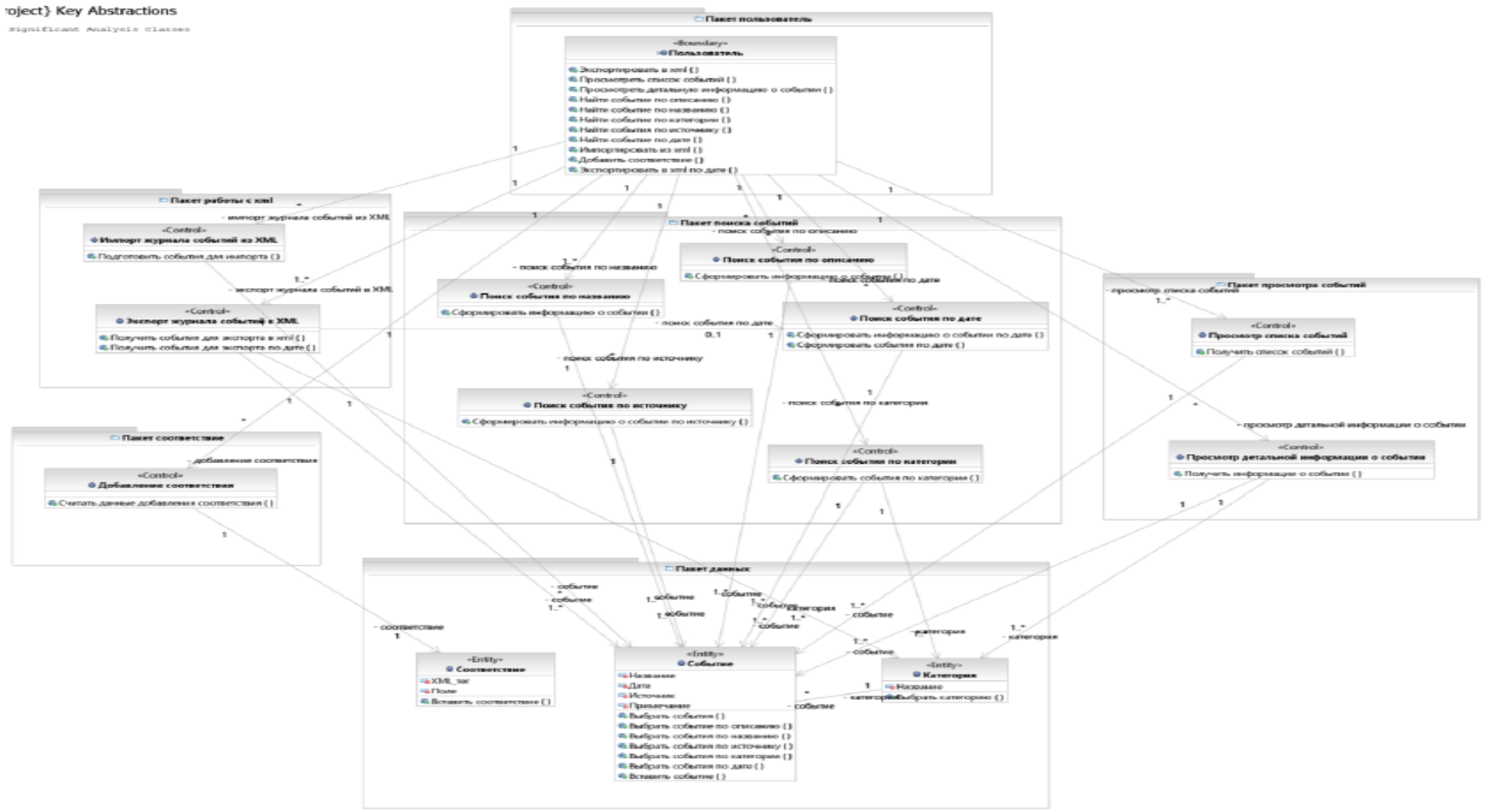
# Анализ классов – пример ЖСС



# Диаграмма последовательностей кооперации – пример ЖСС



# Анализ пакетов – пример ЖСС



# Рабочий процесс – проектирование

- Проектирование архитектуры
- Проектирование прецедентов
- Проектирование классов
- Проектирование подсистем

# Проектирование архитектуры

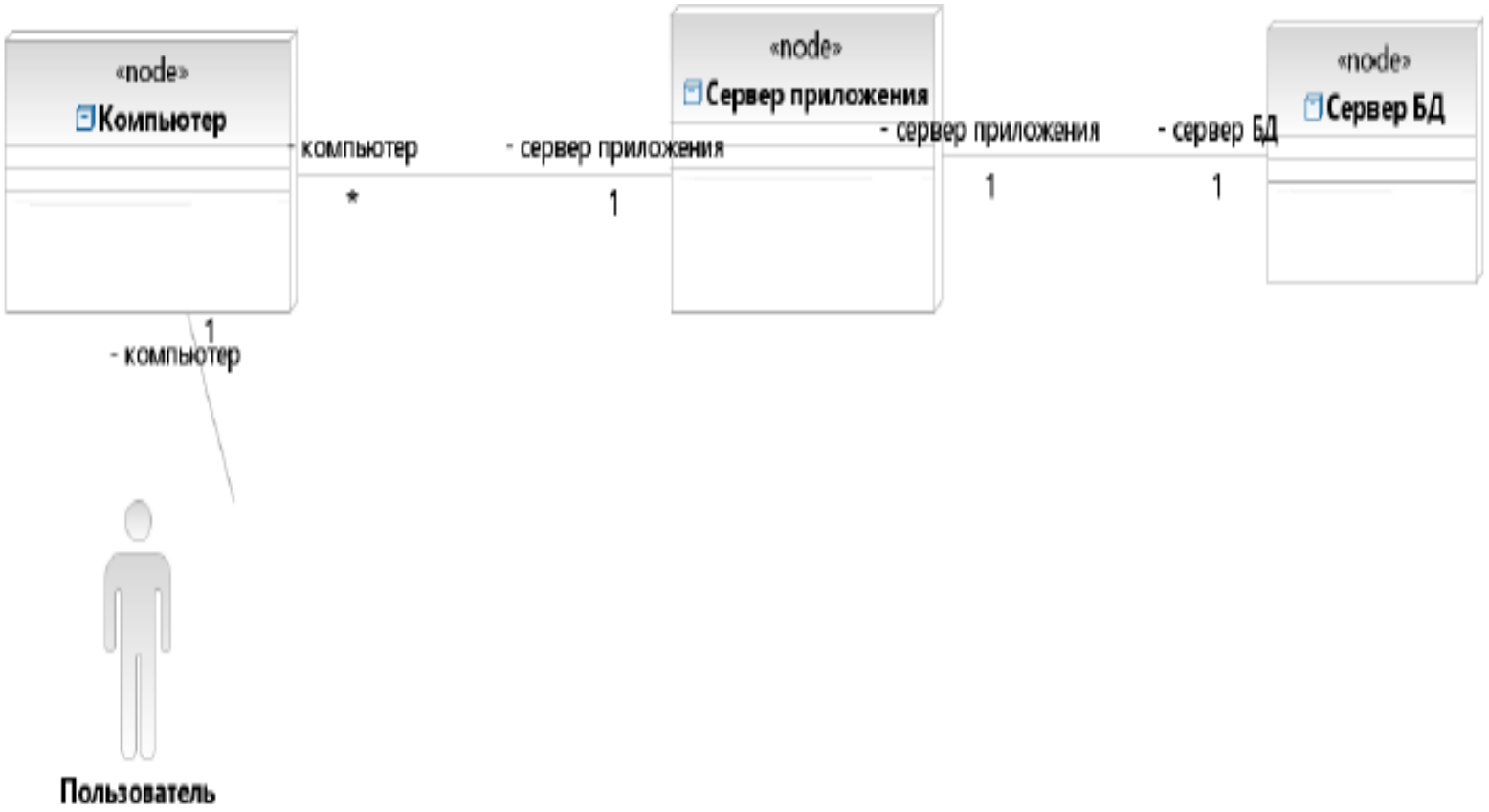
- Определение узлов и сетевых конфигураций
- Определение подсистем и их интерфейсов
  - Определение прикладных подсистем
  - Определение сервисных подсистем
- Определение интерфейсов подсистем
- Определение архитектурно значимых классов проектирования
- Определение обобщенных механизмов проектирование

# Определение узлов и сетевых конфигураций

- перечень узлов и их конфигураций
  - конфигурация сети (линии связи между узлами; протоколы; характеристики передачи; скорость; качество;...)
- дополнительные требования
- конфигурация для тестирования / моделирования ПО

=> диаграмма развертывания

# Диаграмма развертывания – пример ЖСС

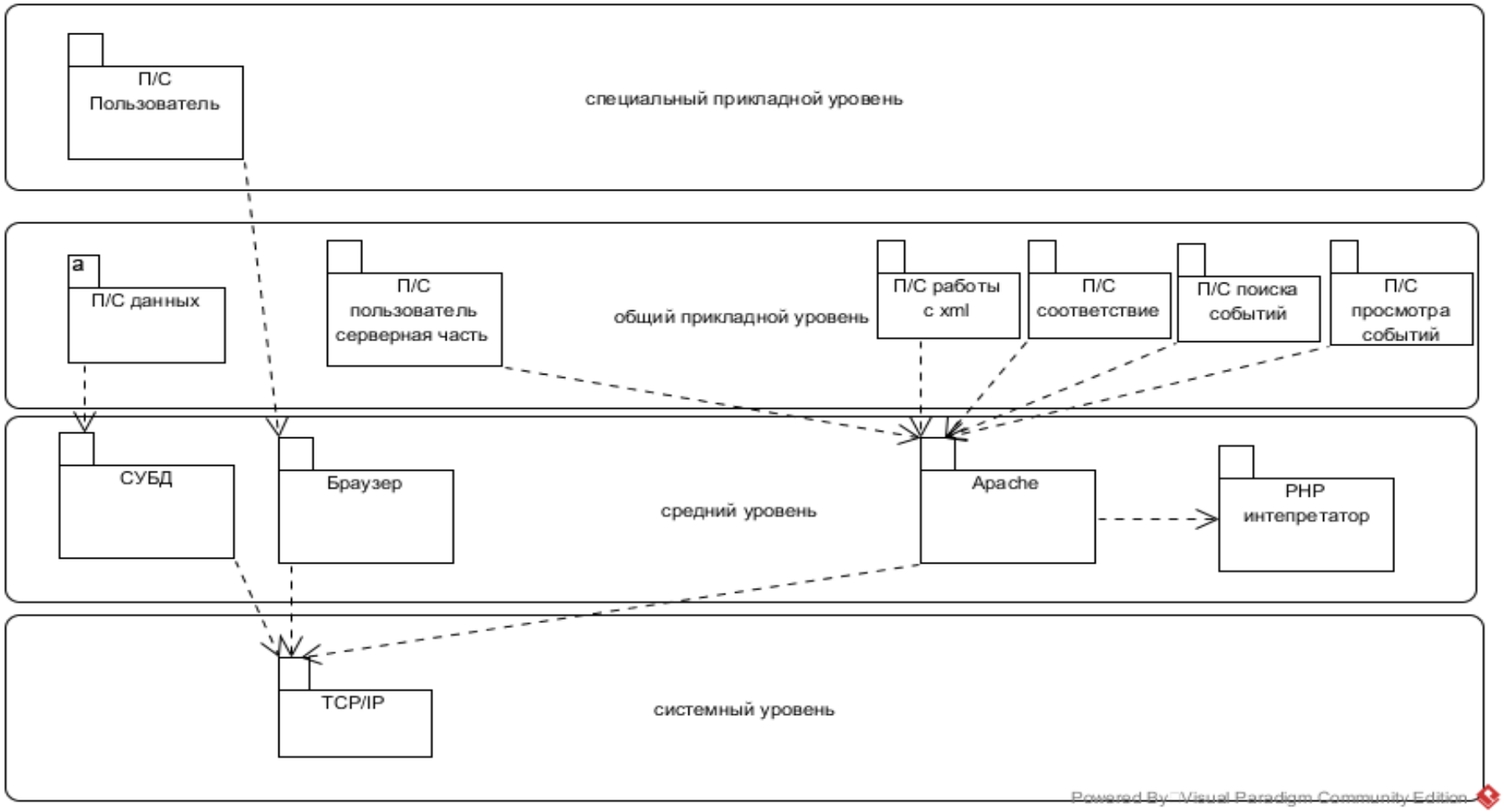


# Определение подсистем и их интерфейсов

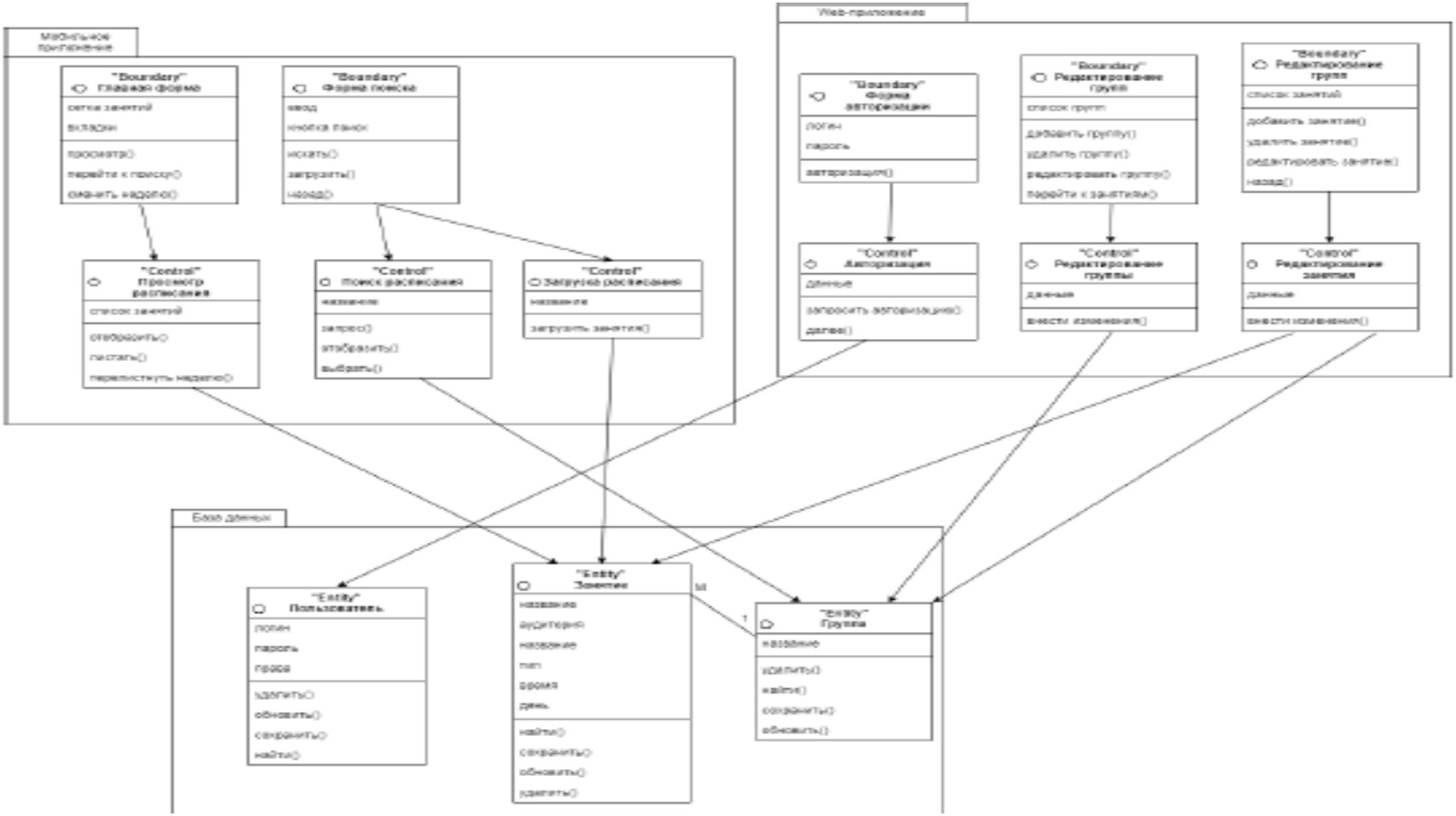
- Определение прикладных подсистем
  - пакет анализа трассируется в подсистему проектирования
  - декомпозиции при совместном использованиеи
  - разнесение по узлам
- Определение подсистем среднего уровня и уровня системного ПО
  - ОС, СУБД, коммуникация,  распределение, транзакции
  - Интеграция покупаемых / создаваемых компонентов
  - Оценка эффективности и  пригодности
- Определение зависимостей между подсистемами

- => диаграмма уровней подсистем

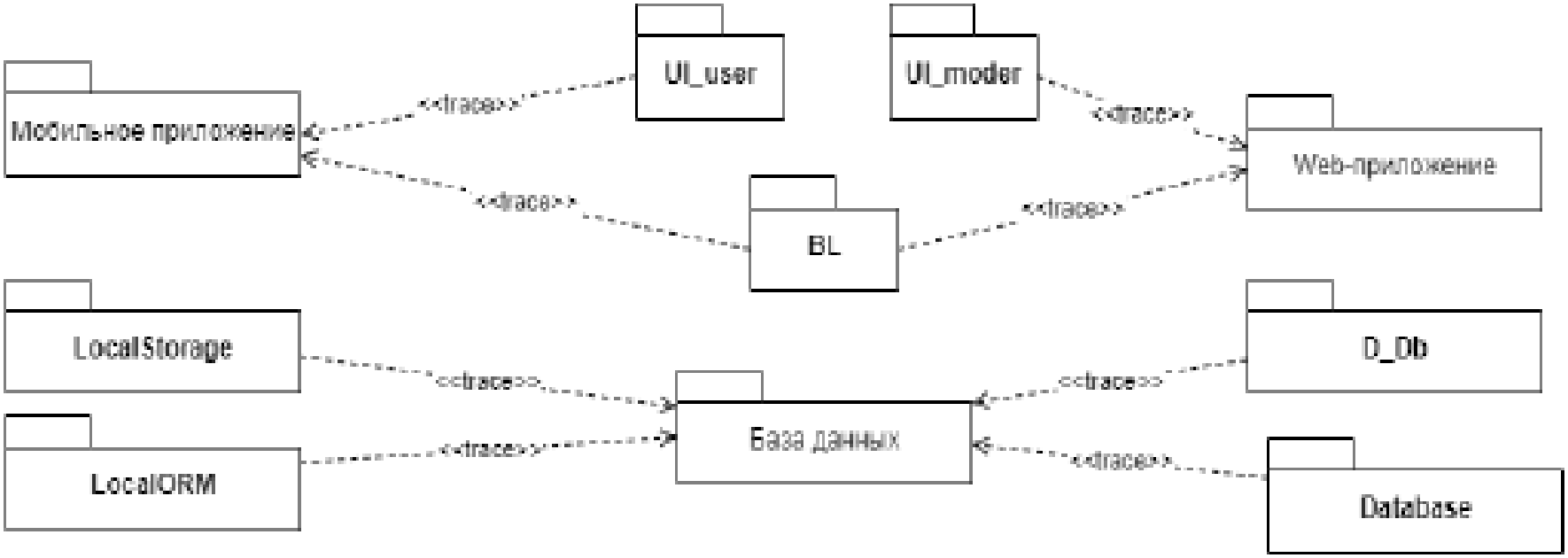
# Диаграмма уровней подсистем – пример ЖСС



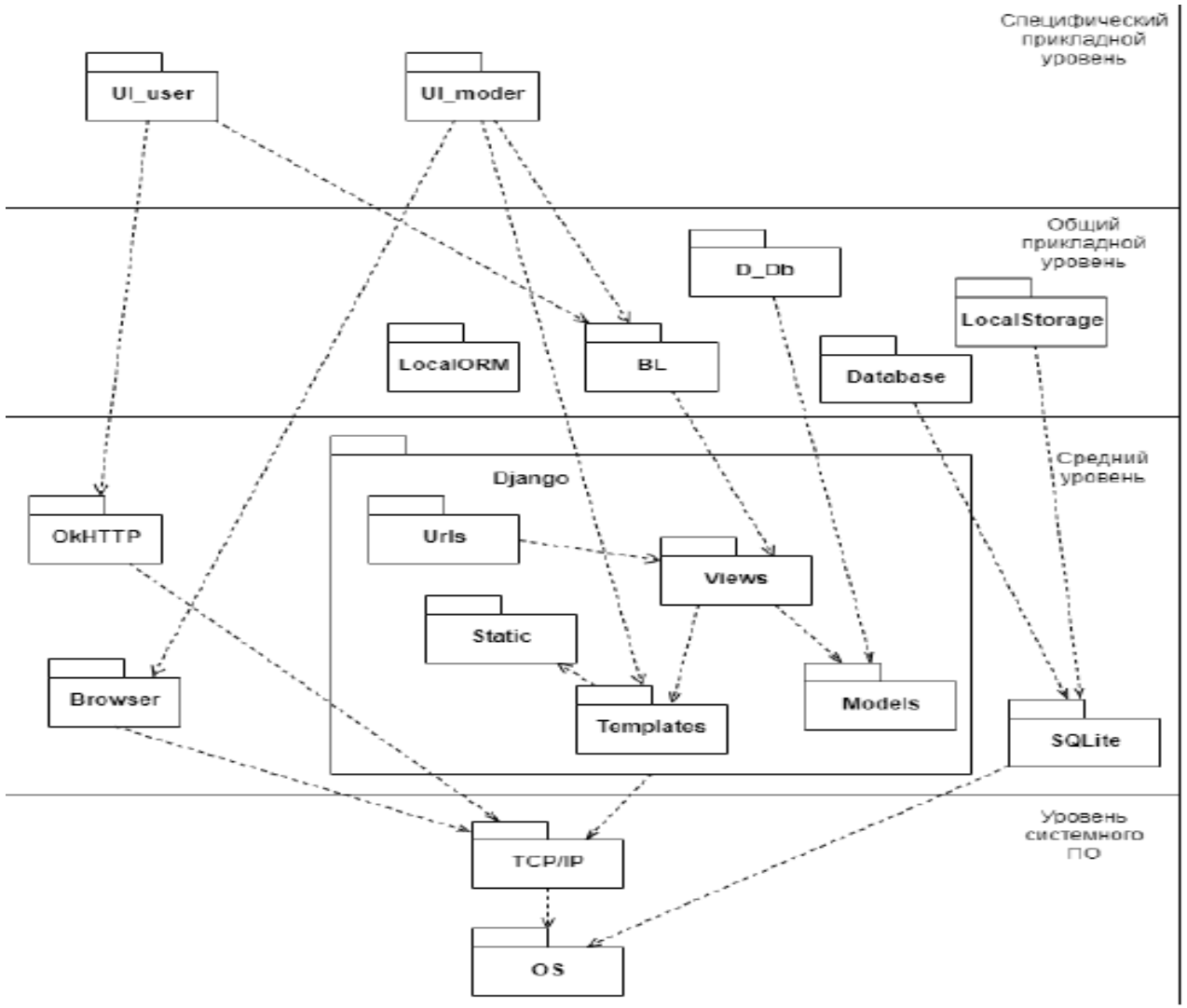
# Пакеты анализа - пример



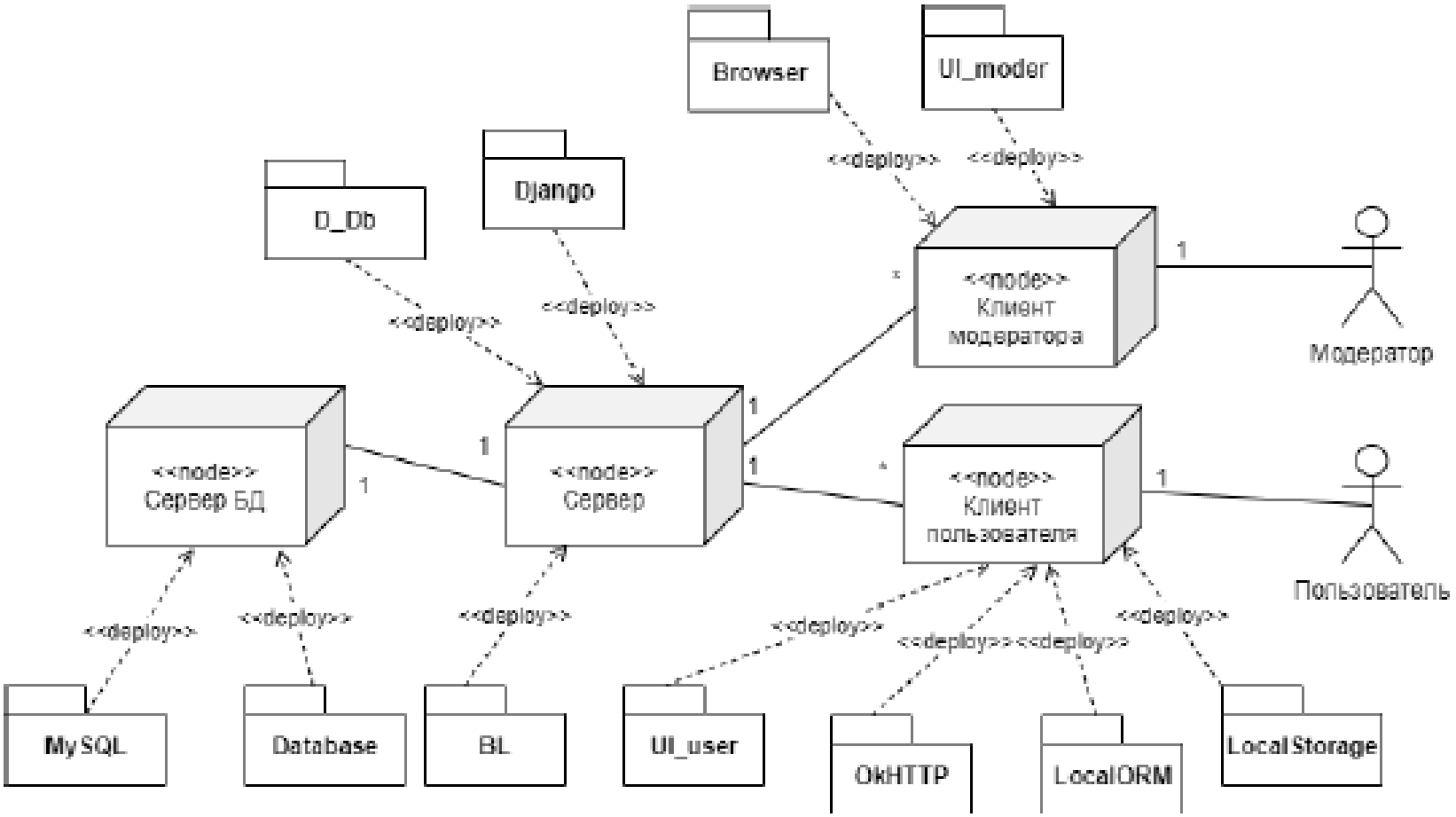
# Трассировка - пример



# Диаграмма уровней подсистем – пример, фрагмент



# Диаграмма распределения подсистем по узлам - пример



# Определение интерфейсов подсистем

- перечень операций, доступных извне;
- на основе классов в пакетах анализа (к которым обращаются извне);
- на основе зависимостей подсистем;
- начать с подсистем верхнего уровня;

- Для среднего и нижнего уровня могут быть существующие заранее (существующее ПО)

# Определение архитектурно-значимых классов проектирования

- Определение классов проектирования на основе сущностных классов анализа.
- Определение активных классов (своя нить управления, процесс)
  - один или более на узел;
  - один на обмен между узлами;
  - для повышения производительности (ответ без задержек);
  - для запуска / остановки / конфигурации / снятие блокировки / восстановление и т.д.

- Дальнейшая трассировка:
  - активный класс -> исполняемый файл;
  - подсистема -> компонент

# Определение обобщенных механизмов проектирования

- Реализуют требования:
  - длительное хранение;
  - распределение объектов;
  - средства безопасности;
  - контроль и исправление ошибок;
  - управление транзакциями.
- Выбор существующих технологий проектирования и реализации
- Обобщенные механизмы проектирования:
  - для хранения -> реляционная база данных, файл, объектно-ориентированная база данных,...
  - распределенная обоботка -> java.rmi, cobra, com,...
- Использование обобщённых кооперацій - образцов, паттернов
- Обычно:
  - Поставляемое ПО -> средний уровень и уровень СПО
  - Создаемое ПО -> прикладной уровень

# Проектирование прецедентов (кооперацій)

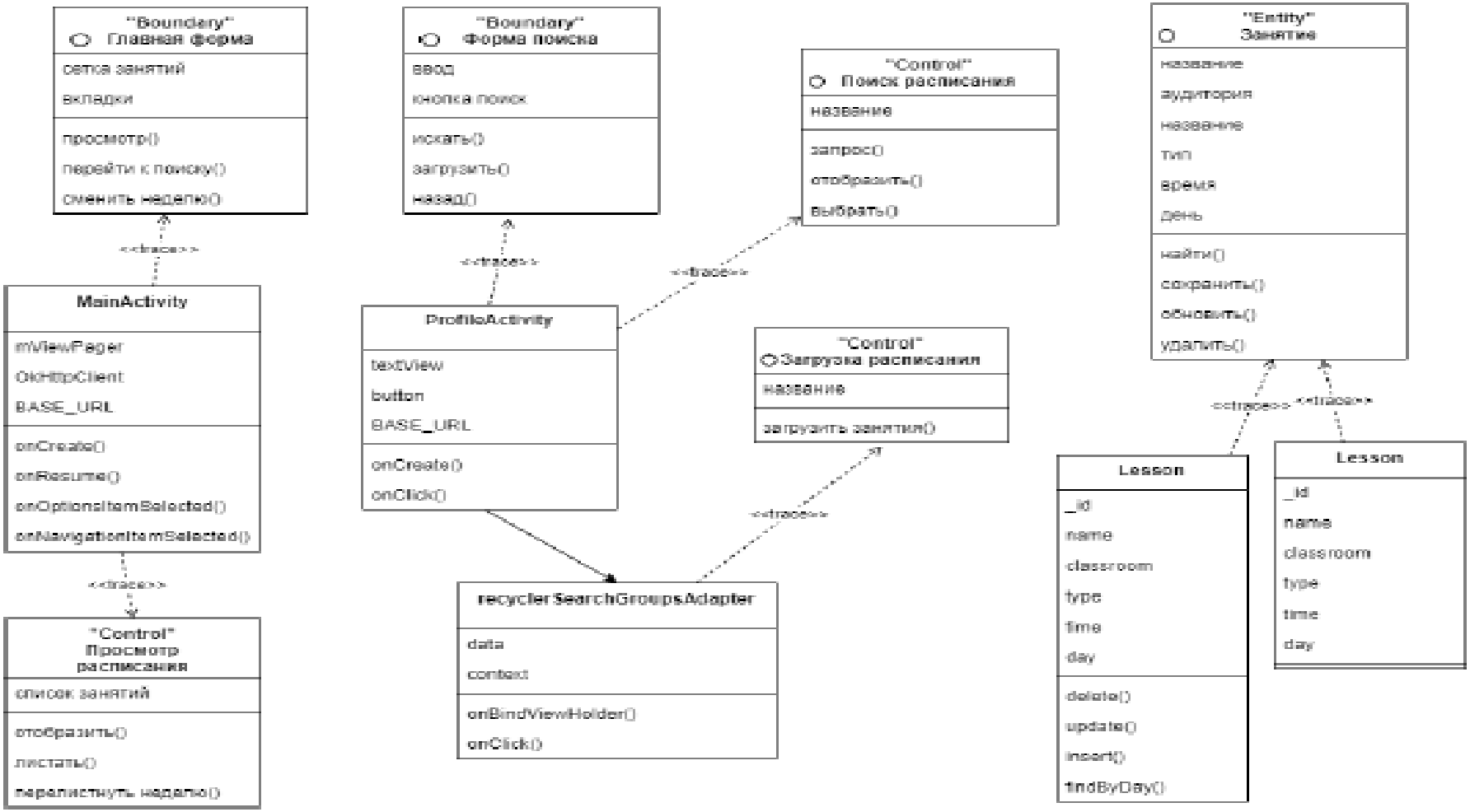
- Определение участвующих в кооперации классов проектирования
- Описание взаимодействия объектов проектирования
- Определение участвующих подсистем и их интерфейсов
- Описание взаимодействия подсистем
- Определение специальных требований к кооперациям

# Трассировка классов анализа в классы проектирования

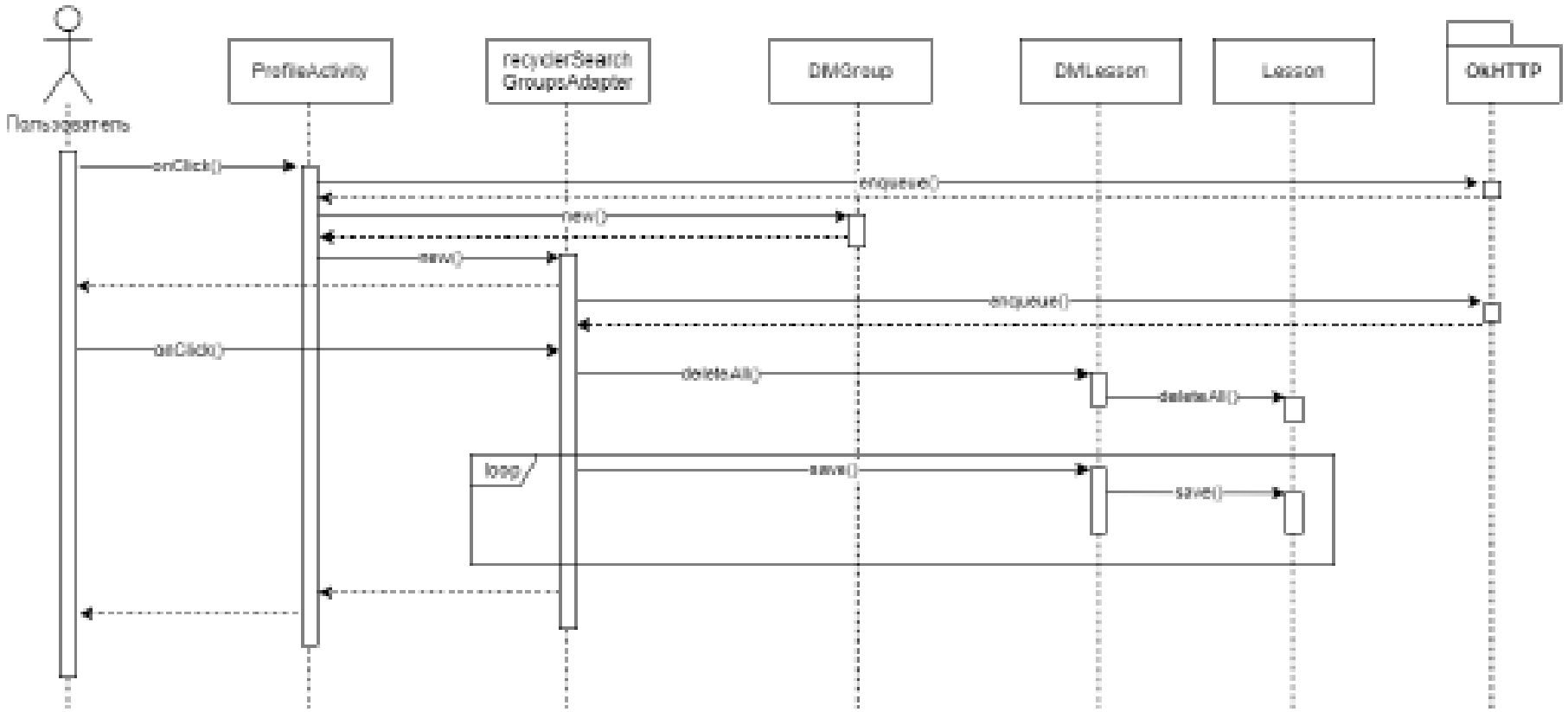
- **Граничные** – в классы форм и их компонентов (производные от стандартных)
- **Сущности** – в таблицы БД, файлы, объекты в ОП
- **Управляющие** – в классы реализации бизнес-логики

- если класса проектирования нет, то его надо определить

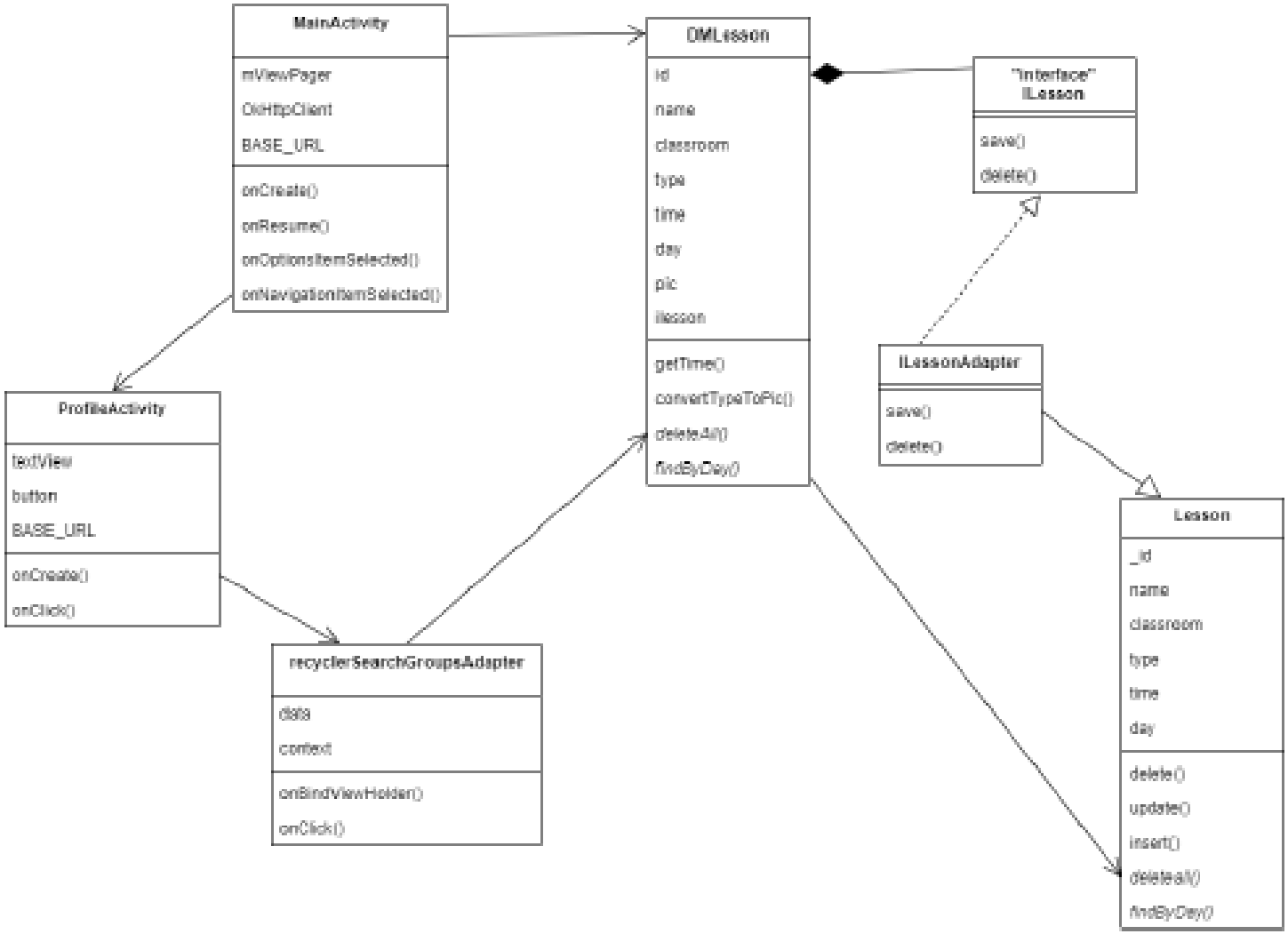
# Трассировка граничного класса - пример



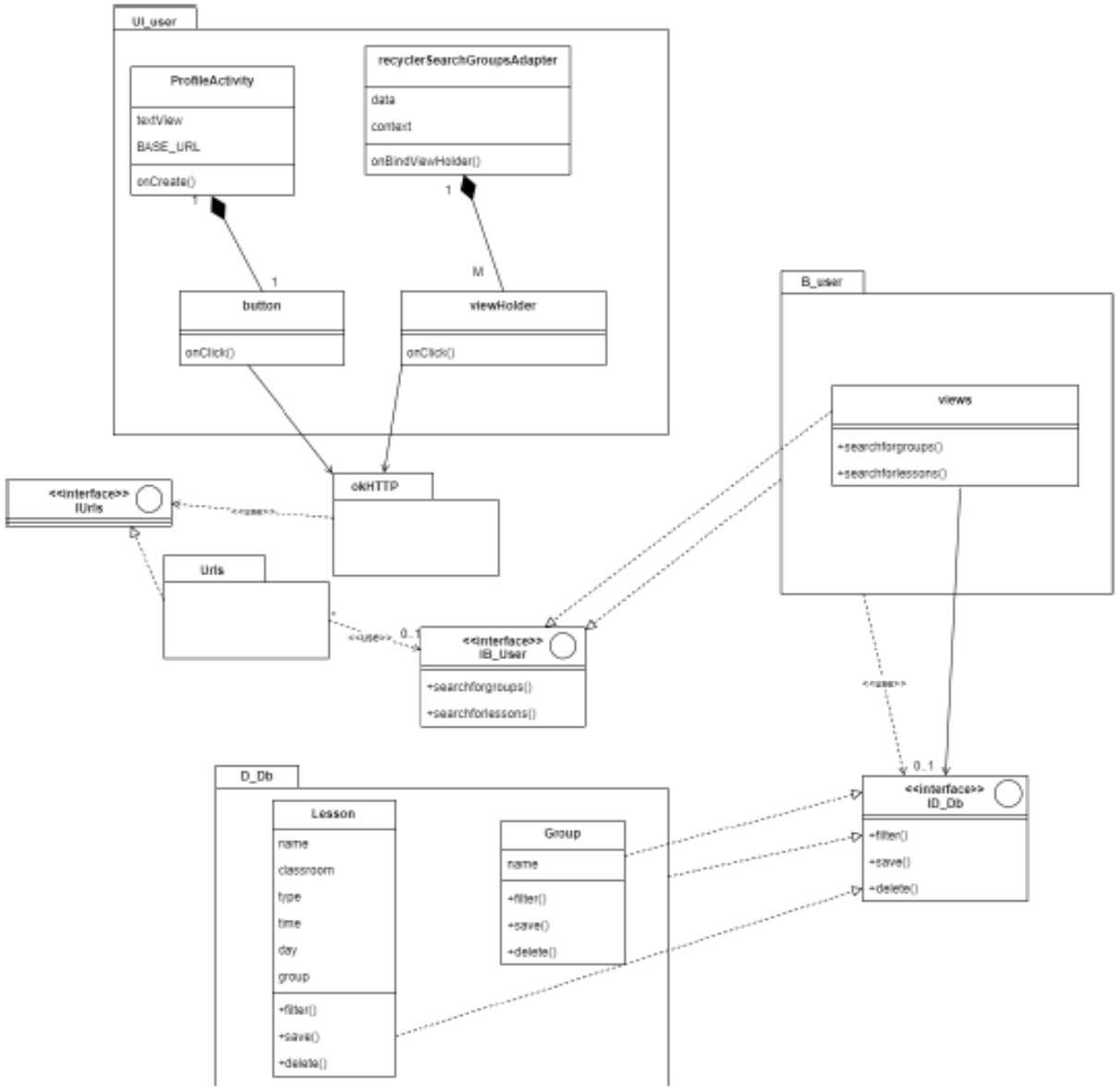
# Диаграмма последовательности объектов - пример



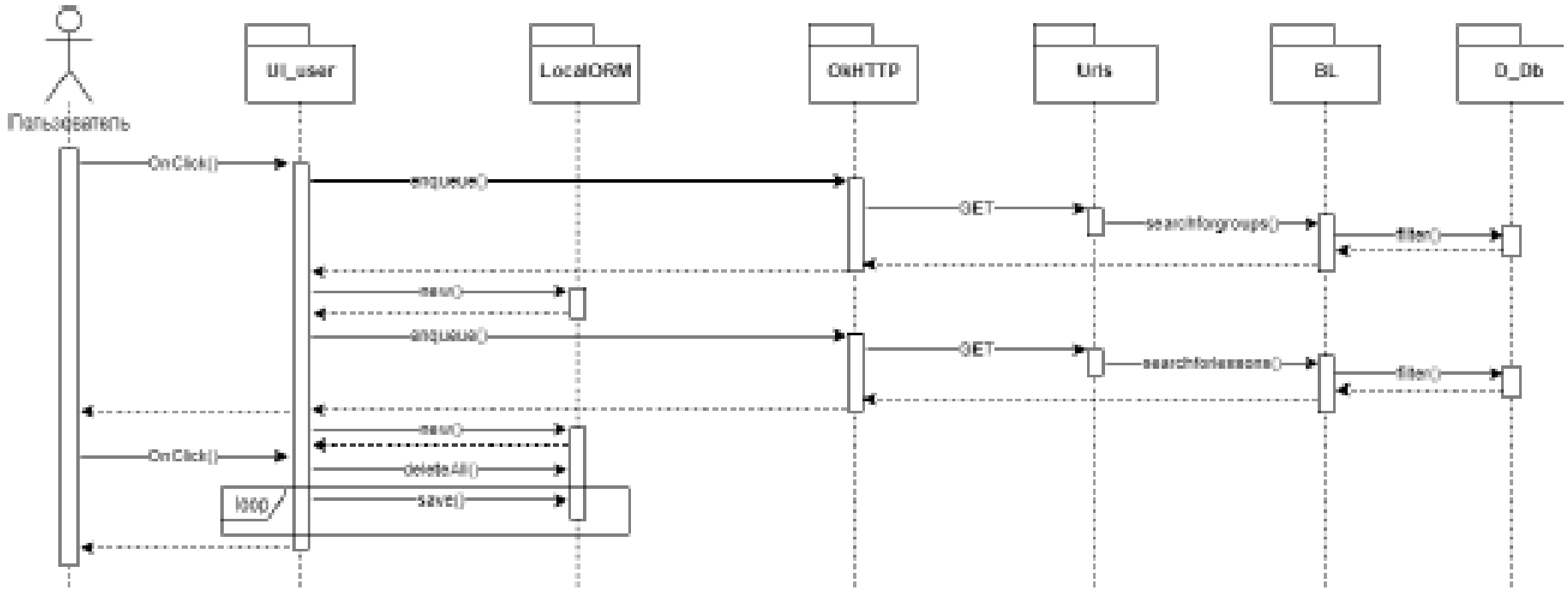
# Диаграмма классов проектирования - пример



# Распределение классов по подсистемам



# Диаграмма последовательности подсистем - пример



# Проектирование классов

- - операции (сигратуры в терминах языка);
  - атрибуты (стандартные типы);
  - отношения (ассоциации, агрегации, обобщение);
  - методы реализации (алгоритмы)
  - состояния
- - зависимость от обобщенных механизмов проектирования
  - требования к реализации
  - правильность реализации всех интерфейсов

# Проектирование подсистем

- сильная связность и слабое сцепление (перемещение классов)
- выполнение задач
- правильный интерфейс.

- Сохранение зависимостей между подсистемами
  - Связь классов подсистемы => связь подсистем
    Желательнее связь между интерфейсами, чем подсистемами
- Сохранение предоставляемых подсистемой интерфейсов
  - Операции интерфейса подсистем должны соответствовать всем ролям подсистем.
- Сохранение содержимого подсистемы
  - Для любого интерфейса подсистемы должен быть класс его реализации;
  - Для любого интерфейса / любого прецедента должна быть кооперация в терминах классов подсистемы

# ДЗ-2

- Срок – 10 ноября (защита до 30 ноября)

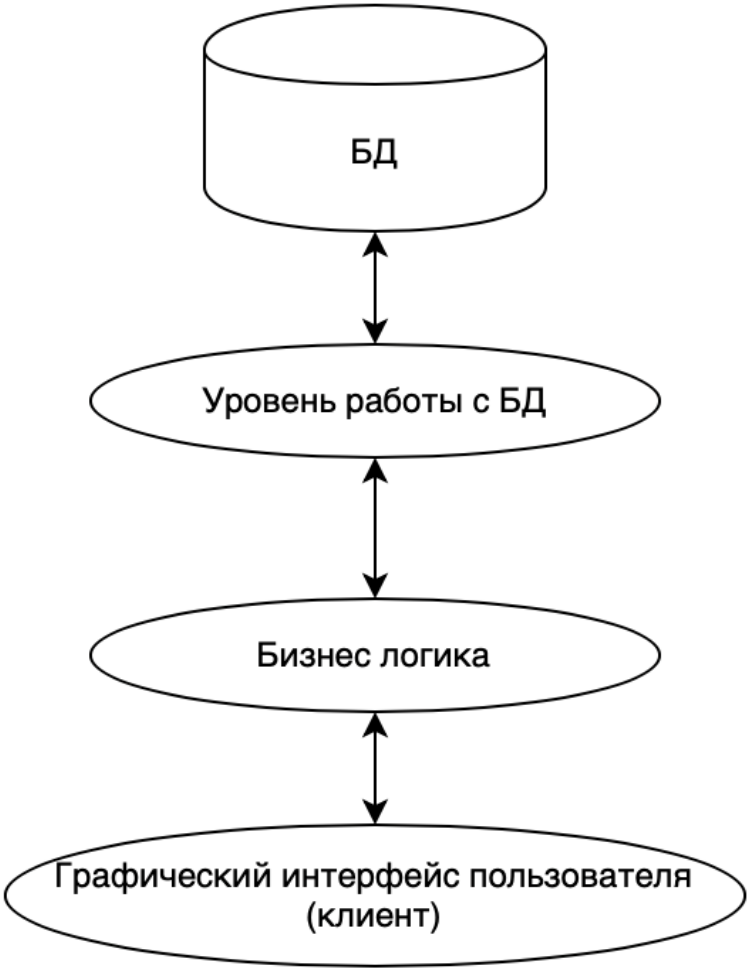
**Задание:**

- Разработать программу (основные прецеденты) по варианту.
- Реализовать в программе паттерны (по варианту) бизнес-логики и работы с БД.
- Составить набор диаграмм классов и последовательностей, которые демонстрируют структуру и поведение программы.
- Отдельно составить диаграммы классов и последовательностей для иллюстрации примененных паттернов.

# Паттерн

- Типовое проектное решение
- Повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования, в рамках некоторого часто возникающего контекста
- Описывается кооперацией
  - Диаграмма классов
  - Диаграмма взаимодействия/ последовательности

# Уровни приложения



# Паттерны работы с базой данных:

- Row Data Gateway (Шлюз записи данных)
- Table Data Gateway (Шлюз таблицы данных)
- Active Record (Активная запись)
- Data Mapper (Преобразователь данных)

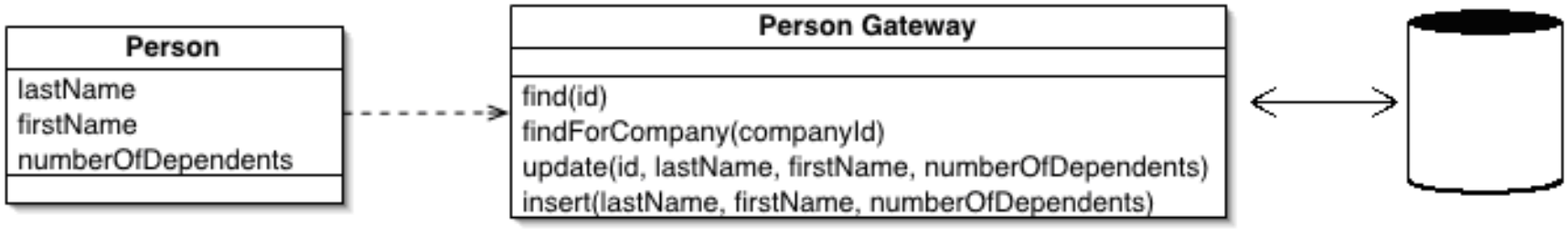
# Table Data Gateway (Шлюз таблицы данных)

- *Объект выступает в качестве шлюза между данными в приложении и в БД.*

- Объект (Класс) содержит методы для доступа к отдельной таблице или представлению (view): выборка, обновление, вставка, удаление (CRUD) и вызов хранимых процедур.
- Один объект работает сразу со всеми записями в таблице.
- Может быть один шлюз на несколько таблиц
- М.б. Класс со статическими методами

- **Вход методов:** параметры для формирования запроса
  - Для insert - все поля (атрибуты) объекта
  - Для update – атрибуты сущности БД + ид
  - Для delete - часть параметров (ид) для корректного определения объекта и удаления его
- **Выход методов:** поиск: Record set, Коллекция, Объект; или признак удачной операции

# Table Data Gateway - пример

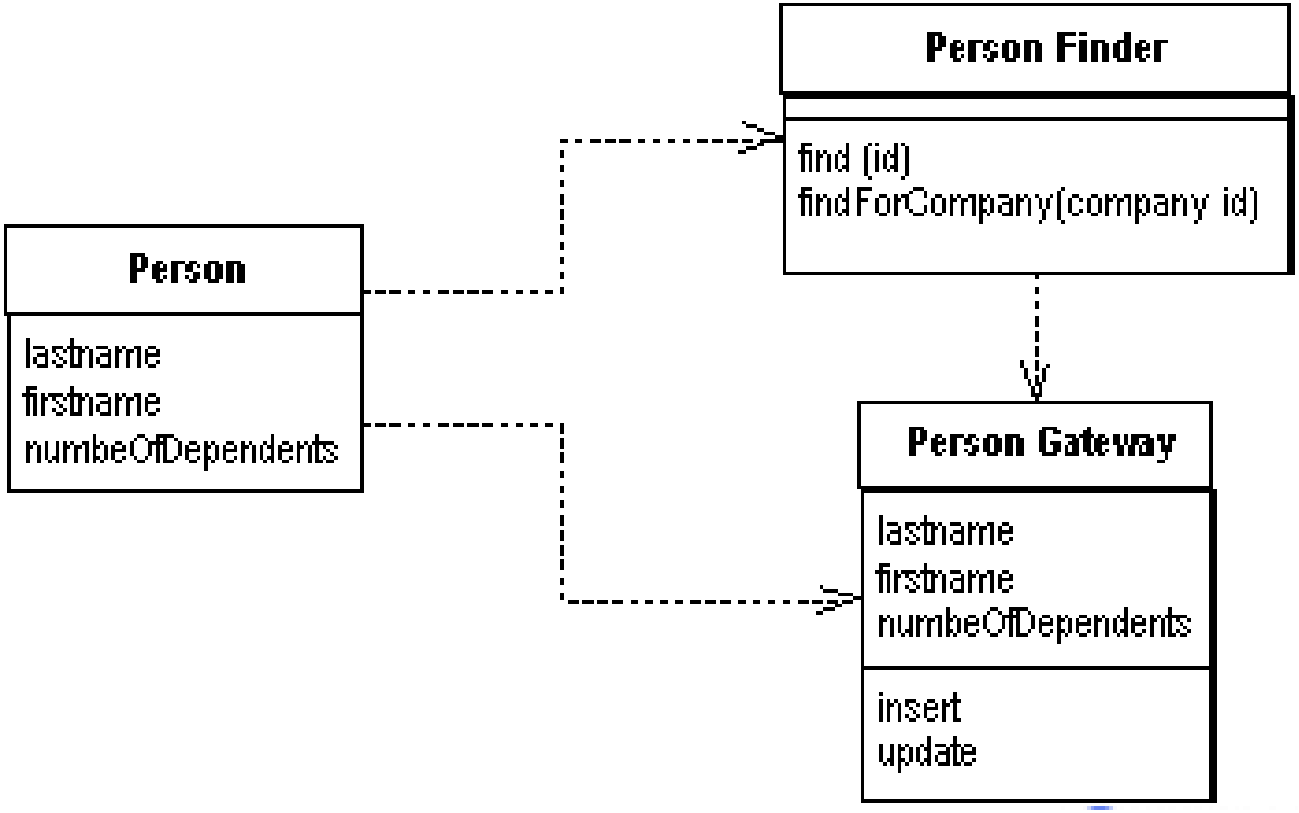


# Row Data Gateway (Шлюз записи данных)

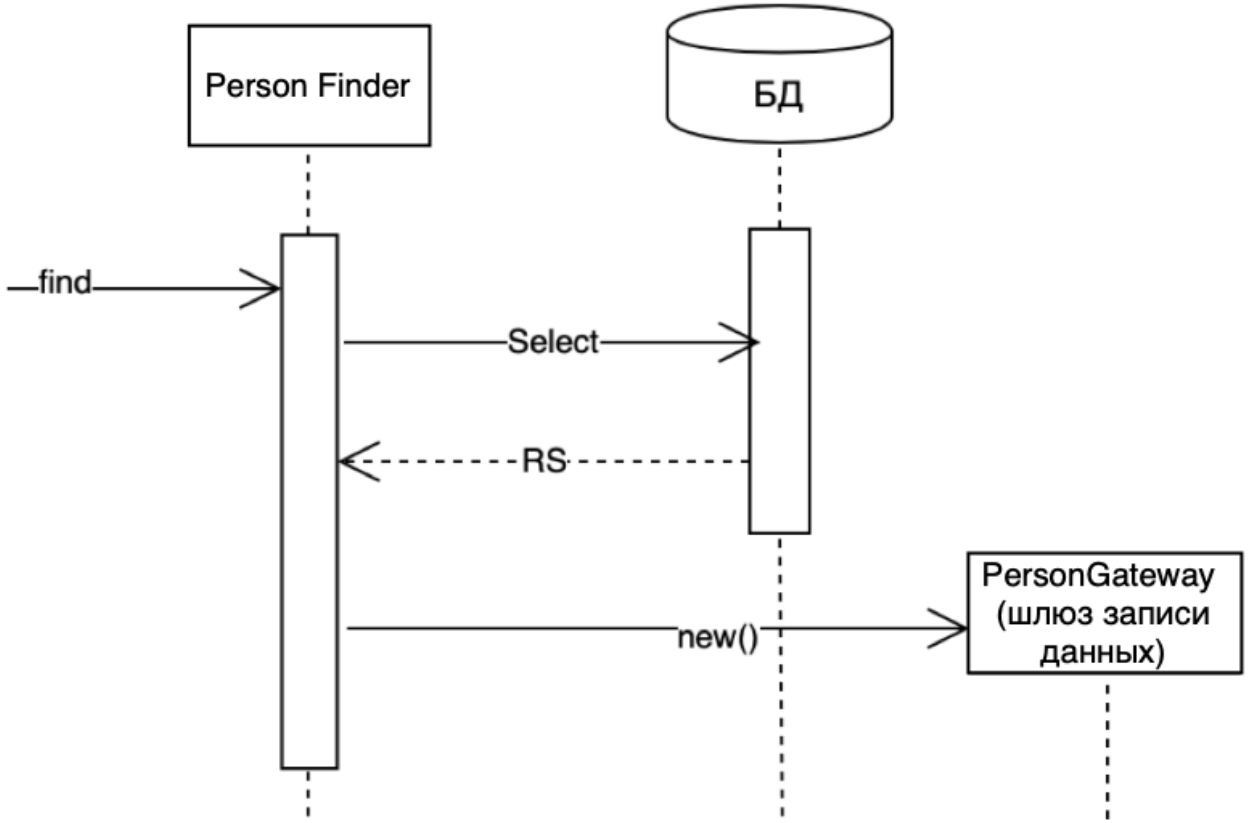
- *Объект выступает в роли шлюза к отдельной записи в источнике данных. Один экземпляр на одну запись.*

- 1 объект - это одна запись из таблицы БД,
- атрибуты объекта включают все поля записи из БД,
- методы CRUD работают без параметров,
- методы CRUD берут данные для работы из свойств объекта,
- методы поиска статичны, либо прописываются отдельно(class PersonFinder).

# Row Data Gateway - пример



# Row Data Gateway - пример



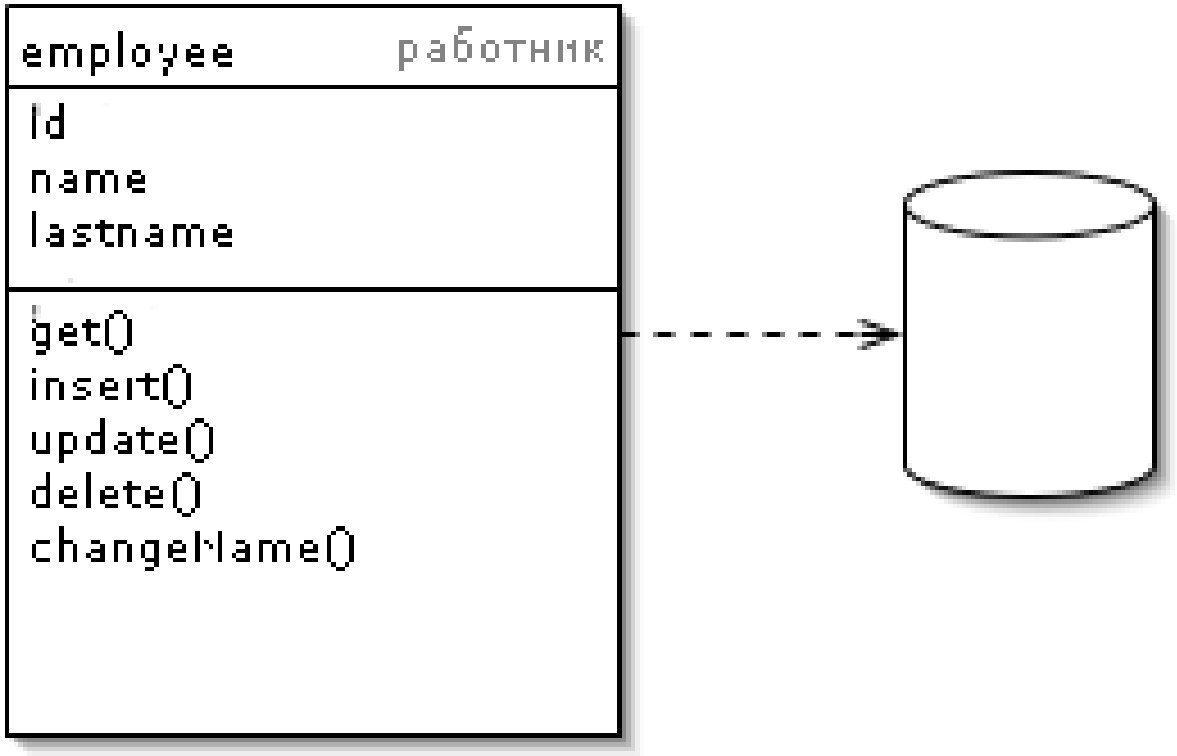
# Active Record (Активная запись)

- *Объект, выполняющий роль оболочки для строки таблицы БД или представления. Инкапсулирует доступ к БД и добавляет к данным логику домена*

- атрибуты объекта включают все поля записи из БД,
- методы CRUD работают без параметров,
- методы CRUD берут данные для работы из свойств объекта,
- содержит бизнес- логику обращения с данными.

- Отличие от шлюза записи – наличие бизнес-логики

# Active Record - пример

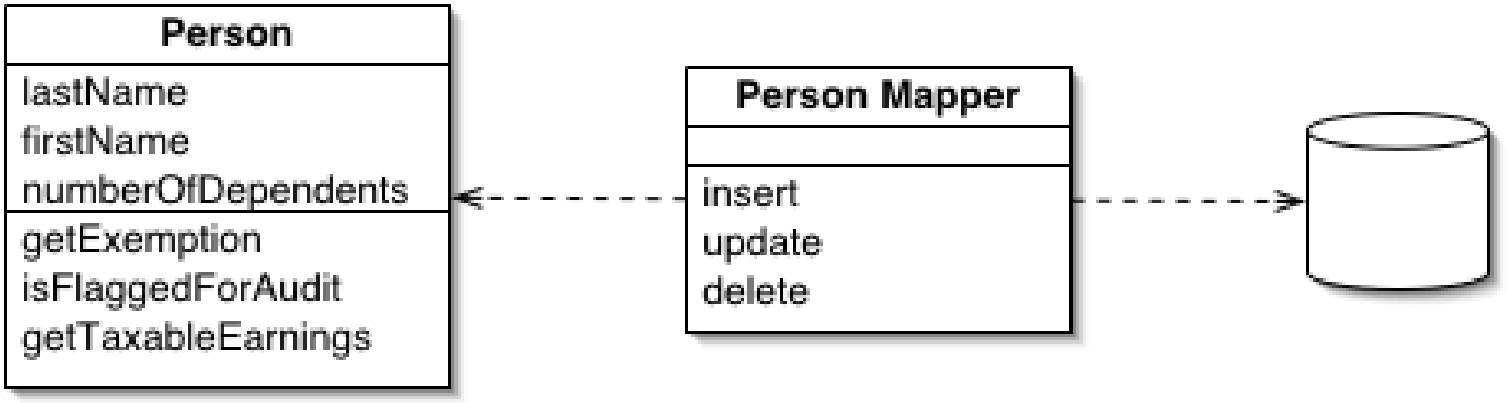


# Data Mapper (Преобразователь данных)

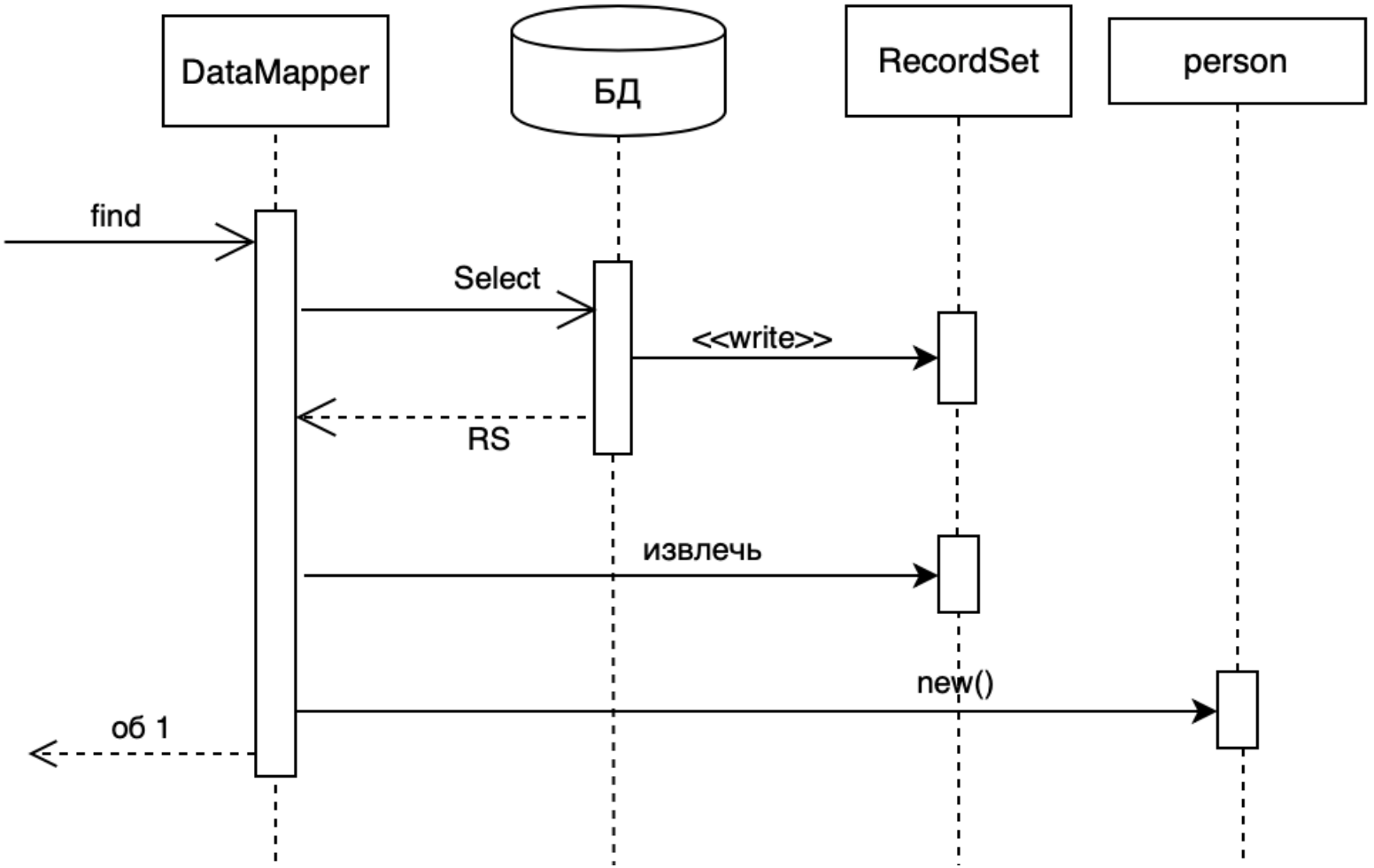
- *Слой преобразователей для передачи данных между объектами и таблицами БД, сохраняя их независимо дуруг от друга и преобразователей*

- Объекты не нуждаются в знании о существовании БД. Они не нуждаются в SQL-коде и в информации о структуре БД.
- **Вход методов:** объекты бизнес-логики (CUD) и параметры запроса поиска (R)
- **Выход методов:** объекты, коллекция объектов (поиск)

# Data Mapper


Person
lastName
firstName
numberOfDependents
getExemption
isFlaggedForAudit
getTaxableEarnings
Person Mapper
insert
update
delete

# Data Mapper


DataMapper
БД
RecordSet
person
find
Select
<<write>>
RS
извлечь
new()
об 1

# Паттерны бизнес логики:

- Transaction Script (Сценарий транзакций)
- Domain Model (Модель предметной области)
- Table Module (Модуль таблицы)
- Service Layer (Слой служб)

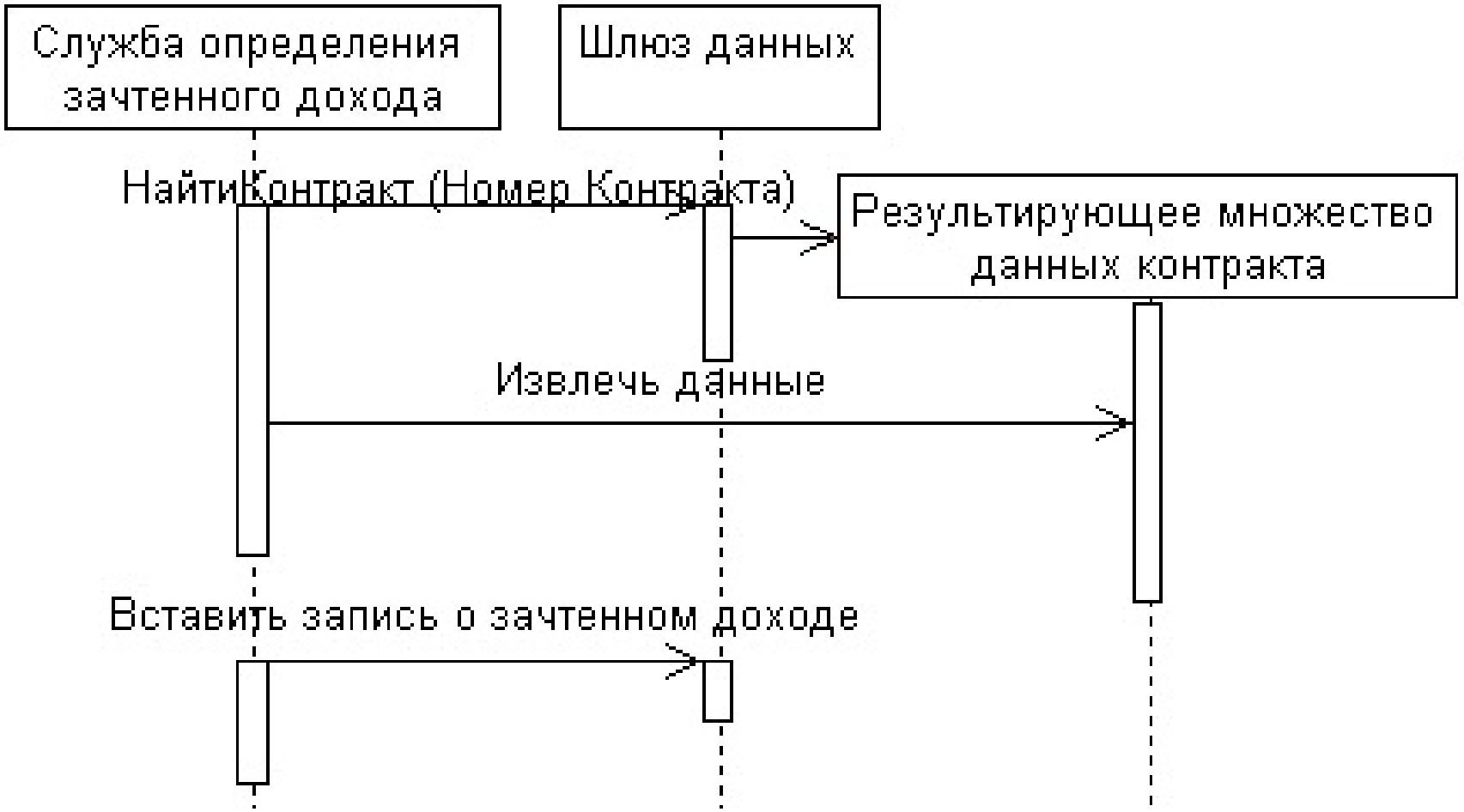
# Transaction Script (Сценарий транзакций)

- *Способ организации бизнес-логики по процедурам, каждая из которых обслуживает один запрос, инициируемый слоем представления.*

- Сценарий транзакции организует логику вычислительного процесса в виде единой процедуры, которая выполняет все действия, необходимые для выполнения прикладной функции.
- Каждой функции приложения ставится в соответствие собственный сценарий транзакции (общие подзадачи могут быть вынесены в подчиненные процедуры).

- Вся бизнес логика делится на процедуры:
  - 1 класс – одна процедура (соответствует функции приложения).
  - Базовый класс класс с методом run() от которого наследуются другие классы для реализации отдельных функций.

# Transaction Script - пример



# Transaction Script - особенности

- Просто разрабатывать.
- Характерно для небольших приложений.
- Возможно дублирование кода

**Рекомендуется использовать с:**

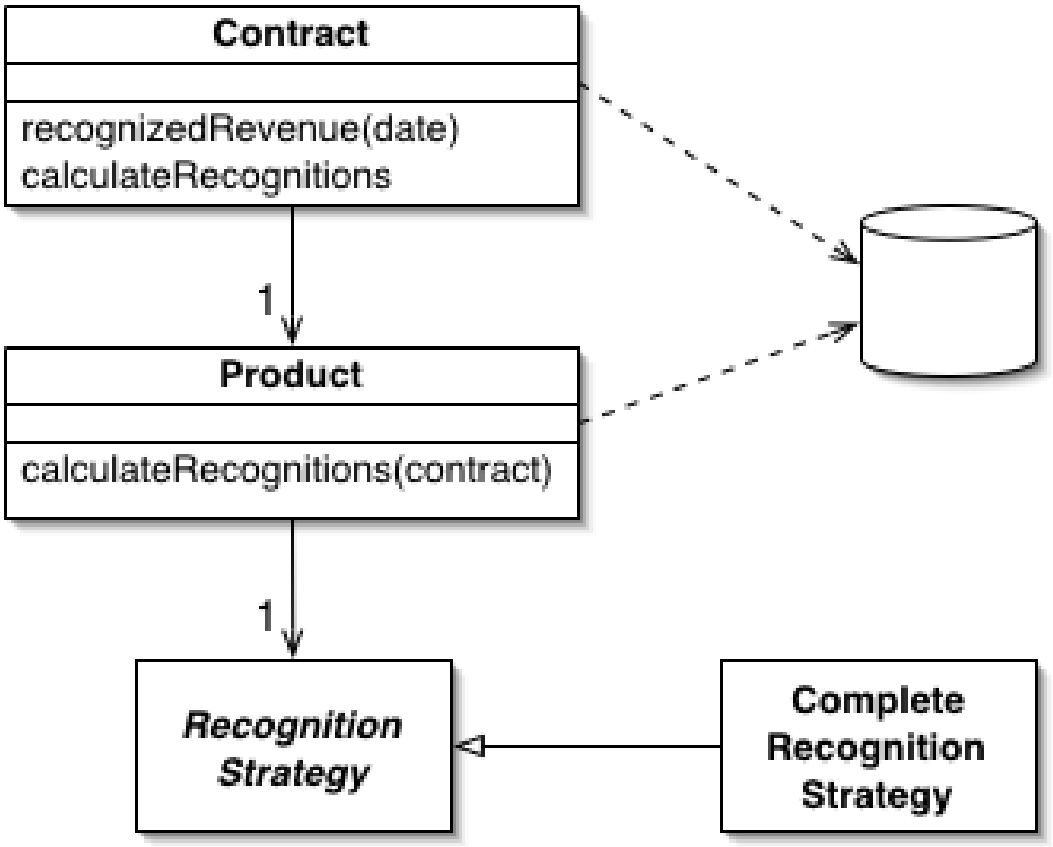
- Шлюз таблицы данных
- Шлюз записи данных

# Domain Model (Модель предметной области)

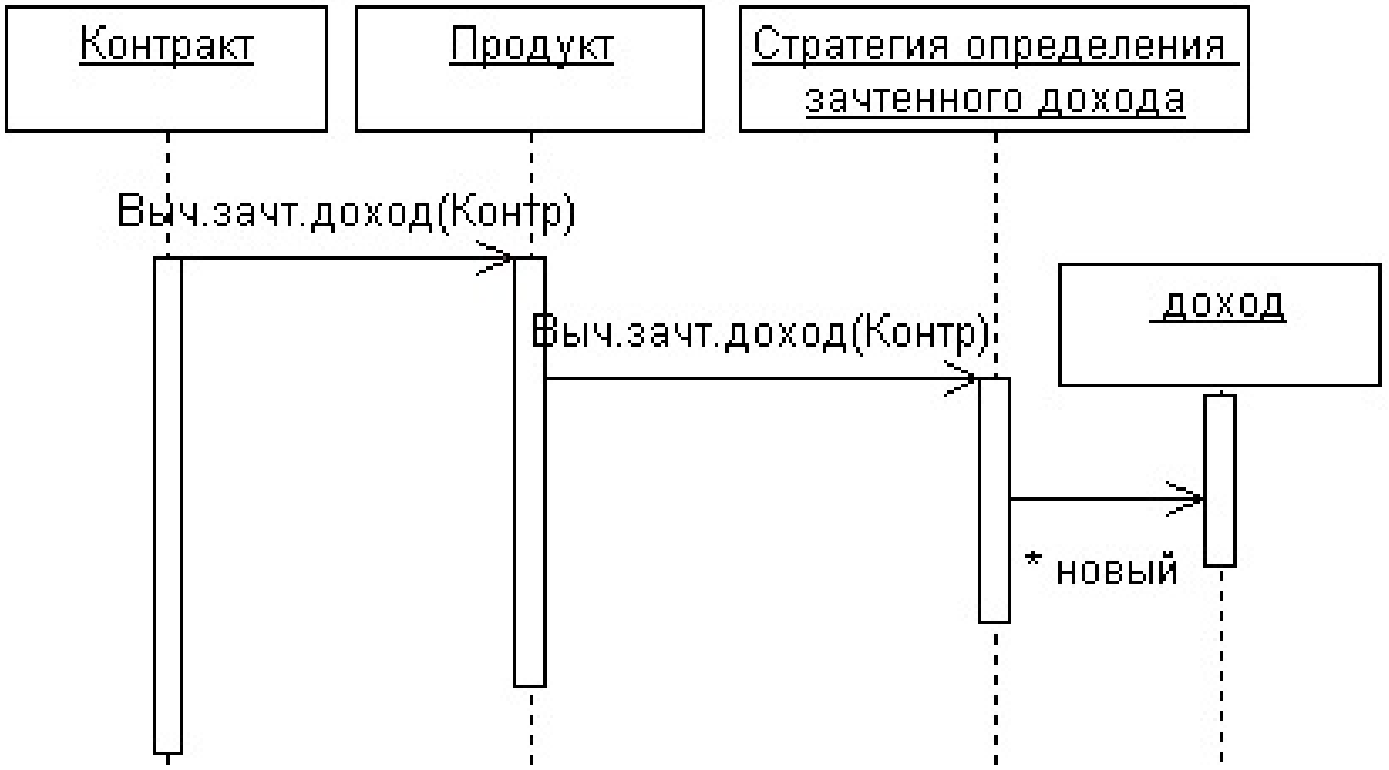
- *Предусматривает создание сети взаимосвязанных объектов, каждый из которых представляет некую осмысленную сущность.*

- Означает наполнение приложения слоем объектов, описывающих сущности реального мира и их взаимодействие.
- Одни объекты соответствуют элементам данных, которыми оперируют в этой области, а другие реализуют те или иные бизнес-правила. Функции тесно сочетаются с данными, которыми они манипулируют.
- Объект содержит данные (для одной записи/сущности БД) и бизнес-логику их обработки.
- Объект выполняет обработку только своих данных, для обработки данных другого объекта вызывает методы другого объекта. Выполнение прикладной функции приложения будет последовательностью вызовов методов всех задействованных объектов.
- Один объект хранит данные для одной записи БД.

# Domain Model - пример



# Domain Model - пример



# Domain Model - особенности

- Две разновидности:
  - "Простая" во многом походит на схему базы данных и содержит, как правило, по одному объекту домена в расчете на каждую запись таблицу.
  - "Сложная" модель может отличаться от структуры базы данных и содержать иерархии наследования, стратегии и иные шаблоны, а также сложные сети мелких взаимосвязанных объектов. Сложная модель более адекватно представляет запутанную бизнес-логику, но труднее поддается отображению в реляционную схему базы данных.

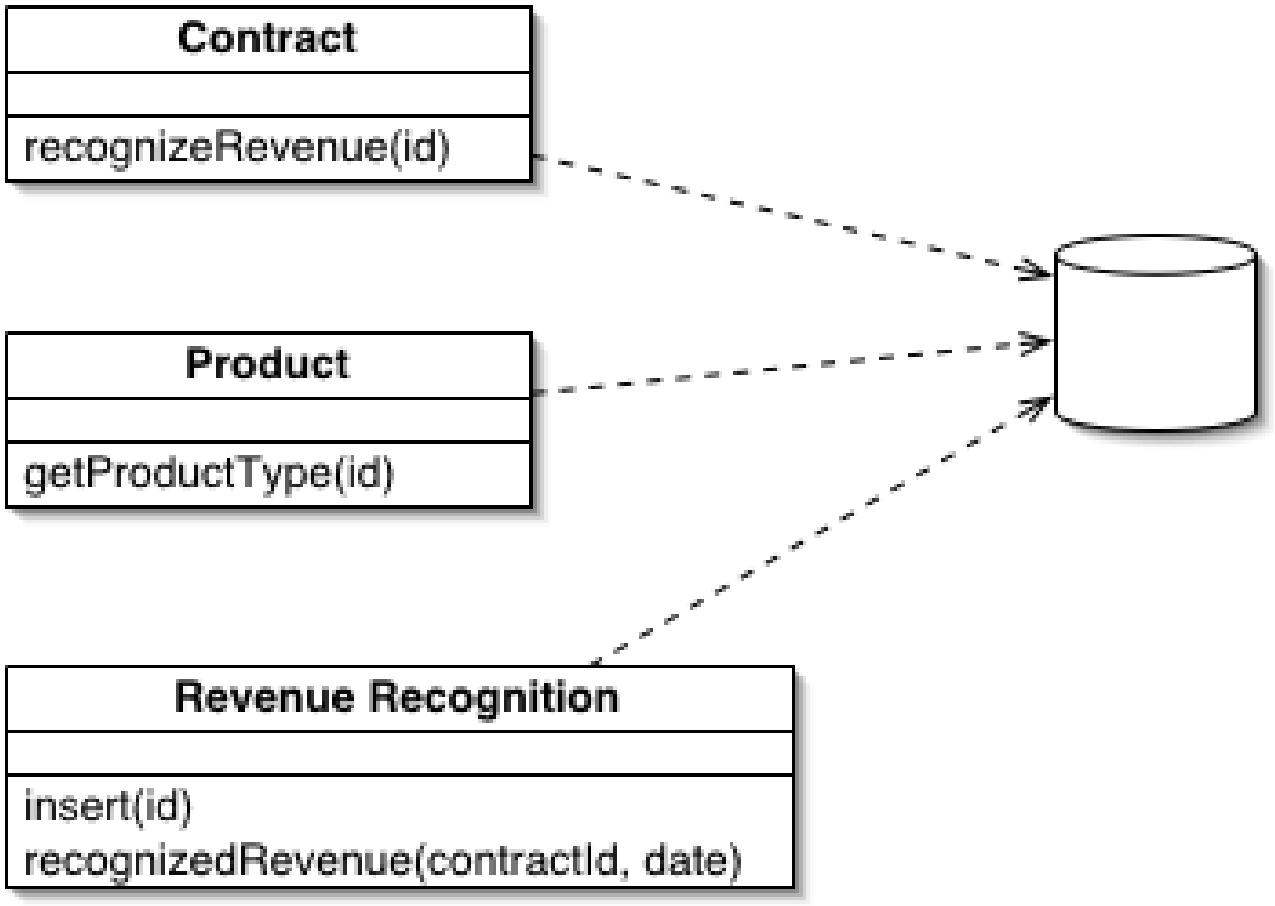
- **Рекомендуется использовать с:**
  - Активной записью
  - Преобразователь данных

# Table Module (Модуль таблицы)

- *Предусматривает создание по одному классу на каждую таблицу базы данных, и единственный экземпляр класса содержит всю логику обработки данных таблицы.*

- Основное отличие модуля таблицы от модели предметной области состоит в том, что в соответствии с моделью предметной области придется сконструировать по одному объекту на каждую запись БД, а при использовании модуля таблицы понадобится всего один объект, представляющий одновременно записи таблицы БД.

# Table Module - пример



# Table Module - особенности

- Содержит методы для обработки данных таблицы (CRUD, типовые запросы), вызова хранимых процедур.
- Возможны статические методы.
- Хранит данные таблиц, информацию о таблице, можно производить переход по связям (запрос к нескольким таблицам).
- Не содержит упоминания об идентификационном признаке объекта.

**Рекомендуется использовать с:**

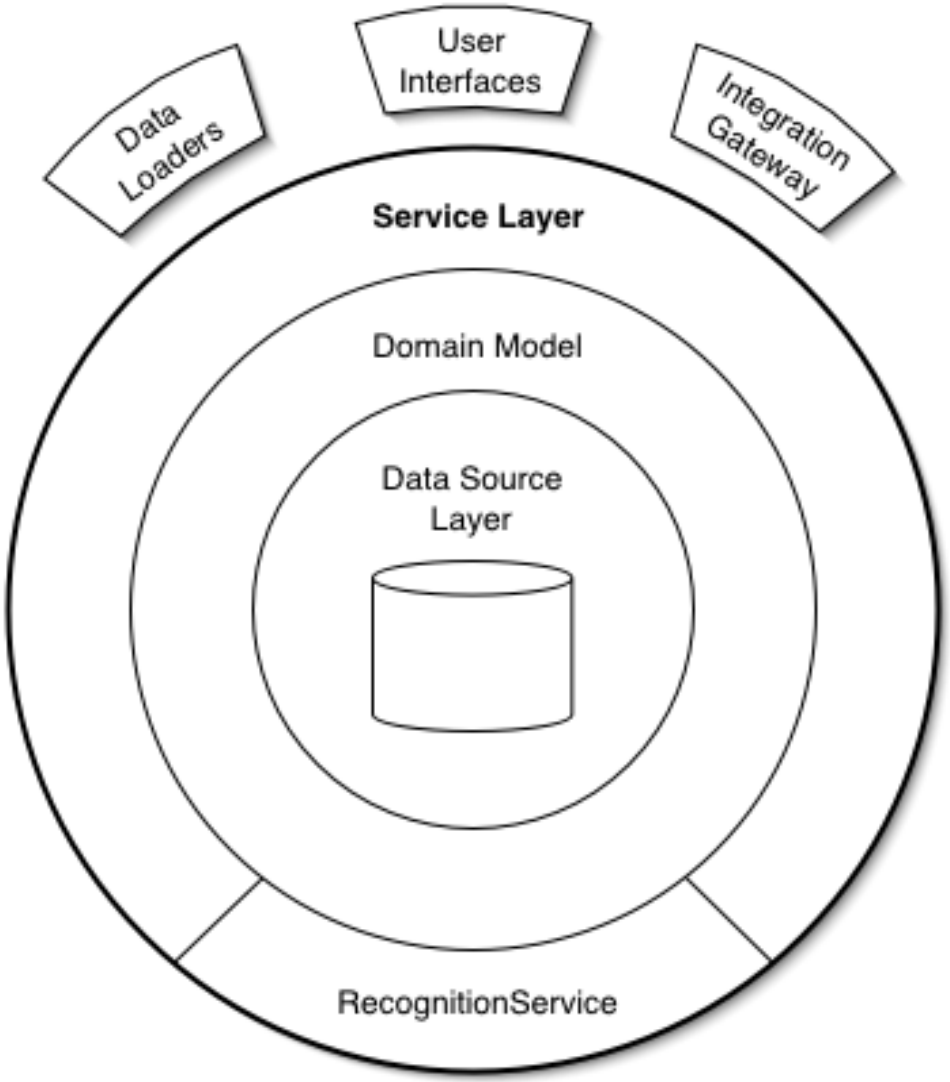
  – Шлюз таблицы данных

# Service Layer (Слой служб)

- *Определяет границу между приложением и слоем сервисов, который образует набор доступных операций и управляет ответом приложения в каждой операции.*

- Определяет для приложения границу и набор допустимых операций с точки зрения взаимодействующих с ним клиентских модулей.
- Он инкапсулирует бизнес-логику приложения, управляя транзакциями и управляя ответами в реализации этих операций.

# Service Layer - пример



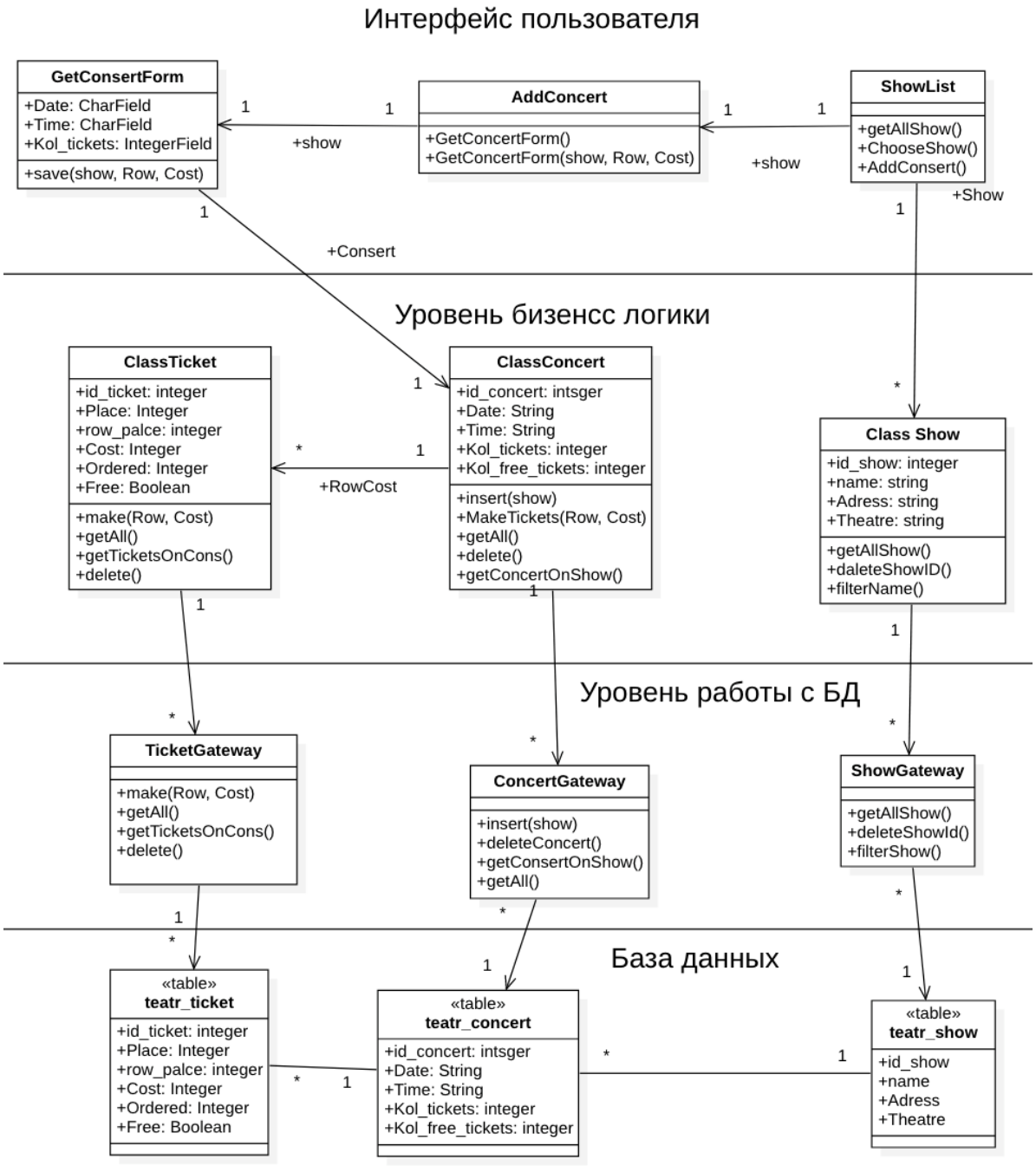
# Service Layer - особенности

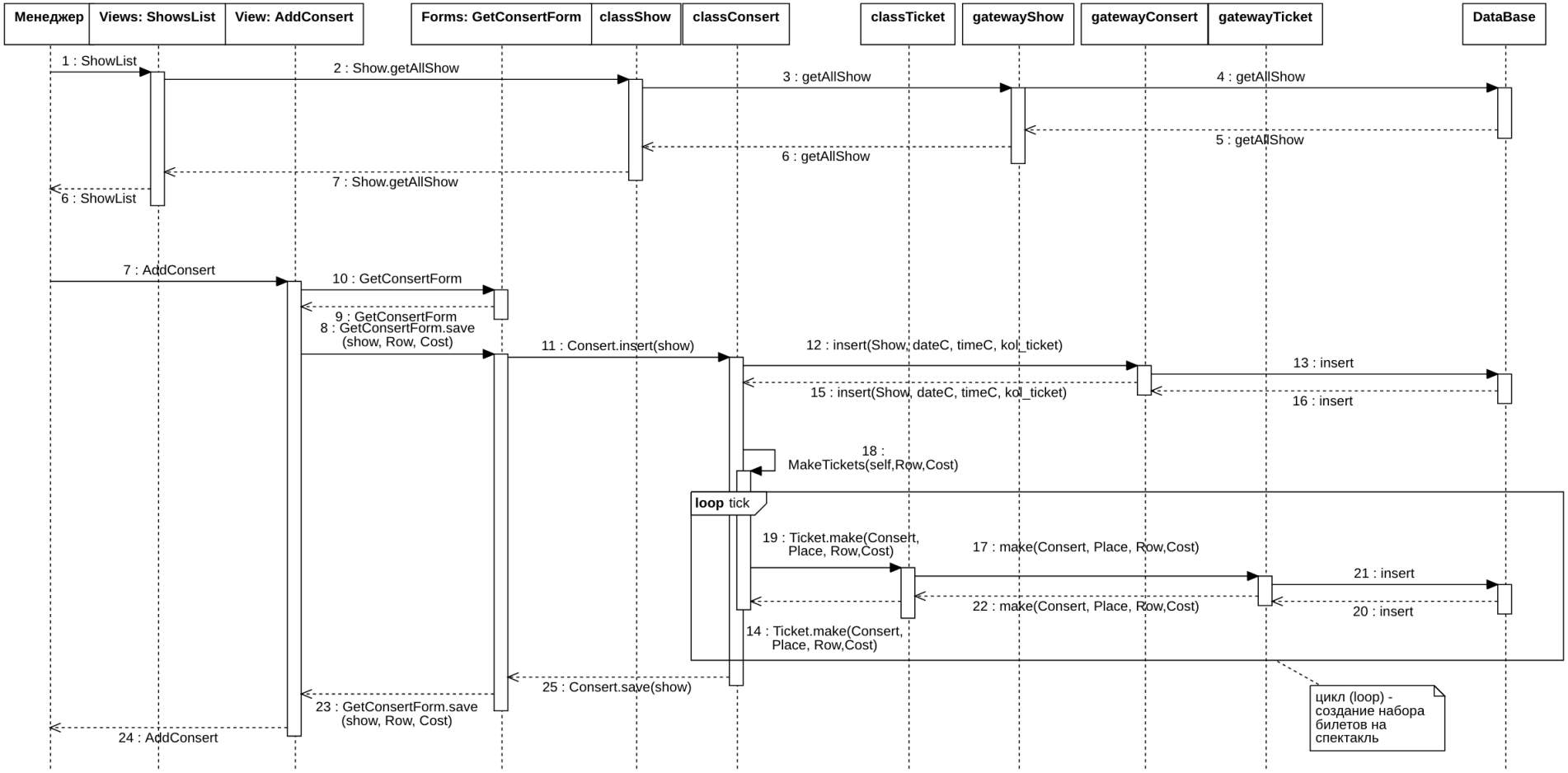
Варианты:

- Интерфейс доступа к домену
  - тонкие интерфейсы поверх модели предметной области/ модуля таблицы
  - набор операций без бизнес логики
- Сценарий операций
  - толстый интерфейс с логикой приложения (не бизнес логикой)
- Реализация транзакций и безопасности

# Пример ДЗ2: Domain Model + Table Data Gateway

- Пример реализован на языке программирования Python с использованием фреймворка Django.
- Предметная область “Бронирование билетов в театр”.
- Нужно разработать программу «Бронирование билетов в театр» и реализовать в ней паттерн бизнес-логики – Domain Model и паттерн работы с базой данных Table Data Gateway.


Интерфейс пользователя
GetConsertForm
+Date: CharField
+Time: CharField
+Kol_tickets: IntegerField
+save(show, Row, Cost)
AddConcert
+GetConcertForm()
+GetConcertForm(show, Row, Cost)
ShowList
+getAllShow()
+ChooseShow()
+AddConsert()
1
1
+show
1
1
+show
1
+Show
1
+Consert
Уровень бизенсс логики
ClassTicket
+id_ticket: integer
+Place: Integer
+row_palce: integer
+Cost: Integer
+Ordered: Integer
+Free: Boolean
+make(Row, Cost)
+getAll()
+getTicketsOnCons()
+delete()
ClassConcert
+id_concert: intsger
+Date: String
+Time: String
+Kol_tickets: integer
+Kol_free_tickets: integer
+insert(show)
+MakeTickets(Row, Cost)
+getAll()
+delete()
+getConcertOnShow()
Class Show
+id_show: integer
+name: string
+Adress: string
+Theatre: string
+getAllShow()
+daleteShowID()
+filterName()
1
*
1
+RowCost
*
1
1
*
Уровень работы с БД
TicketGateway
+make(Row, Cost)
+getAll()
+getTicketsOnCons()
+delete()
ConcertGateway
+insert(show)
+deleteConcert()
+getConsertOnShow()
+getAll()
ShowGateway
+getAllShow()
+deleteShowId()
+filterShow()
*
*
*
*
*
1
*
База данных
«table»
teatr_ticket
+id_ticket: integer
+Place: Integer
+row_palce: integer
+Cost: Integer
+Ordered: Integer
+Free: Boolean
«table»
teatr_concert
+id_concert: intsger
+Date: String
+Time: String
+Kol_tickets: integer
+Kol_free_tickets: integer
«table»
teatr_show
+id_show
+name
+Adress
+Theatre
*
1
*
1
1
1


Менеджер
Views: ShowsList
View: AddConsert
Forms: GetConsertForm
classShow
classConsert
classTicket
gatewayShow
gatewayConsert
gatewayTicket
DataBase
1 : ShowList
2 : Show.getAllShow
3 : getAllShow
4 : getAllShow
5 : getAllShow
6 : getAllShow
7 : Show.getAllShow
6 : ShowList
7 : AddConsert
10 : GetConsertForm
9 : GetConsertForm
8 : GetConsertForm.save
(show, Row, Cost)
11 : Consert.insert(show)
12 : insert(Show, dateC, timeC, kol_ticket)
13 : insert
15 : insert(Show, dateC, timeC, kol_ticket)
16 : insert
18 :
MakeTickets(self,Row,Cost)
loop tick
19 : Ticket.make(Consert,
Place, Row,Cost)
17 : make(Consert, Place, Row,Cost)
21 : insert
22 : make(Consert, Place, Row,Cost)
20 : insert
14 : Ticket.make(Consert,
Place, Row,Cost)
25 : Consert.save(show)
23 : GetConsertForm.save
(show, Row, Cost)
24 : AddConsert
цикл (loop) -
создание набора
билетов на
спектакль