Проектирование программных систем

Технологии разработки программного обеспечения

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

Разработка программных систем

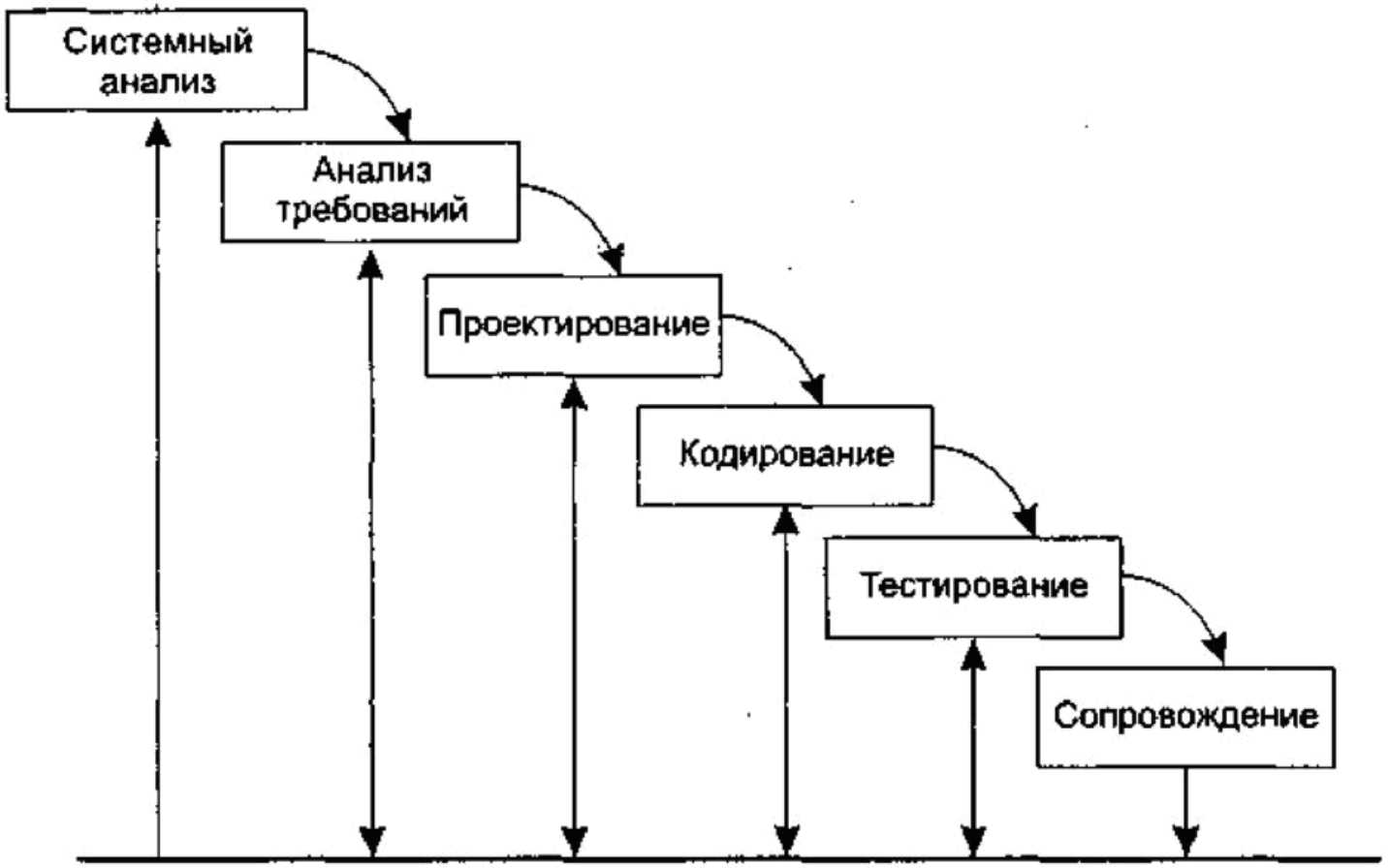
- **Разработка ПО = анализ + синтез + сопровождение**
- Анализ:
  - Определение требований к системе
  - Создание моделей:
    - Информационная (информация для обработки в ПО)
    - Функциональная (перечень функций для обработки)
    - Поведенческая (динамика системы)

- **Синтез = проектирование + кодирование + тестирование**
- Проектирование:
  - На основе требований и моделей определить:
    - Структуры данных (на основе информационной модели)
    - Архитектуру системы (компоненты и их связи)
    - Процедурная разработка (содержание компонентов как последовательности действий)
- Кодирование:
  - Создание текстов модулей

Классический жизненный цикл

Каскадная (водопадная) модель;

Особенность - переход на следующий этап после завершения предыдущего.



Этап ЖЦ каскадной модели

- **Проектирование**
  - архитектура ПО;
  - модульная структура;
  - алгоритмы;
  - структуры данных;
  - входной/выходной интерфейс (формы данных);
  - оценка качества ПО.

=> множество проектных представлений

Проектирование

- Синтез = 75% разработки
- Успех ПО <= хорошее проектирование
- Проектирование обеспечивает перевод требований заказчика в конечное ПО; обеспечивает качество ПО.

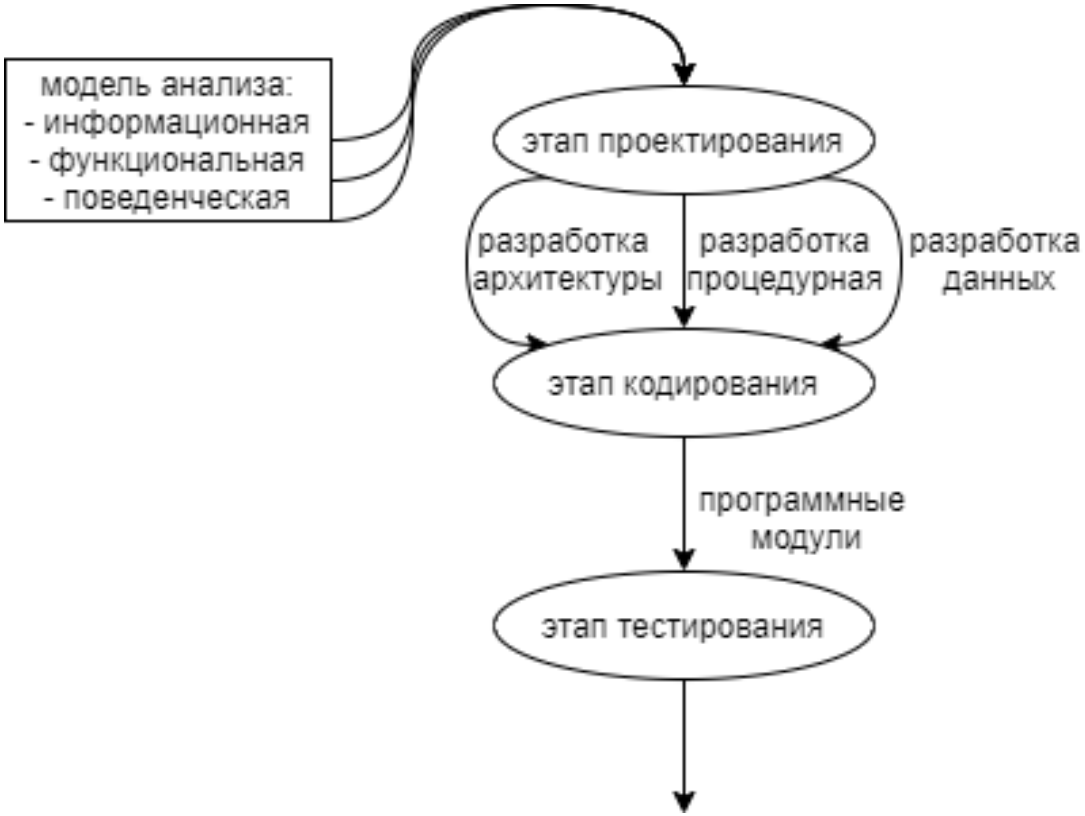
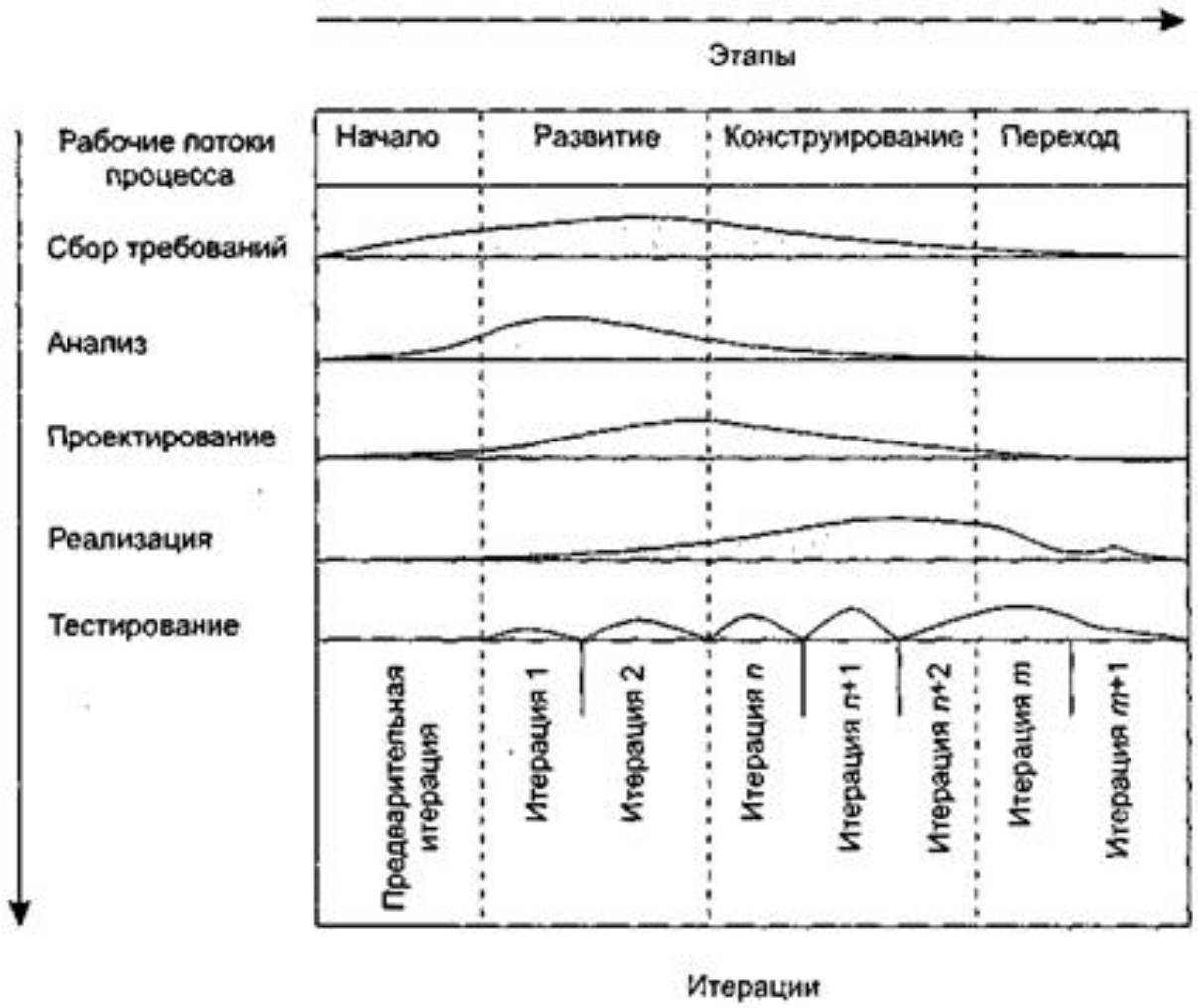


График RUP


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

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.

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 обобщенных механизмов проектирования

- Реализуют требования:
  - длительное хранение;
  - распределение объектов;
  - средства безопасности;
  - контроль и исправление ошибок;
  - управление транзакциями.

- Выбор существующих технологий проектирования и реализации

- Обобщенный механизм проектирования:
  - для хранения -> реляционная база данных, файл, объектно-ориентированная база данных,...
  - распределенная обоботка -> java.rmi, cobra, com,...

- Использование обобщённых кооперацій - образцов, паттернов

- Поставляемое ПО -> средний уровень и уровень СПО
  Создаемое ПО -> прикладной уровень

Методологии проектирования

- **Структурная** (функциональная) декомпозиция:
  Структура ПО как иерархия функций и передача информации по иерархии (вызов)
- **Объектная** декомпозиция:
  Структура объектов и связи между ними;
  поведение ПО как обмен сообщений между объектами (синхронные и асинхронные).

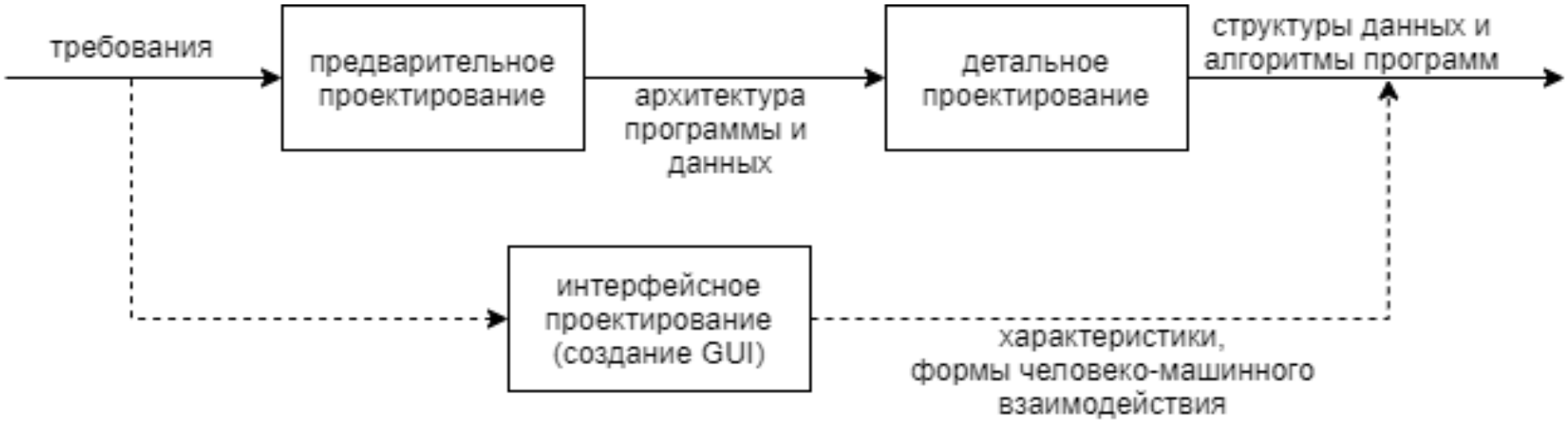
- Проектирование – итерационный процесс;
- реализует переход от требований к инженерному представлению ПО:
  - 1-ый уровень проектирования – высокий уровень абстракции
  - N-ый уровень – близки к текстам на языке программы

Виды структурного проектирования

- Нисходящее:
  - Пошаговое уточнение (декомпозиция с определением модулей нижнего уровня)
  - Анализ сообщений (анализ потоков данных, обработанных модулями)
- Восходящее
  - Определение функций и их соединение друг с другом
  - Определение вспомогательных модулей в первую очередь
- Метод расширения ядра
  - Выявление множества вспомогательных модулей вместо функций всей программы в целом

Этапы проектирования

- **Предварительное** проектирование
  - Идентификация подсистем
  - Определение принципов взаимодействия подсистем и управления ими
- **Детальное** проектирование
  - проектирование модулей



Этапы предварительного проектирование

- Структурирование системы
  - Выделение подсистем (подсистема как независимый компонент)
  - Определение взаимодействия подсистем
- Моделирование управления
  - Определение модели связей управления между частями системы
- Декомпозиция подсистем на модули
  - По любой подсистеме – декомпозиция на модули
  - Определение типов модулей и межмодульные соединения

Структурирование системы

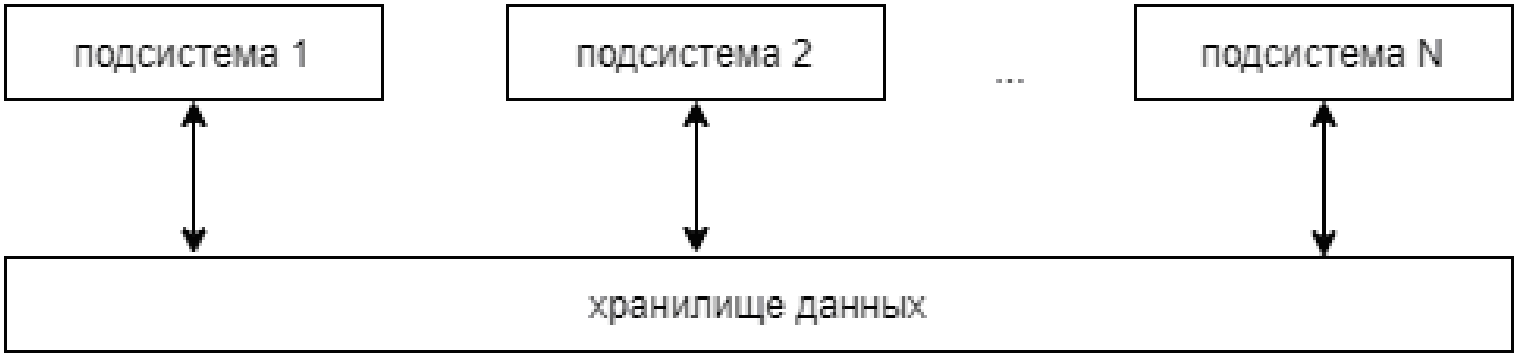
- Выбор/разработка модели архитектуры
- Влияет на
  - производительность и характеристики системы
- Определяет:
  - Компоненты
  - Связи компонентов
  - Характер интерфейсов системы;
  - роли и ответственности подсистем

Модели архитектуры

- Хранилище данных
- Клиент-сервер
- Трехуровневая
- Многоуровневая
- Одноранговая
- Каналы и фильтры
- Компонентная
- Каркасы

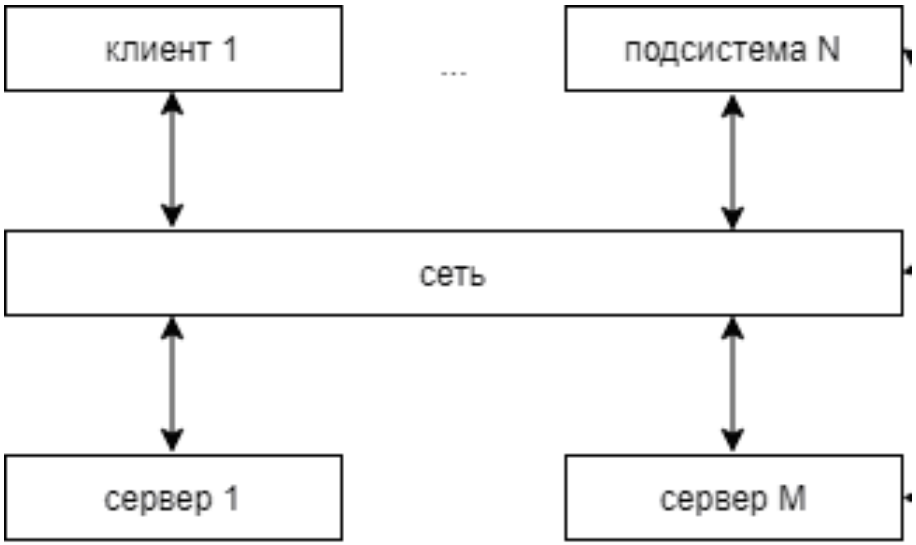
Хранилище данных

– Общие данные
– п/с работают с общим хранилищем



Клиент-серверная архитектура

- Клиент
  - выдает запросы
- Сервер
  - ждет запроса;
  - предоставляет сервис, данные или ресурсы
- Сеть
  - протокол взаимодействия (например, TCP|IP)



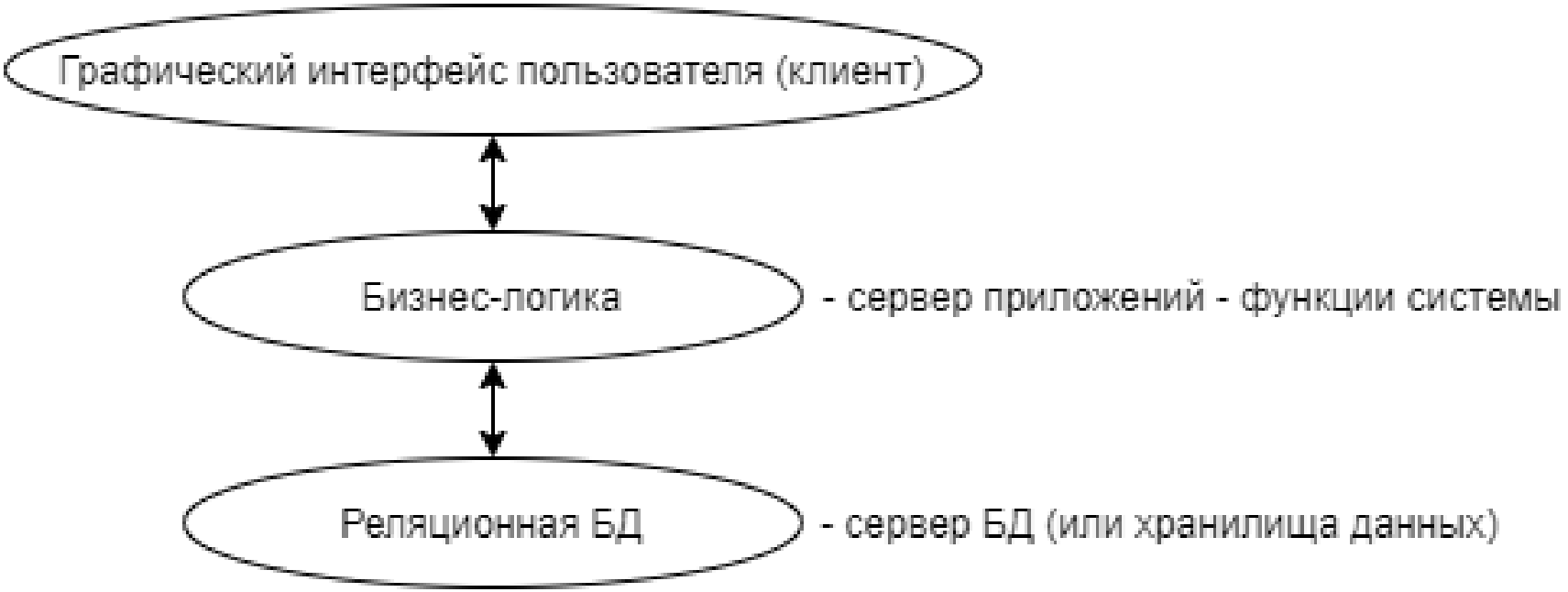
Двухуровневая архитектура

- Клиент
  - использует сервисы сервера,
  - посылает запросы
  - обрабатывает результаты запроса
- 1 сервер – N клиентов (терминал, web-интерфейс, GUI)

- Способы связи:
  - Стандартные сетевые протоколы
  - Удаленный вызов процедур (RPC)
  - Удаленные запросы SQL
  - Собственные протоколы
- Разделение бизнес-логики – толстый/тонкий клиент
- Недостаток:
  - система зависит от работоспособности сервера
  - необходимо администрирование для бесперебойной работы

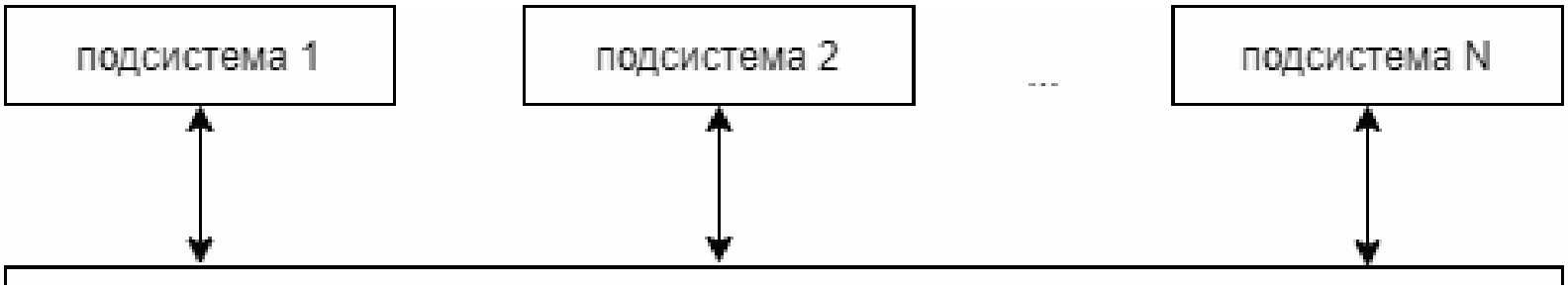
Трехуровневая архитектура

- упрощение модификации уровней
- возрастает оптимизация системы за счет разделения функций БД и прикладных функций

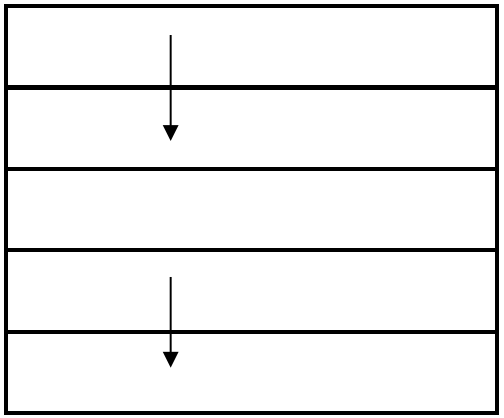


Одноранговая архитектура

- Все сетевые узлы равны по значимости или возможностям
- Преимущество: хорошая устойчивость к отказам



Многоуровневая архитектура



пользовательский интерфейс

.

.

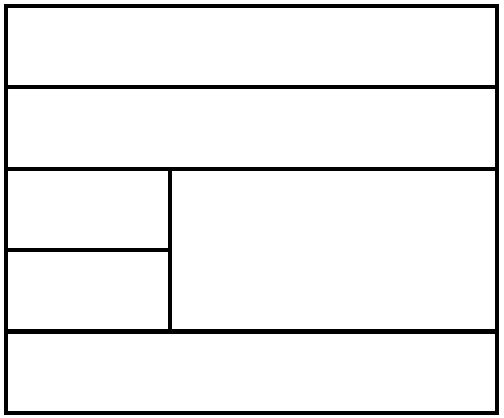
.

.

.

интерфейс аппаратуры / ОС / CORBA

Вариации:



- 2 уровня

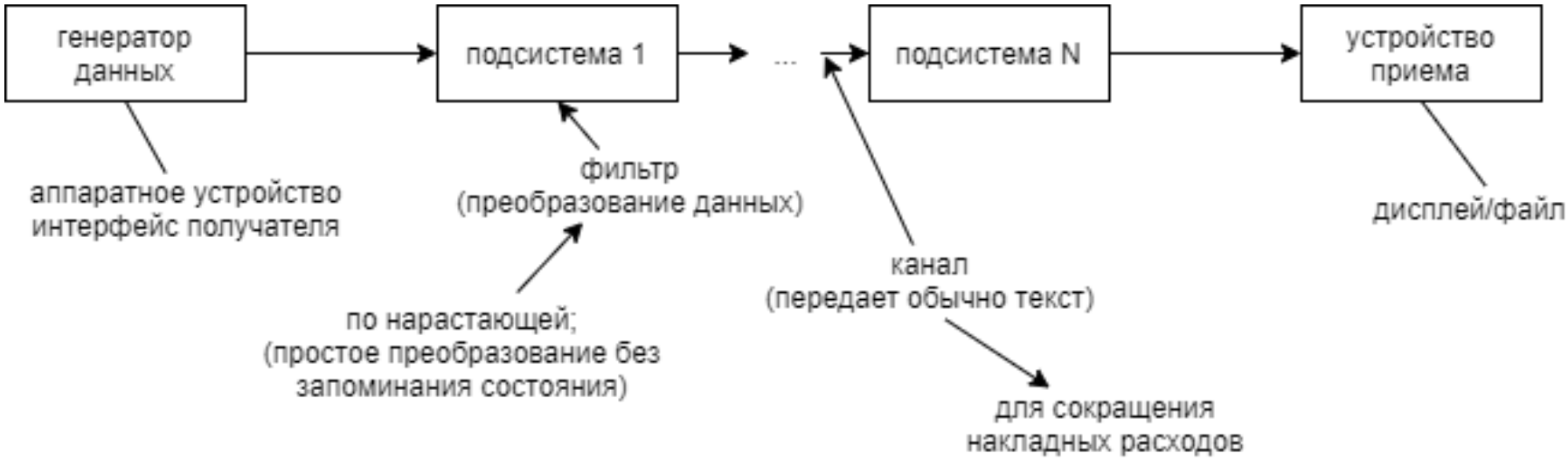
Многоуровневая архитектура

- Текущий слой использует интерфейс нижнего слоя
- Нельзя использовать возможности высшего уровня

- Преимущество:
  - можно заменить уровень без изменения остальных (ОС; язык реализации)

- Не определено:
  - Возможность использовать интерфейсы «вниз через уровень»
  - Возможность использовать интерфейсы «соседнего блока»

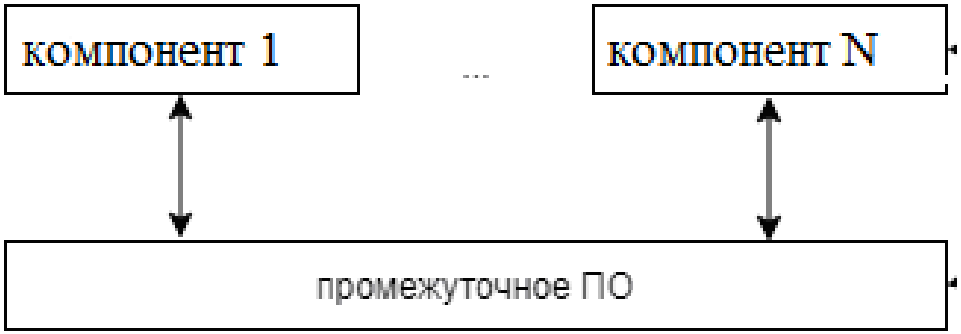
Архитектура с каналами и фильтрами



Компонентная архитектура

- Объектно-ориентированный подход (COM / CORBA / JBeans)

открытый интерфейс IDL
- независима
- реализация скрыта
- могут быть локальные/открытые данные



-инфраструктура коммуникаций (.NET)
- подключение
- оповещение о себе
-объявление услуг

Каркасы (Framework)

- Расширяемая библиотека кода (группа связных классов) для эффективного проектного решения в конкретной области
  - Часть работы сделана
  - Есть пустые поля для расширения
  - Различные архитектурные модели

- Библиотеки:
  - вызов существующих компонентов в своем потоке управления
- Каркас:
  - имеет структуру и поток управления;
  - может вызывать новые функции

Виды интерфейсов

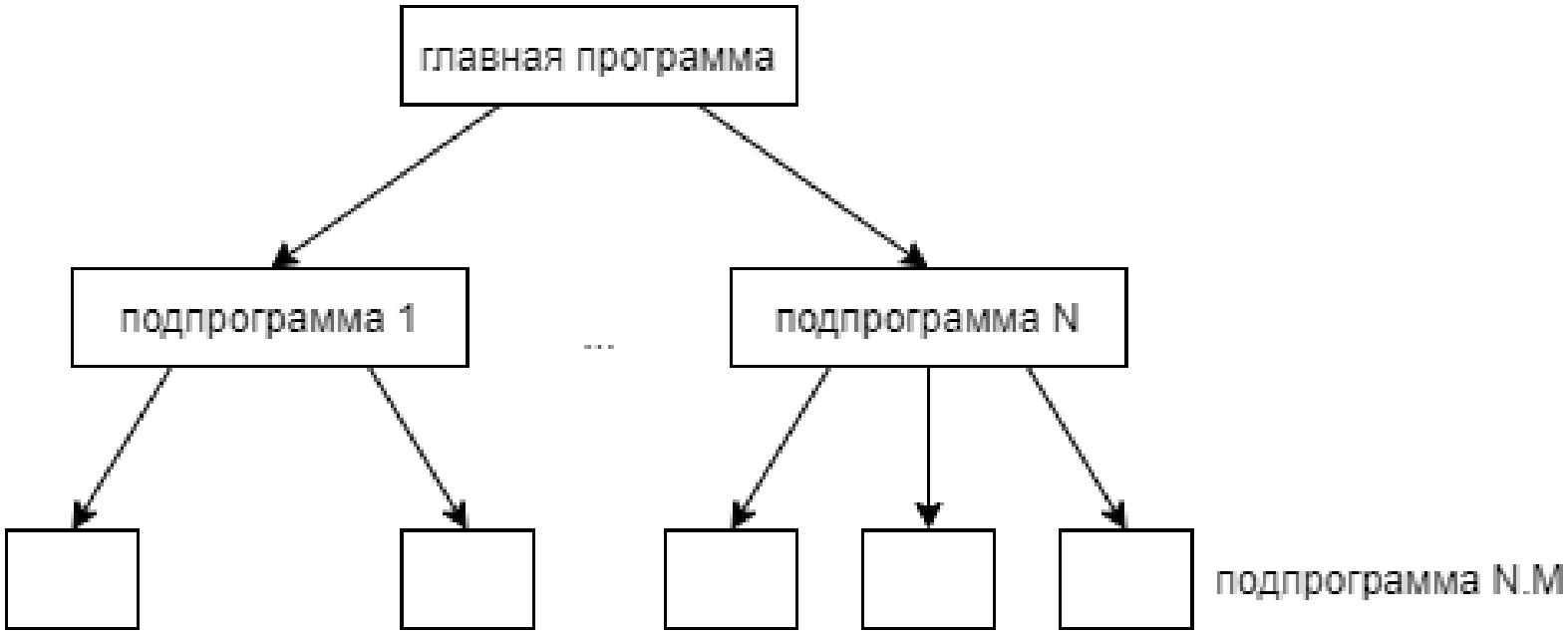
- Точки соединения и границы компонентов
- Внешний вид и доступ

- **API -** интерфейсы прикладного программирования;
  - совокупность функций в физическом компоненте
    Для замены компонента реализовать все функции и заново собрать
- **Иерархия классов**
  - интерфейс сохраняется в производных классах (реализация абстрактных классов)
- **Компонентные технологии** (COM/CORBA)
  - Реализация выбирается при выполнении
  - IDL – язык определения интерфейса (абстрактный)
  - Реализация на любом языке
  - Необходима поддержка со стороны промежуточного ПО или ОС
- **Форматы данных**
  - точки соединения в проектах, ориентированных на перемещение данных, а не потока управления
  - Компонент заменяется аналогичным, работающим с теми же типами данных

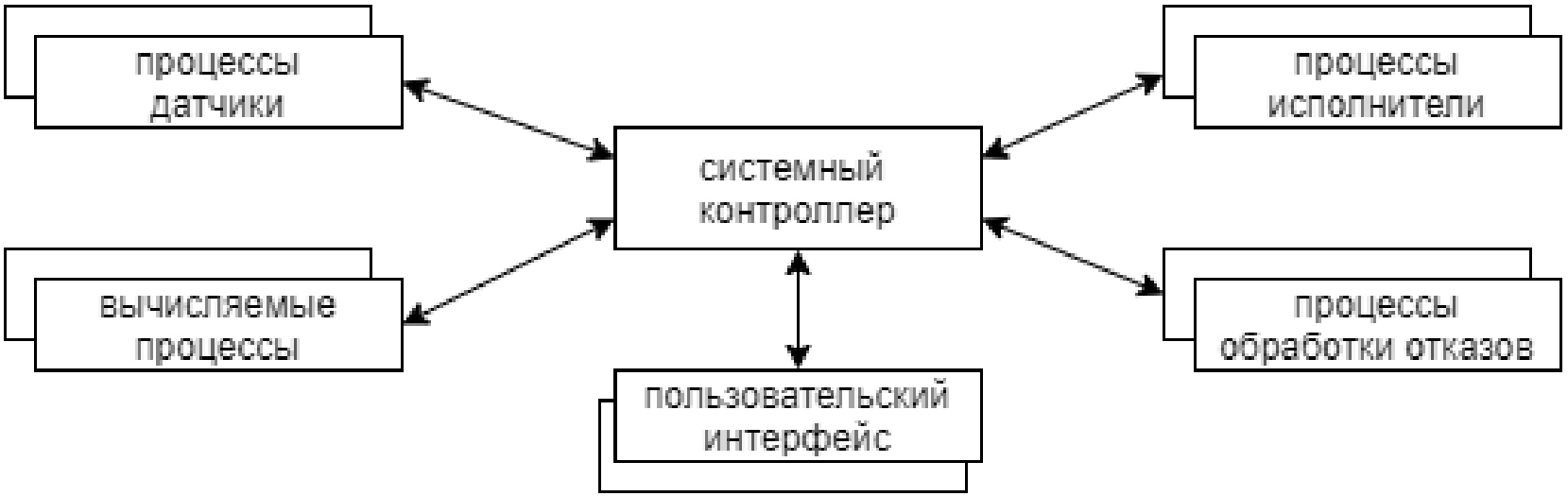
Моделирование управления

- Централизованное управление – одна из подсистем является системным контроллером и управляет другими подсистемами
  - Модель вызов-возврат
  - Модель менеджера (в системах параллельной обработки)
- Модель событийного управления – внешние события управляют системой
  - Широковещательная модель
  - Модель, управляемая прерываниями

Вызов-возврат



Модель менеджера

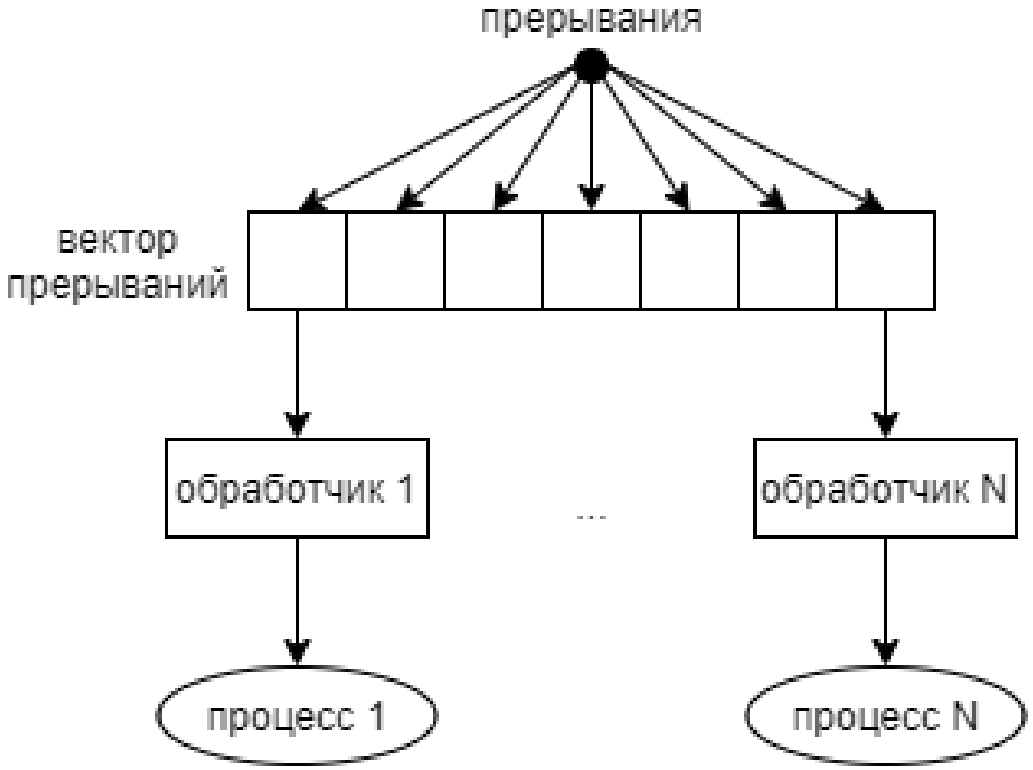


Широковещательная модель

- Подсистема – сообщает обработчику о «ее» событиях
- Обработчик – если было событие, то пересылает его подсистеме. Функций управления не имеет.

Модель, управляемая прерываниями

- Для любого типа (группы) прерываний – свой обработчик
- Любой обработчик реагирует на свой тип прерывания и запускает свой процесс



Структурный проект

- **Структурный проект** – множество решений для создания эффективной структуры программного обеспечения (ПО) с обоснованием решений.

Особенности:

- Касается нефункциональных требований;
- Создаёт фундаментальные решения верхнего уровня;
- Определяет зависимости и компромиссы;
- Обеспечивает формирование и оценку альтернативных решений.
- Структура ПО – организация компонентов ПО в систему для достижения некой цели.

Диаграмма развертывания – компоненты и связи между ними

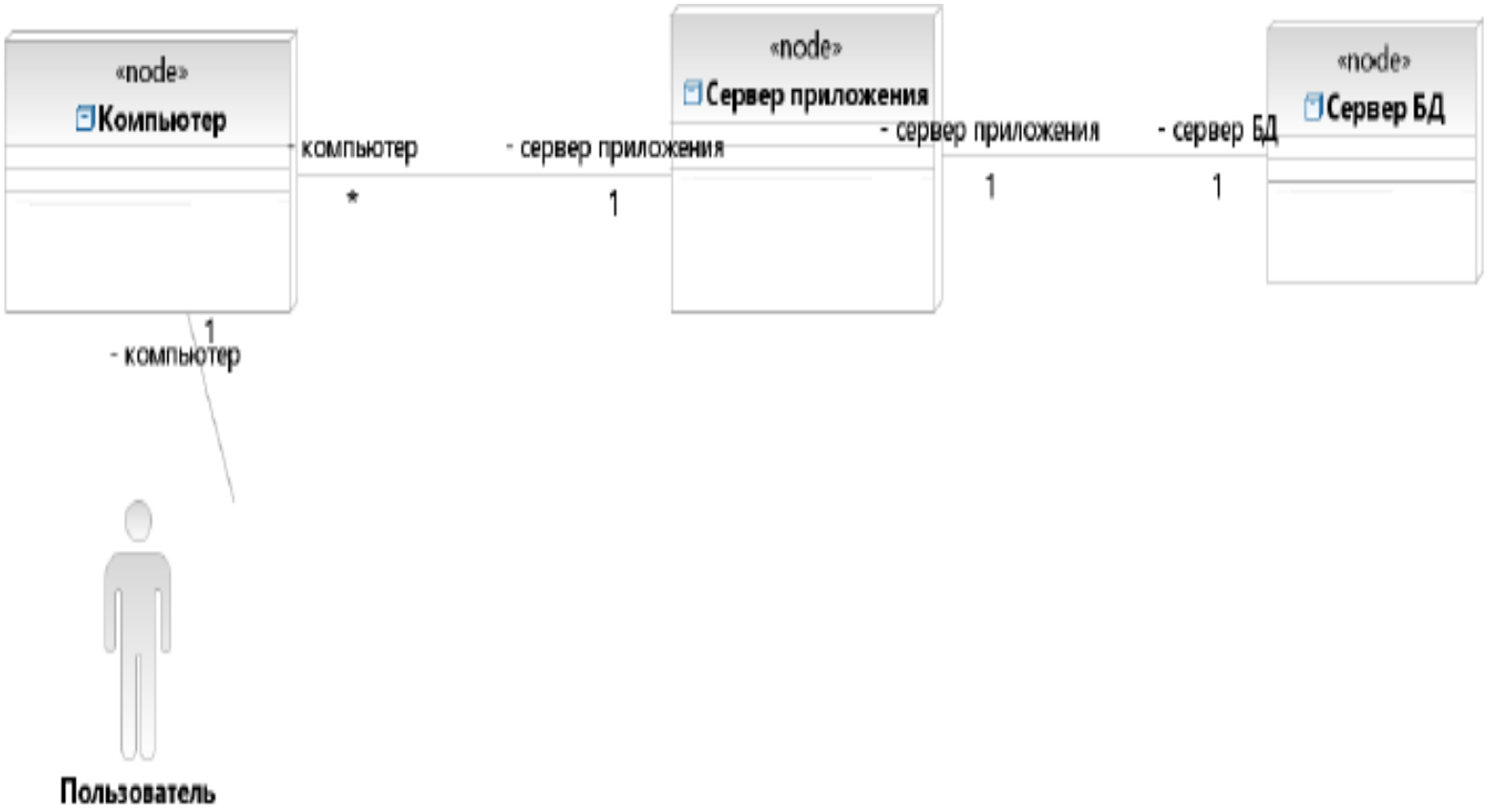
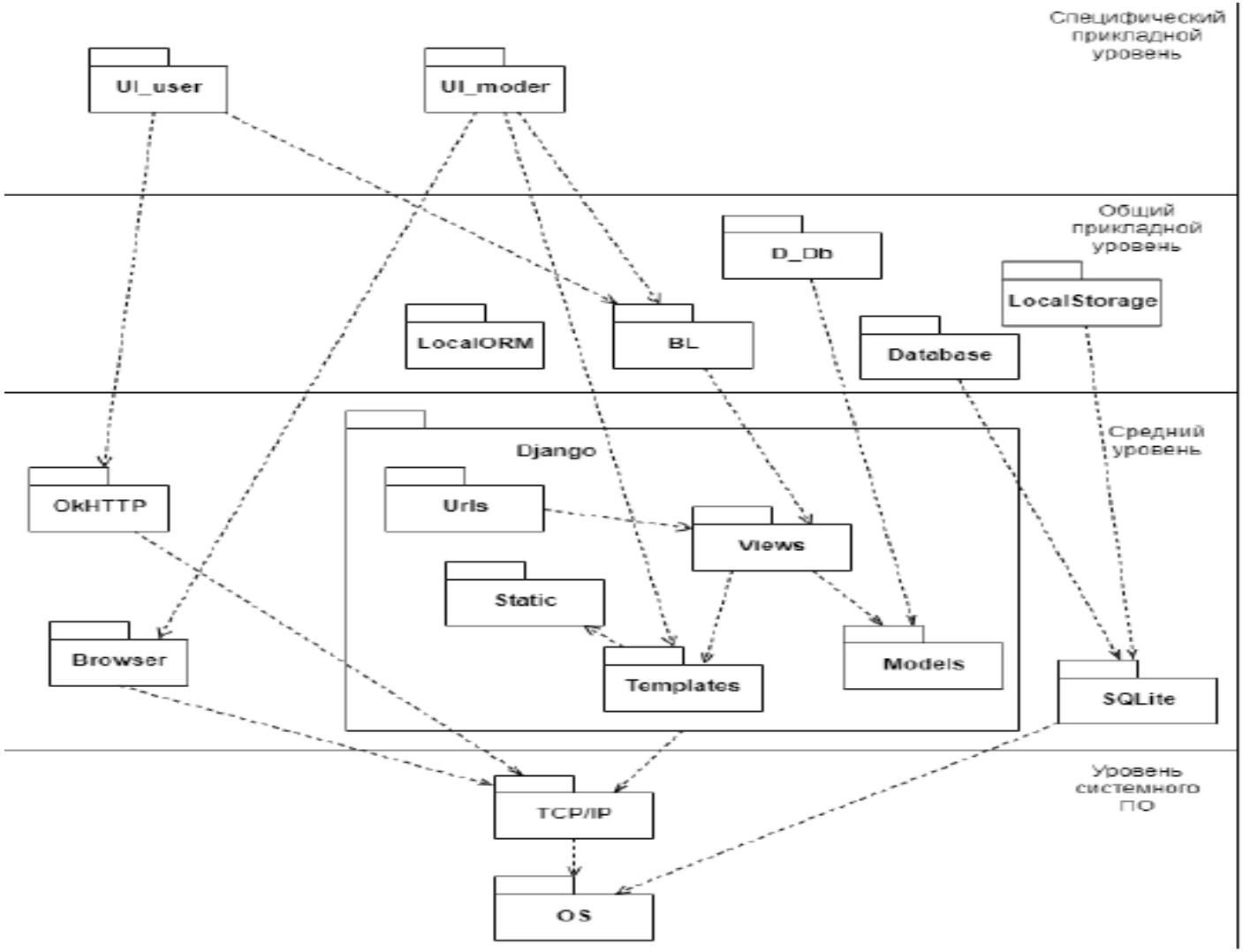
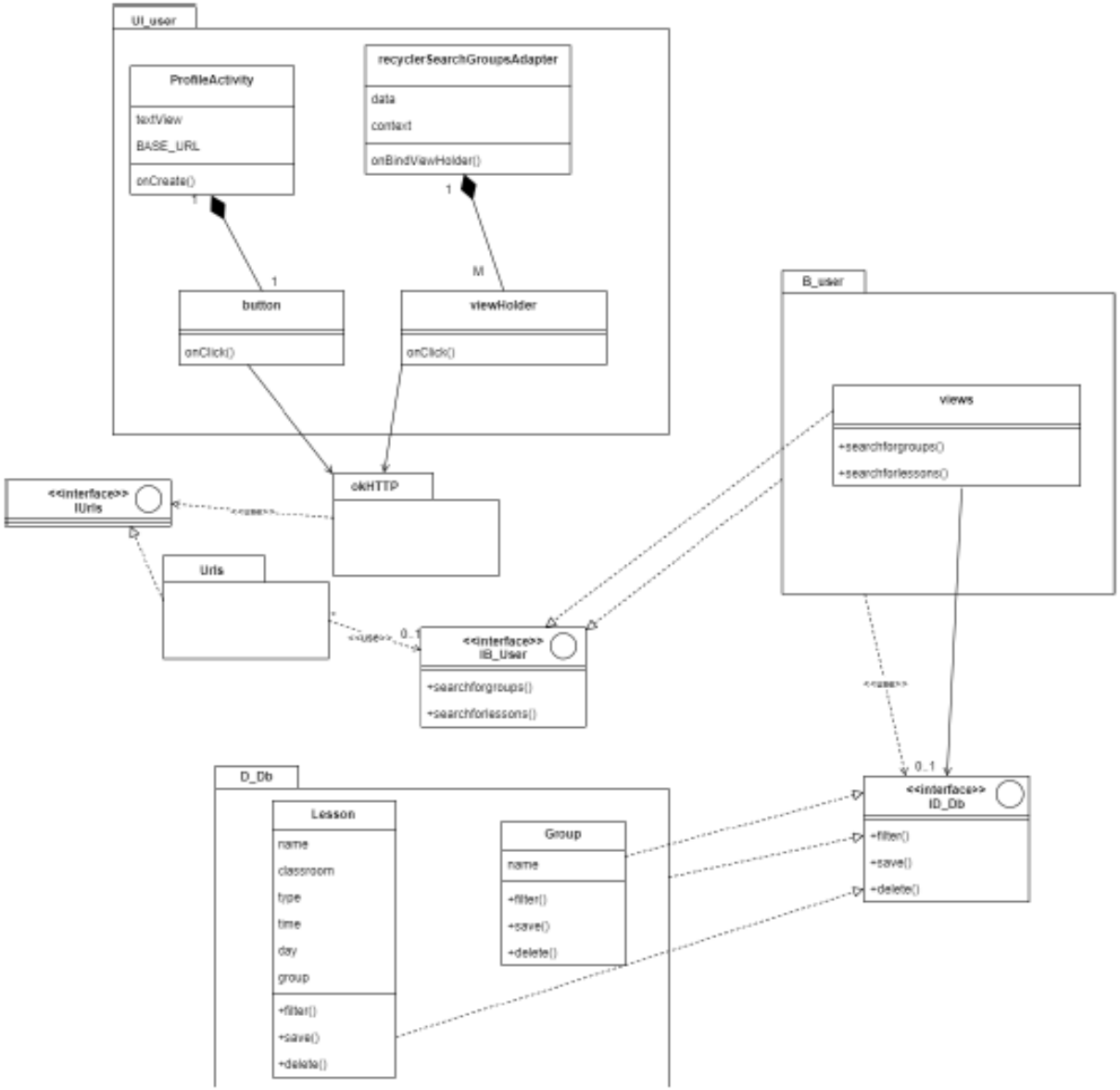


Диаграмма уровней подсистем – структура системы



Распределение классов по подсистемам – структура подсистем



Процесс структурного проектирования

- Методология построения алгоритмов, программ и систем, в основе которой лежит выявление структуры задачи, определение составляющих компонентов и выделение связей между ними.
- Происходит итерационно и пошагово;
- Управляется прецедентами;
- Сконцентрирован вокруг структуры.

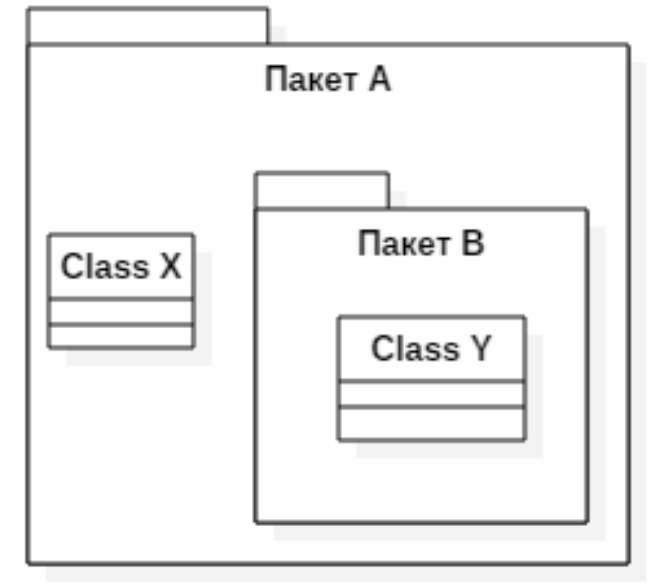
- **Структурный проект как результат:**
  - Иерархическое выделение уровней модулей и определение связей модулей:
    - Понижается сложность;
    - Понижается сцепление.
  - Использование стандартов разработки:
    - Повышается видимость зависимостей;
    - Понижается сложность;
    - Понижается сцепление.
  - Модель / шаблон уровней
  - Шаблоны проектирования

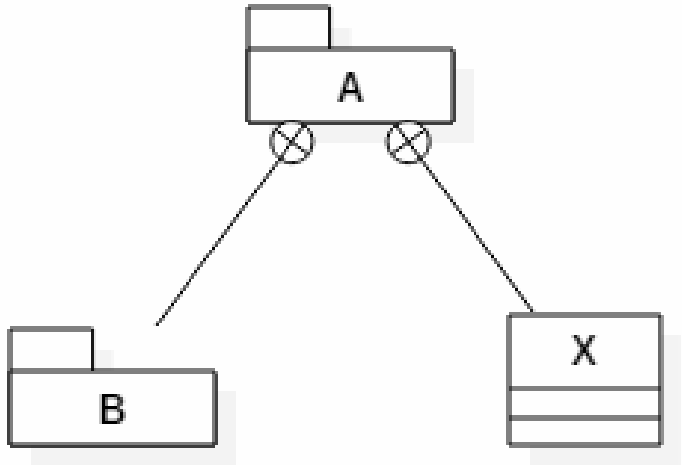
Задачи проектирования

- **Управление зависимостями**
  - Предположим, что существуют два пакета (A и B). Если изменение в пакет A приводит к изменению в пакете B, то B зависит от A.
- **Прямое проектирование**
  - Прямым проектированием (Forward engineering) называется процесс преобразования модели в код путем отображения на некоторый язык реализации.
  - На этапе проектирования количество зависимостей приводится к минимуму, затем происходит процесс реализации.
- **Обратное проектирование**
  - Обратным проектированием (Reverse engineering) называется процесс преобразования в модель кода, записанного на каком-либо языке программирования.
  - Полученный на этапе реализации код используется для построения модели проекта. Происходит оценка количества зависимостей, затем происходит улучшение существующих зависимостей.

Структурные модули

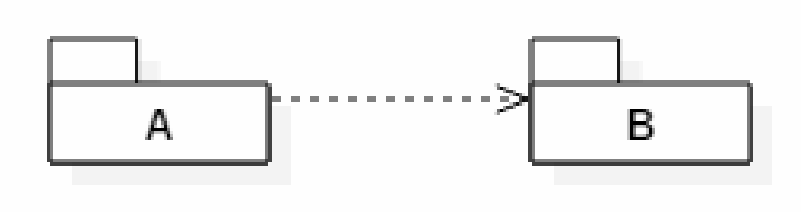
- **Класс** – описание множества объектов, которые разделяют одинаковые свойства, операции, отношения и семантику (смысл).
  - Классы анализа преобразуются (трассируются) в классы проектирования, затем в классы реализации.
- **Пакет** – общий механизм для распределения элементов по группам. В пакет могут помещаться структурные предметы (например, классы или другие пакеты).
  - Пакеты на этапе проектирования трассируются в подсистемы, затем на этапе реализации трассируются в компоненты.
- Примеры вложенности пакетов и классов:





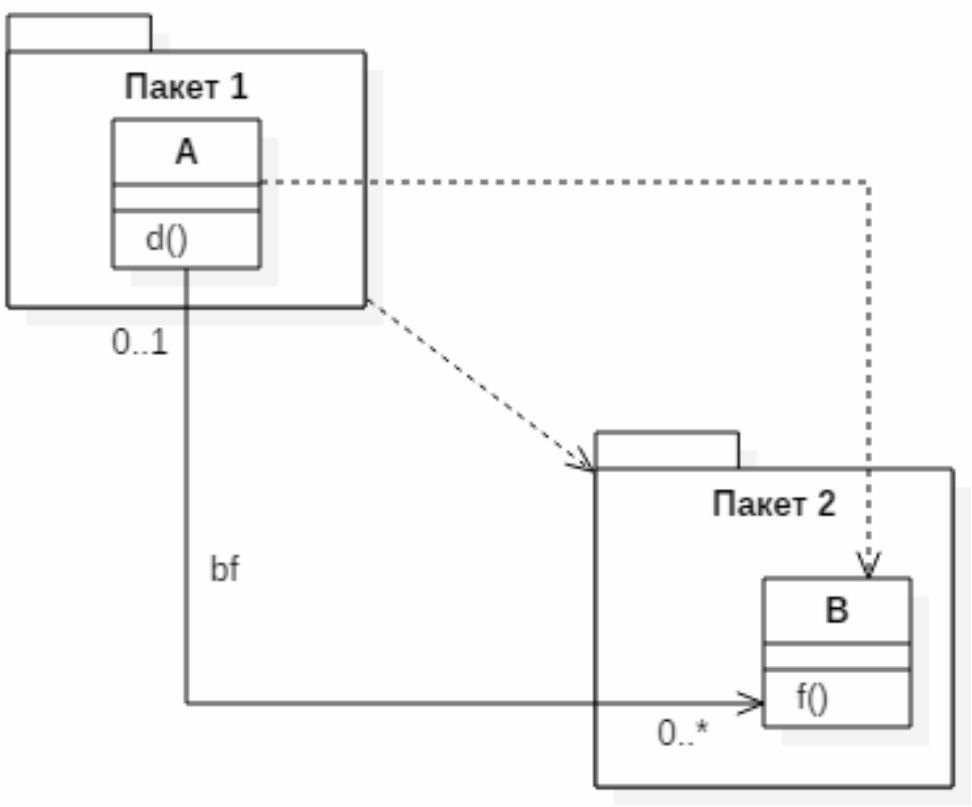
Зависимости

- **С**емантическое отношение между двумя предметами, в котором изменение в одном предмете (независимом предмете) может влиять на семантику другого предмета (зависимого предмета).
- Главная цель разработки программного обеспечения – свести количество зависимостей к минимуму. Ненужные зависимости должны быть устранены из структурного проекта.
- Причины зависимости:
  - Импорт – A (часть A) обращается к B (части B);
  - Изменение B приводит к изменению A (перекомпиляция);
  - A может быть использован только с B (повторно).



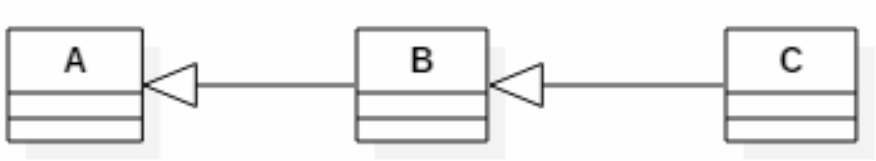
Возникновение зависимости

- Зависимости методов и вытекающие из этого зависимости классов и пакетов (Bf – имя переменной).

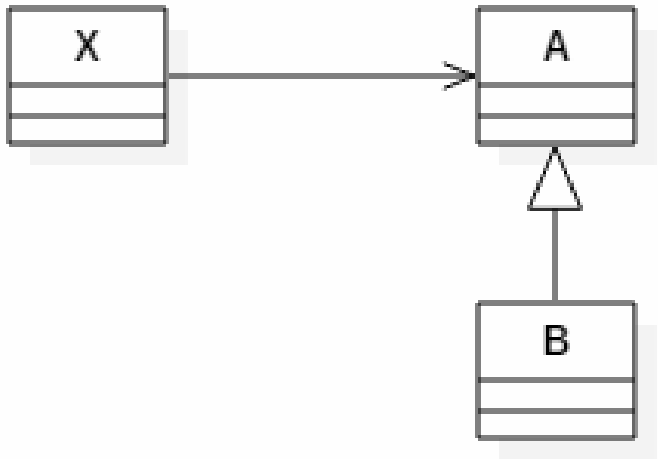


Примеры зависимостей

1. Зависимости наследования времени компиляции

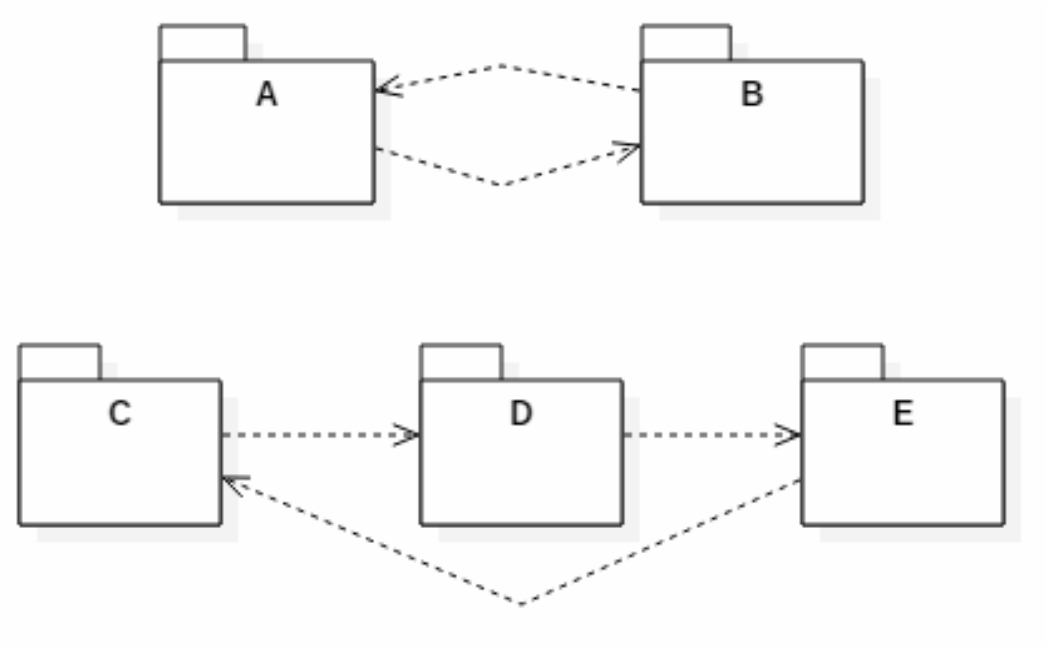


2. Зависимости наследования времени выполнения

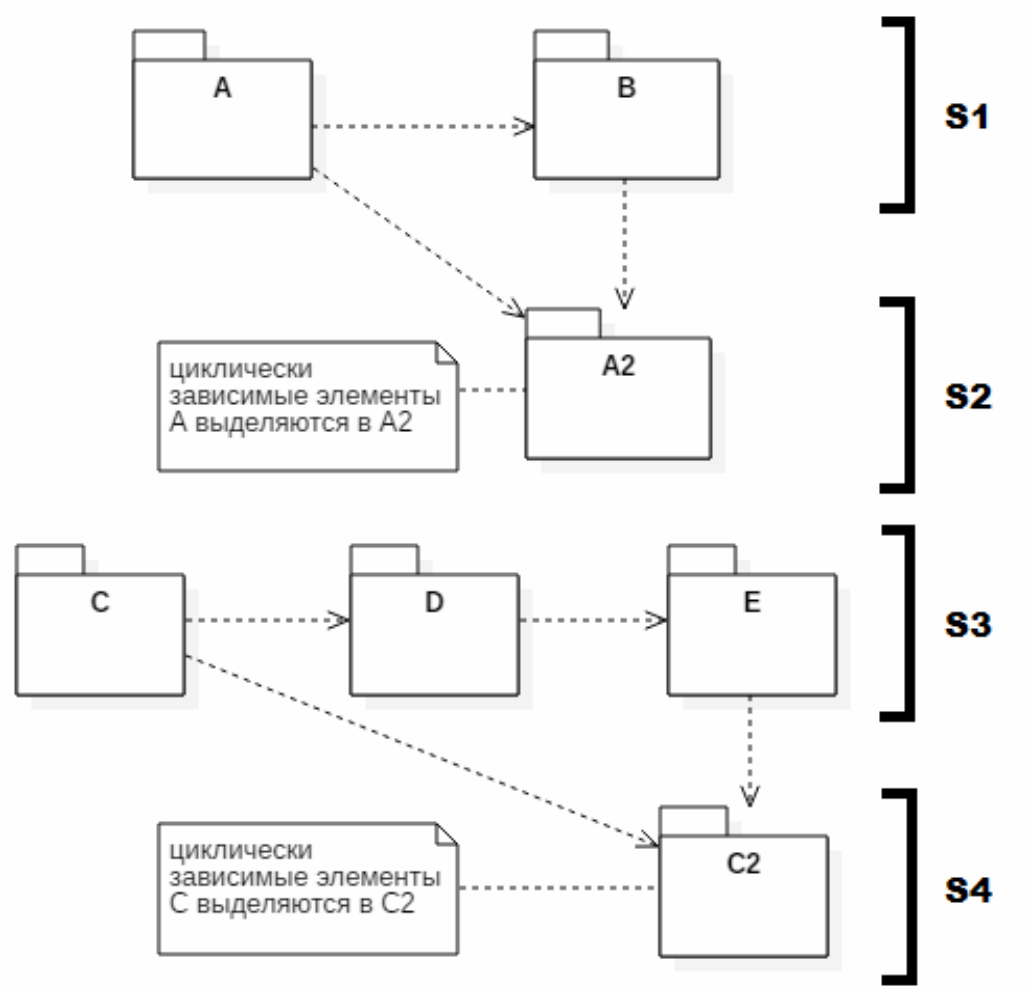


Циклическая зависимость

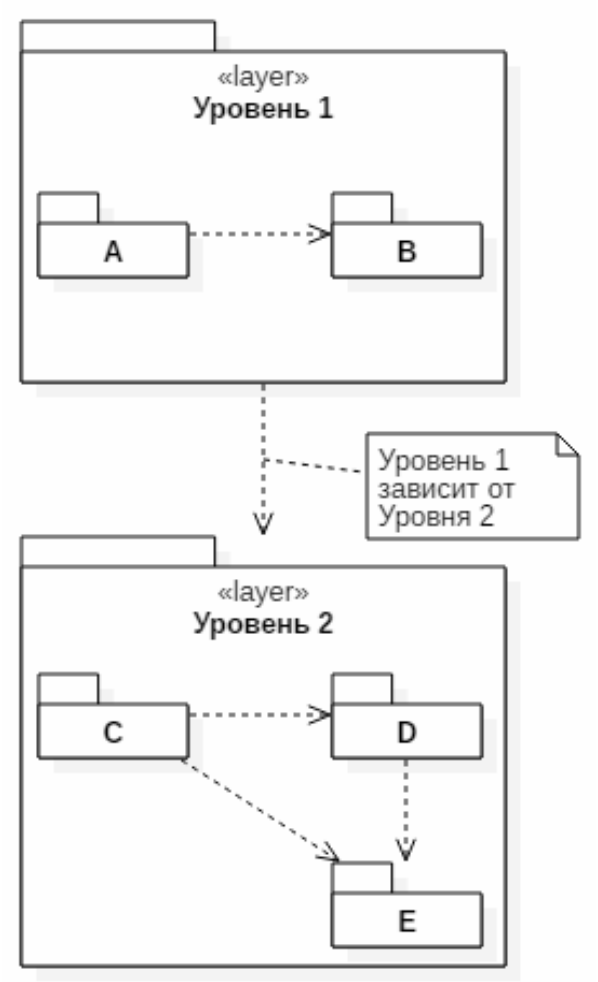
- Циклическую зависимость между пакетами следует устранять при проектировании



Устранение циклических зависимостей



Уровни пакетов



- Пакеты могут быть сгруппированы и структурированы в иерархические уровни, подходящие для выбранной структуры ПО.
- Уровень сам является пакетом;как пакет может содержать другие пакеты.

Зависимости уровней пакетов

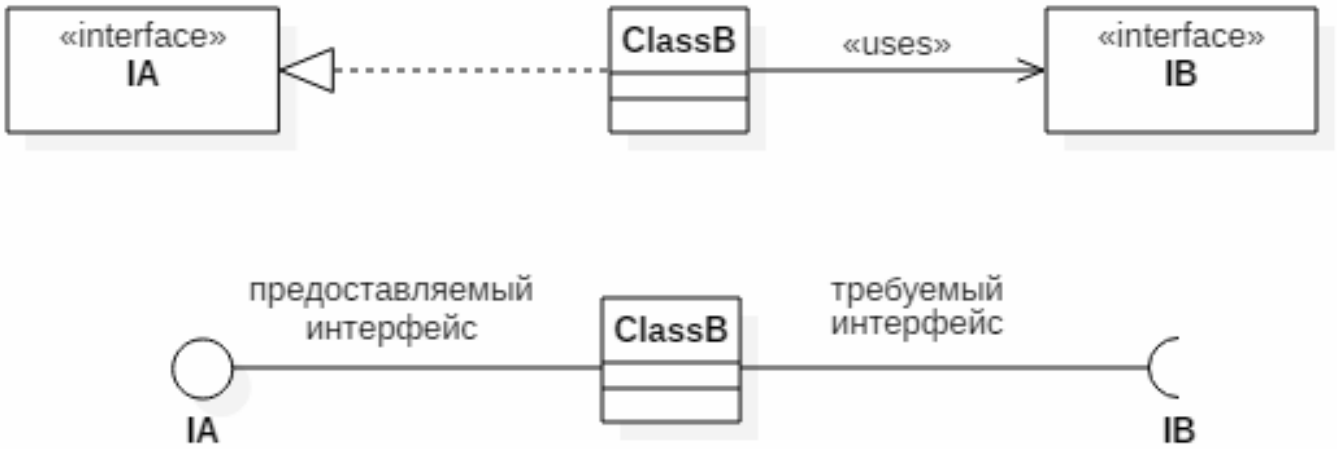
- **Отличительные особенности уровней:**
  - Сетевая структура заменена иерархией;
  - Иерархия уровней минимизирует зависимости между пакетами;
  - Иерархия уровней устанавливает стабильный шаблон для жизненного цикла разработки ПО (устойчивость к изменениям).
- Более высокие уровни зависят от более низких уровней.
- Поэтому зависимости в нисходящем направлении являются сильными (сильная связь), а в восходящем направлении обеспечивается слабая зависимость (слабая связь).

Интерфейсы

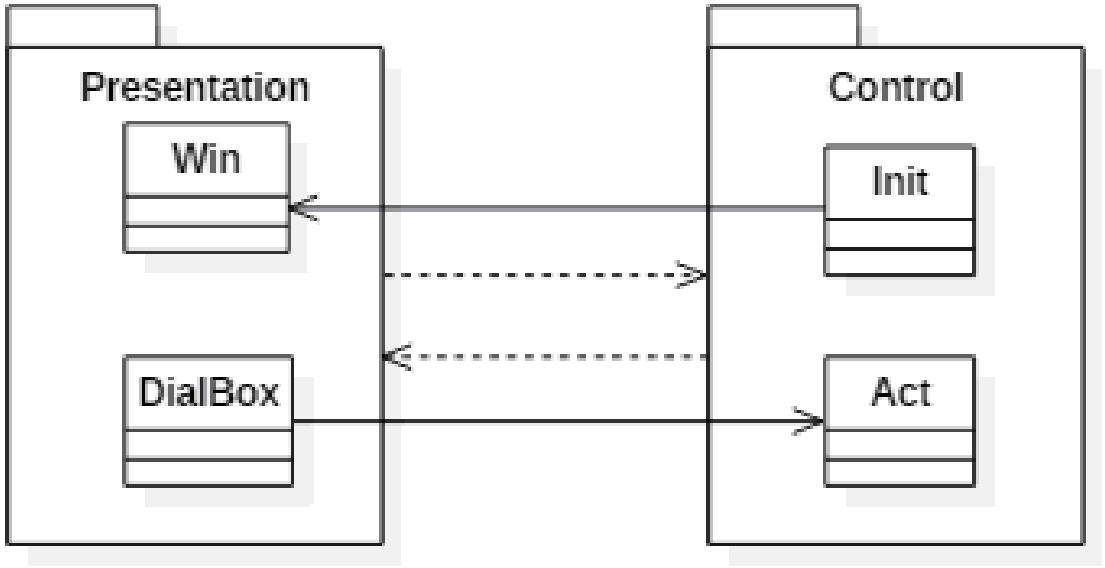
- **Интерфейс** – это объявление ряда особенностей, которые непосредственно не проявляются, то есть, никакие их объекты не могут быть непосредственно созданы.
- Интерфейс предоставляет:
  - Набор операций, видимых извне;
  - Набор атрибутов (в UML 2.0).
- Интерфейс может быть параметром метода, тогда происходит подстановка во время выполнения.
- **Доминирующий класс** – класс, который реализует главные интерфейсы и абстрактные классы в пакете.

Зависимости реализации и использования

- Предоставленные интерфейсы определяются в UML 2.0 отношением зависимости между классом и интерфейсом, реализованным данным классом. Это называется **зависимостью реализации**.
- Требуемые интерфейсы определены в UML 2.0 отношением зависимости между классом (или интерфейсом) и требуемым интерфейсом. Это называется **зависимостью использования**.



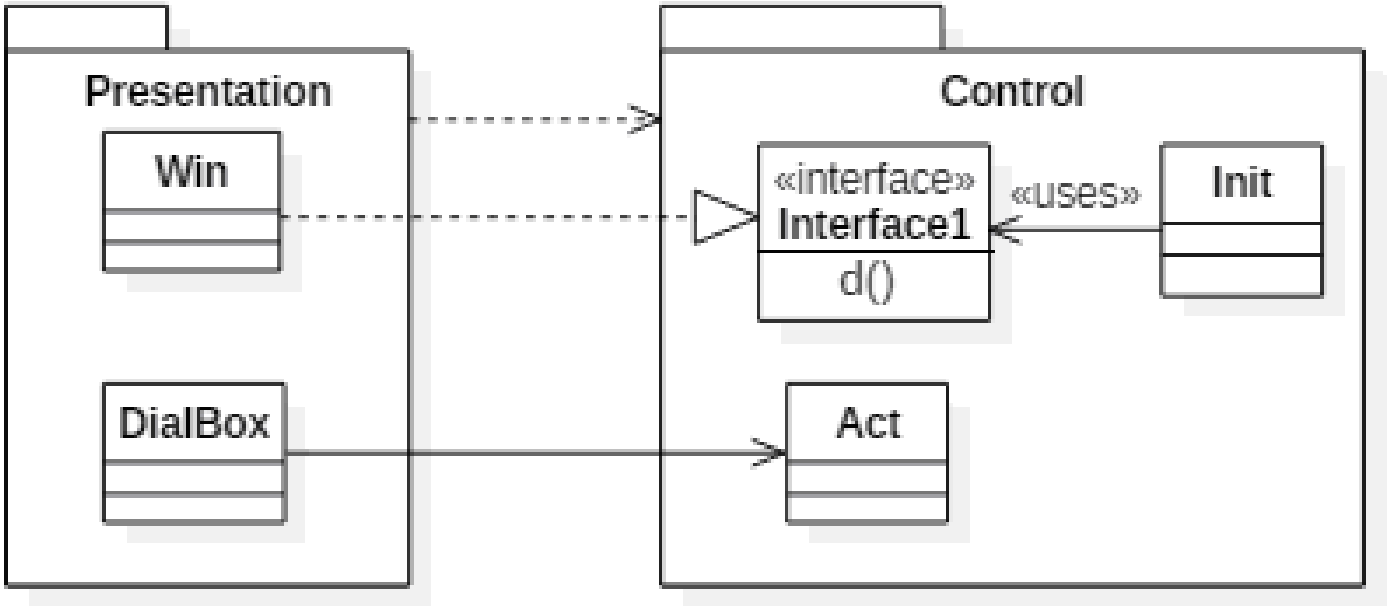
Устранение циклической зависимости через интерфейс



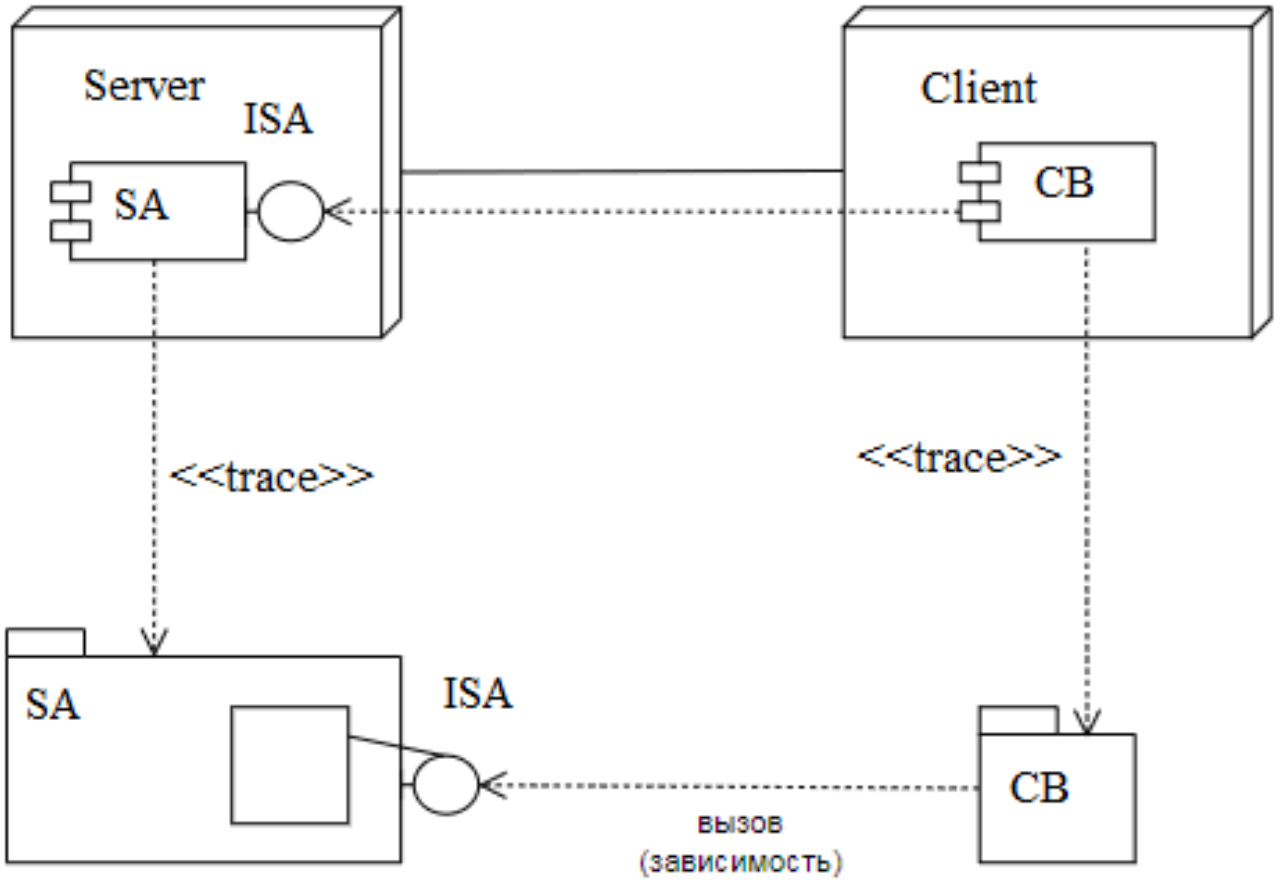
- Для того чтобы нарушить цикл, интерфейс и класс, который реализует его, должны находиться в различных пакетах.
- Либо стоит поместить интерфейс в отдельный пакет.

«Отделенный интерфейс»

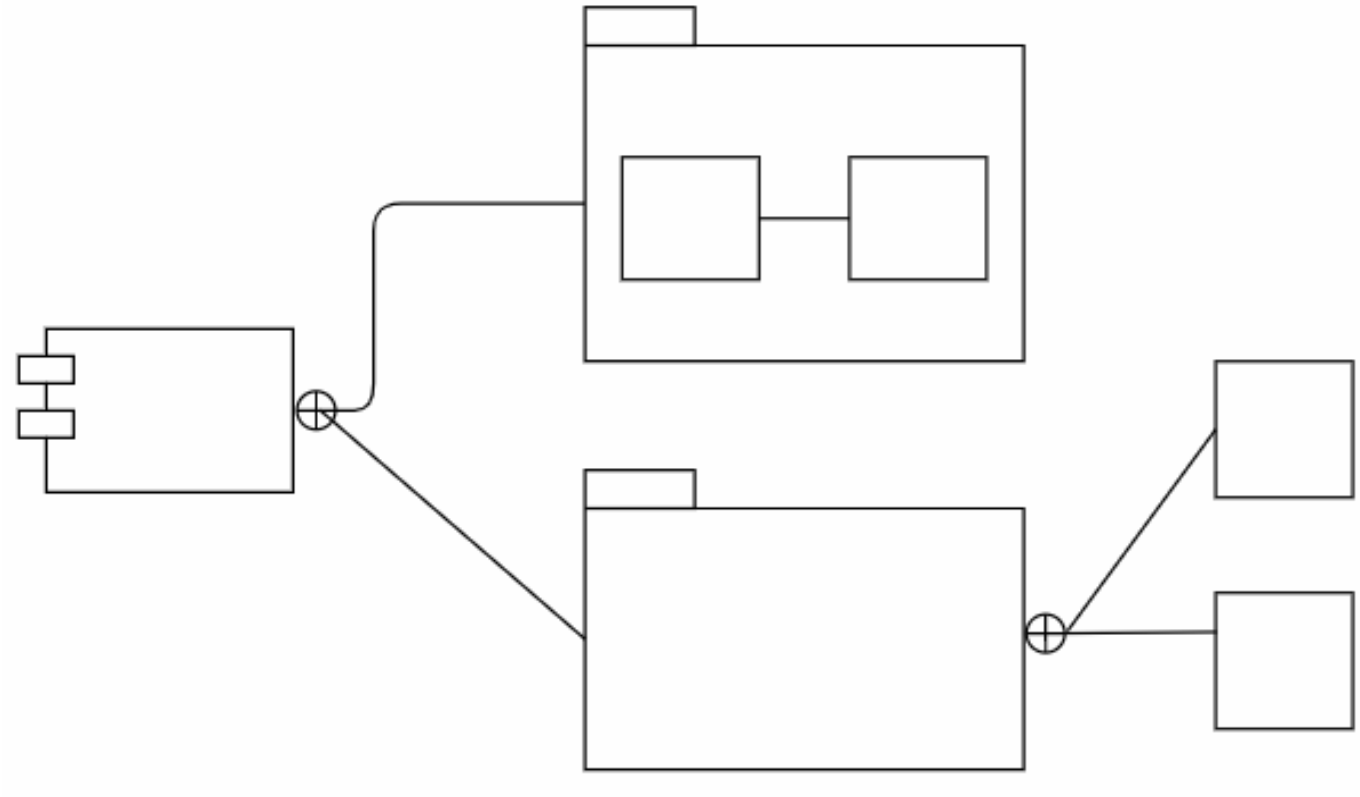
- – паттерн проектирования Фаулера



Зависимости на диаграмме развертывания



Вложенность элементов через отношение включения

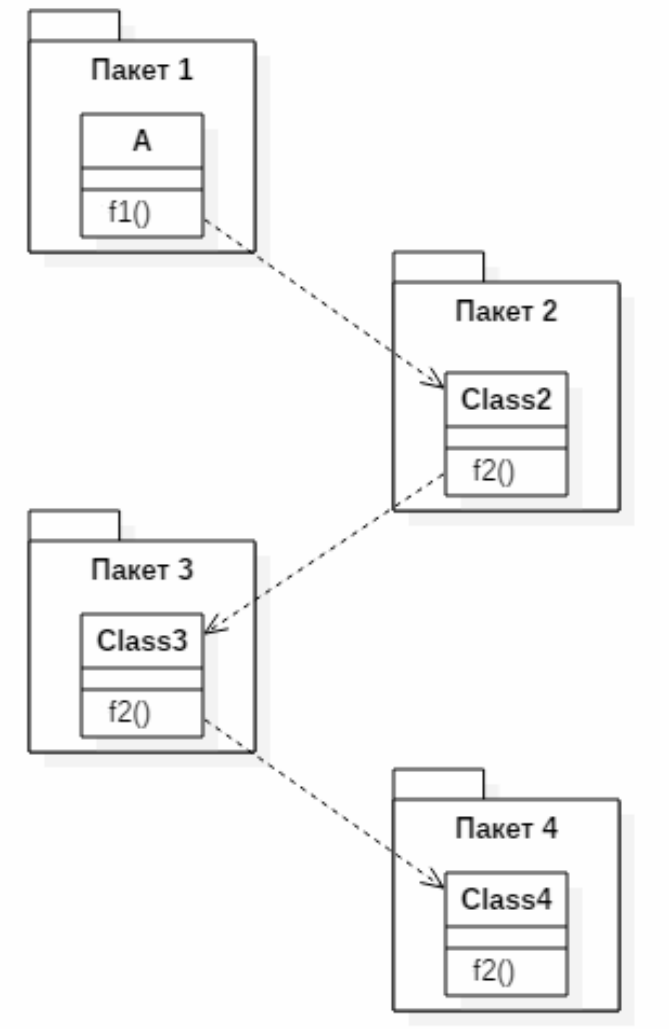


Обработка событий

- **Существует два варианта передачи сообщений:**
  - **Синхронная передача**. При синхронной передаче сообщения объект-клиент A просит объекта-поставщика B выполнить сервис, который может быть делегирован. При этом A зависит от B, потому что A ожидает некоторые результаты выполнения от B.
  - **Асинхронная передача**.

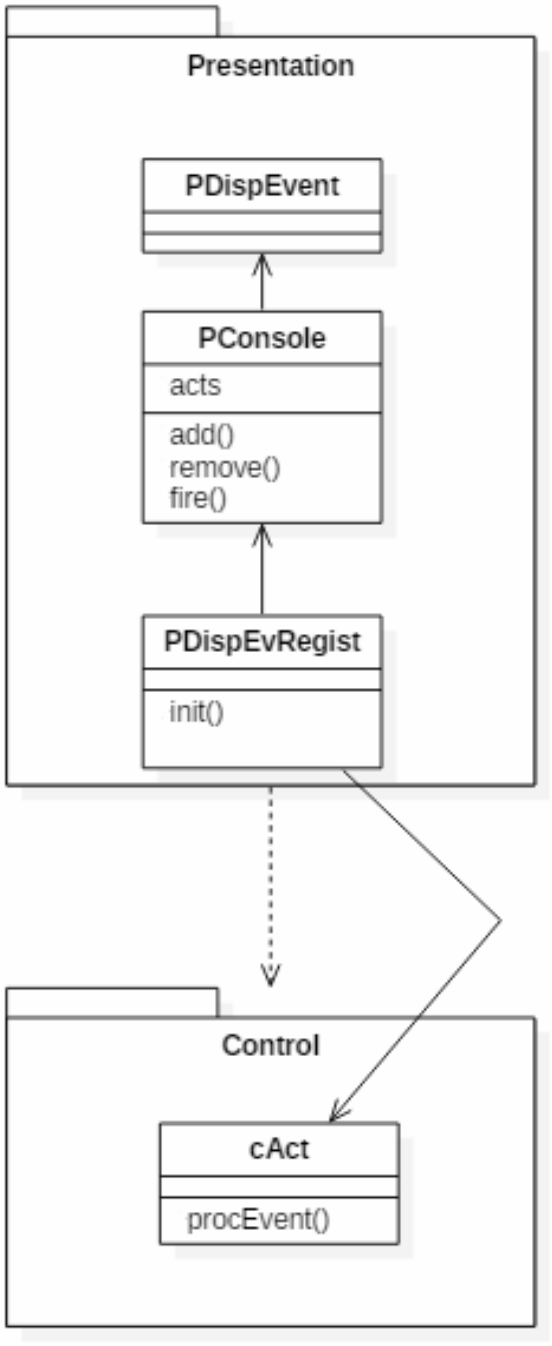
Делегирование

- Передача сообщения реализуется через синхронный вызов от клиента к поставщику сервиса;
- Сообщение от объекта-клиента просит, чтобы объект-поставщик исполнил сервис. Поставщик делегирует выполнение;
- Делегирование обычно необходимо, чтобы позволить объекту-клиенту получить услугу на одном уровне от объекта, находящегося на отдаленном (не соседнем) уровне. Это нужно для поддержания вертикальной структуры уровней. Для выявления зависимостей необходимы явные ассоциации.
- Используются сокрытие через интерфейс и полиморфизм.



Асинхронная передача (Оповещение)

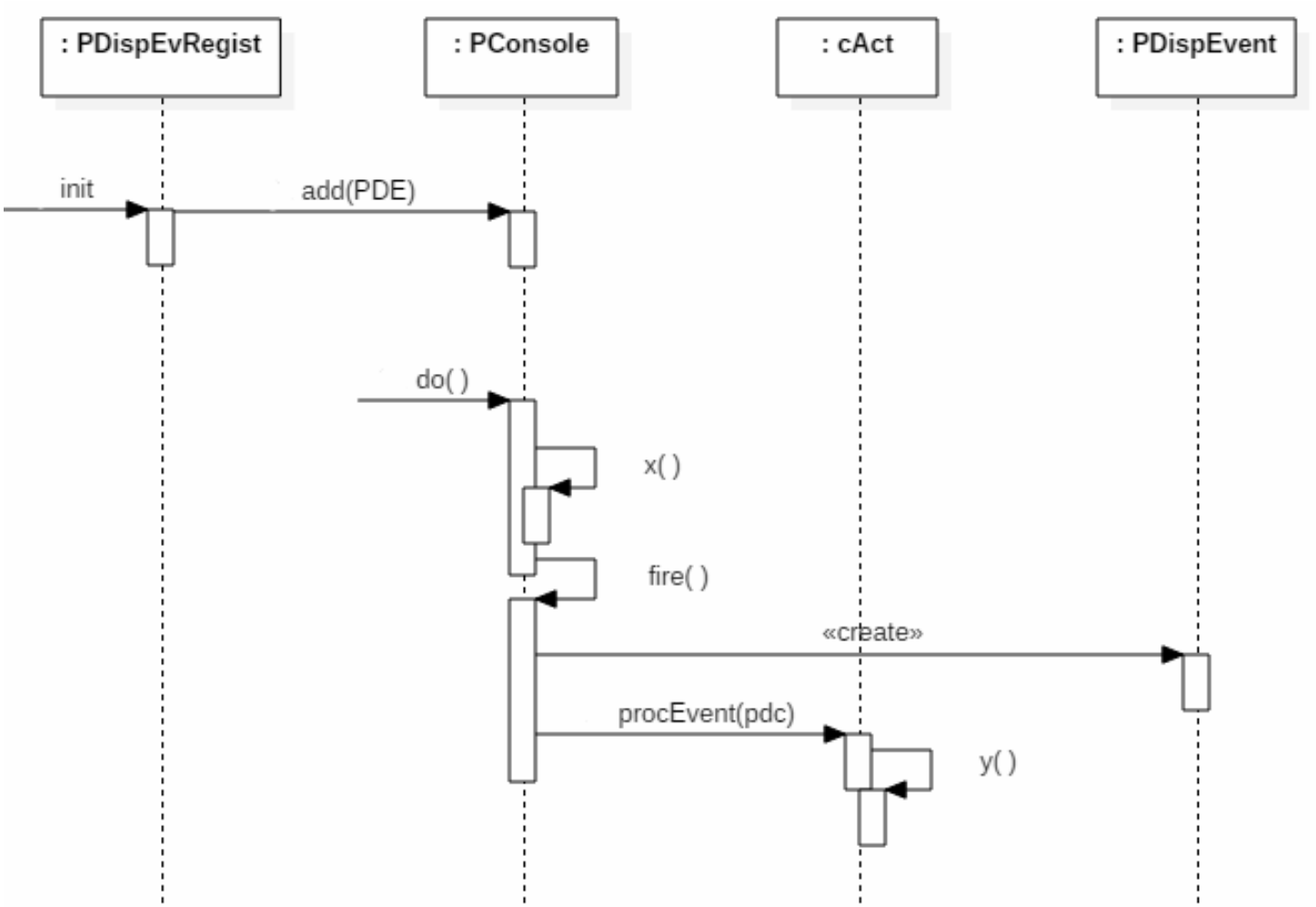
- При асинхронной обработке события для **объекта-издателя** характерно:
  - объект-издатель – отправитель сообщения;
  - объект-издатель имеет список подписчиков и метод их добавления;
  - вызывается извне по событию (например, от GUI), преобразует внешнее событие в свой формат и рассылает его подписчикам;
  - не знает подписчиков.
- **Для объекта-подписчика характерно:**
  - Интересуется событиями, для которых он имеет свои обработчики.
- **Для объекта-регистратора характерно:**
  - наличие необязательно, может быть подписчиком;
  - добавляет подписчика издателю как параметр метода (может быть передан как интерфейс) для нисходящих зависимостей



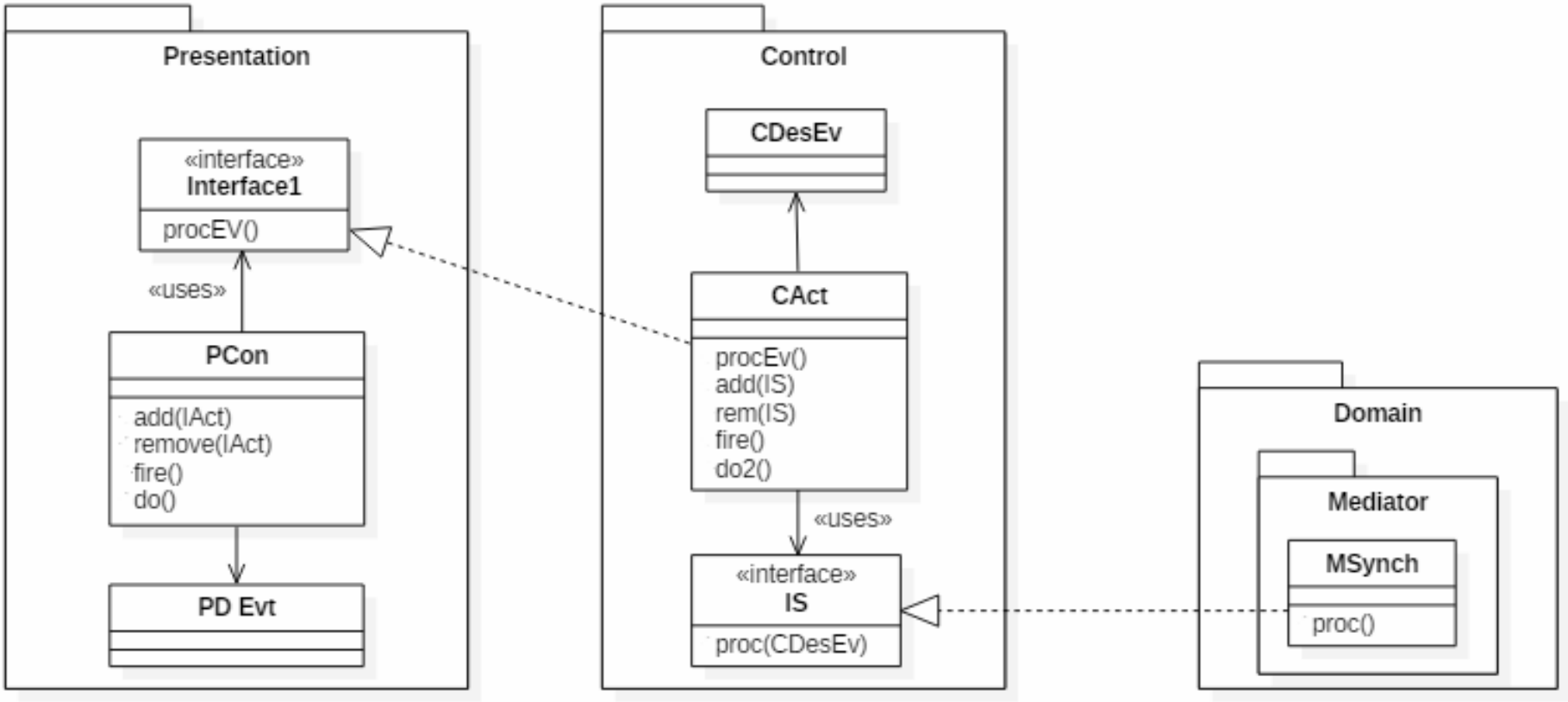
Пример обработки асинхронных событий и зависимости уровней на диаграмме пакетов

- Издатель не зависит от подписчика,
- регистратор зависит от издателя и подписчика, но могут быть использованы интерфейсы, что снизит видимость связей.

Пример обработки асинхронных событий на диаграмме последовательности

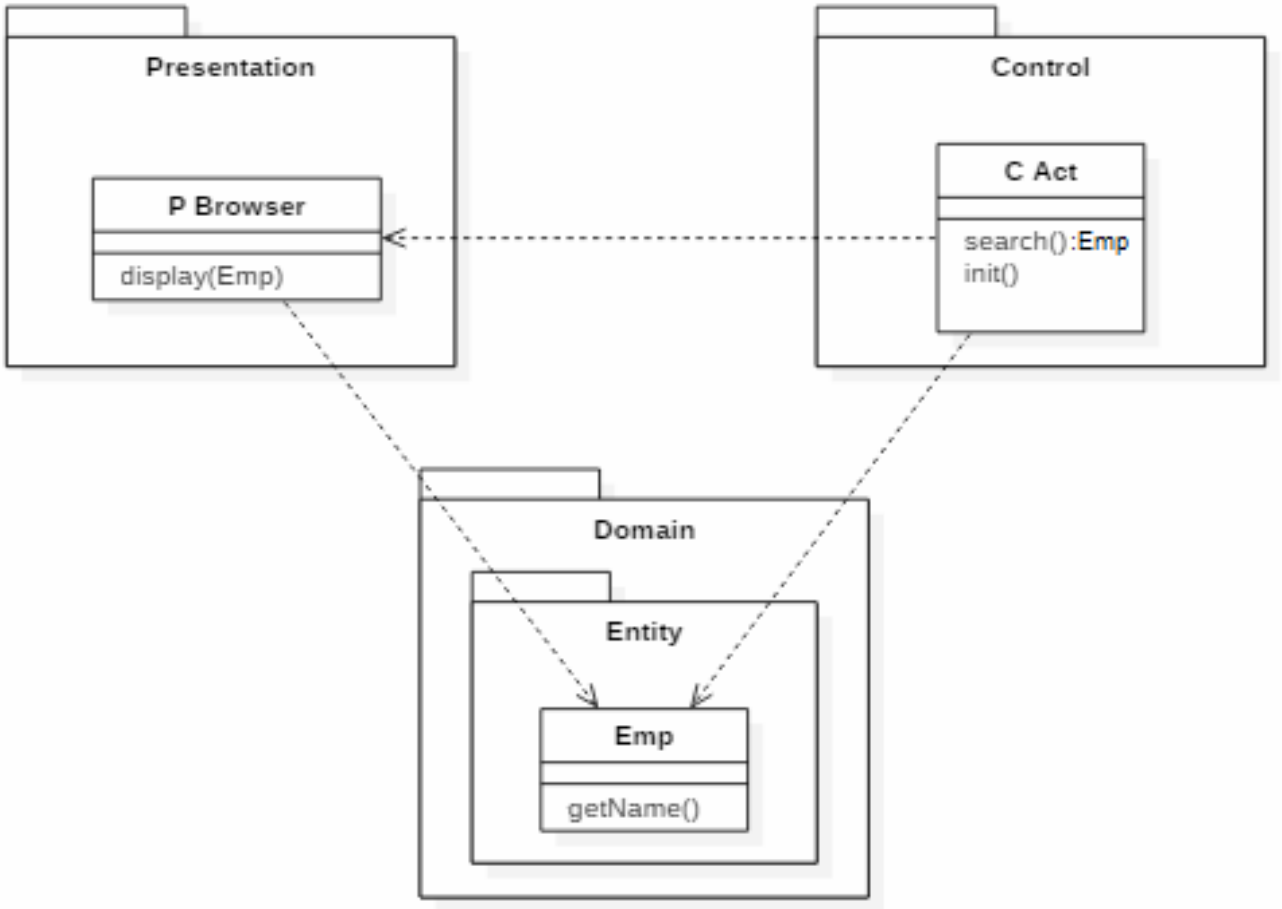


Интерфейсы для нисходящей зависимости обработки событий

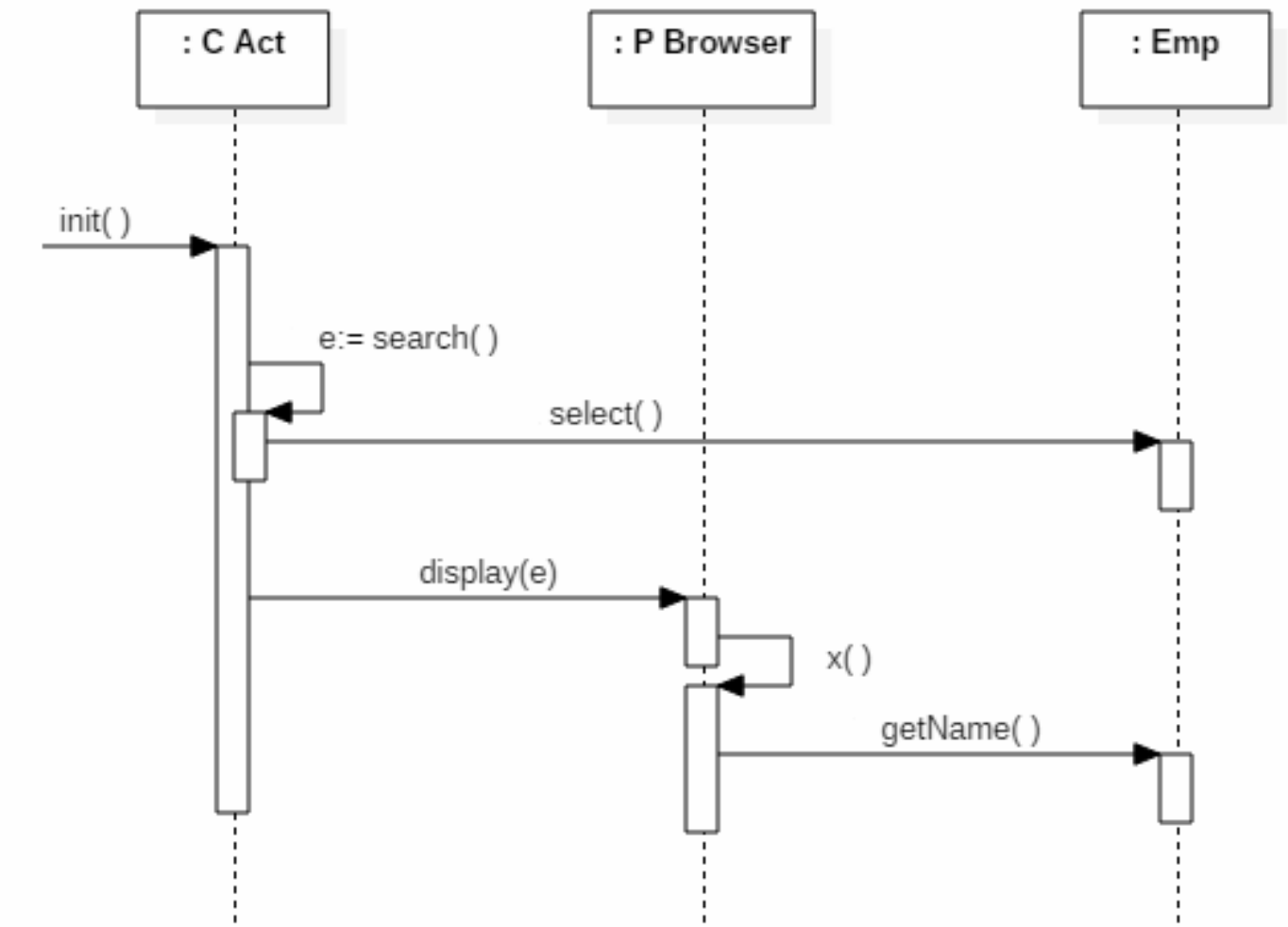


Знакомство

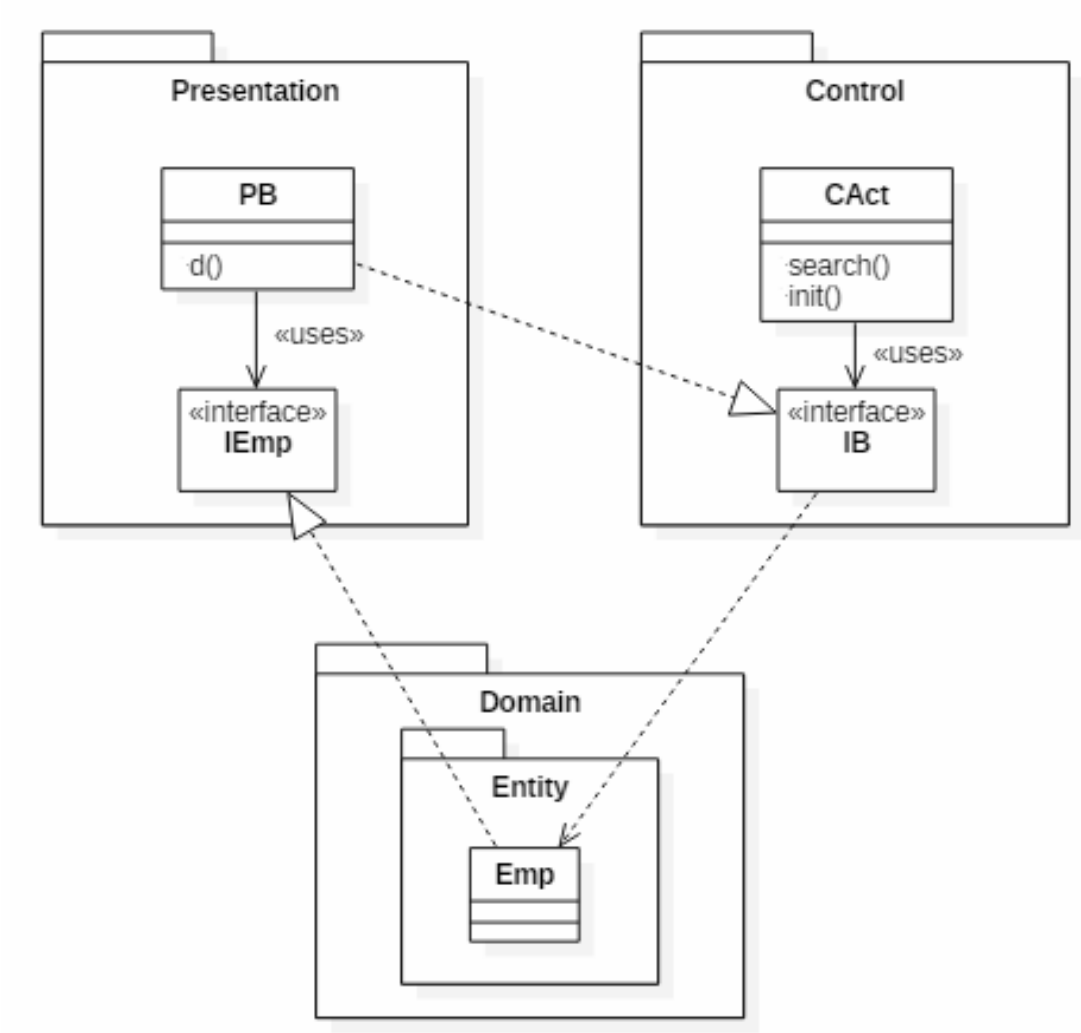
- Знакомство соответствует ситуации, когда объект передает объект другому объекту как аргумент его метода.
- A знакомится с B, если C передаёт B для A как параметр.



Знакомство. Диаграмма последовательности



Реализация знакомства через интерфейсы



Структурный шаблон

- Конструкция (скелет) структурного проекта
- Используется для повторного использования проекта (не кода) для решения определенных проблем/заданий
- Настраивается и расширяется разработчиком (он дописывает код)

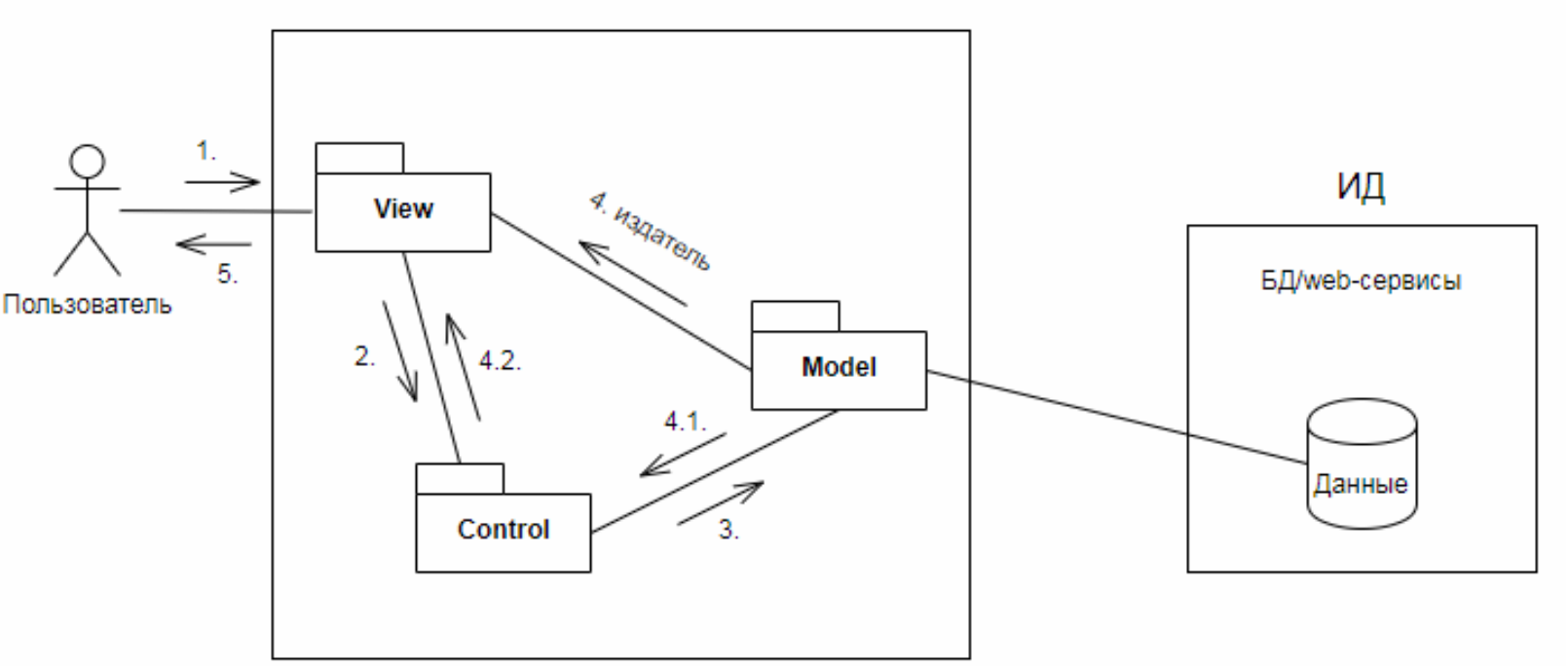
- Шаблон + ПО → ERP (базовое ПО)
- Только проект → MVC, PCMEF, PCMEF+ и др. шаблоны (идея)

Model – View – Controller (MVC)

- (Модель – Представление – Контроллер) – Smalltalk–80.
- 3 группы классов; наследование от абстрактных классов

- **Объекты Model:**
  - Бизнес-сущности, бизнес-права и бизнес-поведения;
  - Связь с источником данных (чтение, запись, соединение и т.д.);
  - Данные из источника данных (ИД) и логика их обработки;
  - Независимы от остальных;
  - Объект-издатель.
- **Объекты View:**
  - Графический (иной) интерфейс с пользователем;
  - Форма и её компоненты (html-страницы и т.д.);
  - Представление подписано на модель (отображение данных и их связей);
  - Отображают состав Model в графическом интерфейсе пользователя (GUI);
  - Передают события от клавиатуры / мыши, т.е. объект-представление связан с объектом-контроллером;
- **Объекты Control:**
  - Преобразует события (от клавиатуры / мыши) в действия, которые выполняются над объектами типа Model;
  - Обычно объект-представление связан с одним объектом-контроллером;
  - Объект-контроллер может быть подписан на объект-модель;
  - Являются инициаторами изменений в Model.

Шаблон MVC


1.
5.
Пользователь
View
4. издатель
2.
4.2.
Model
4.1.
Control
3.
ИД
БД/web-сервисы
Данные

Достоинства шаблона MVC

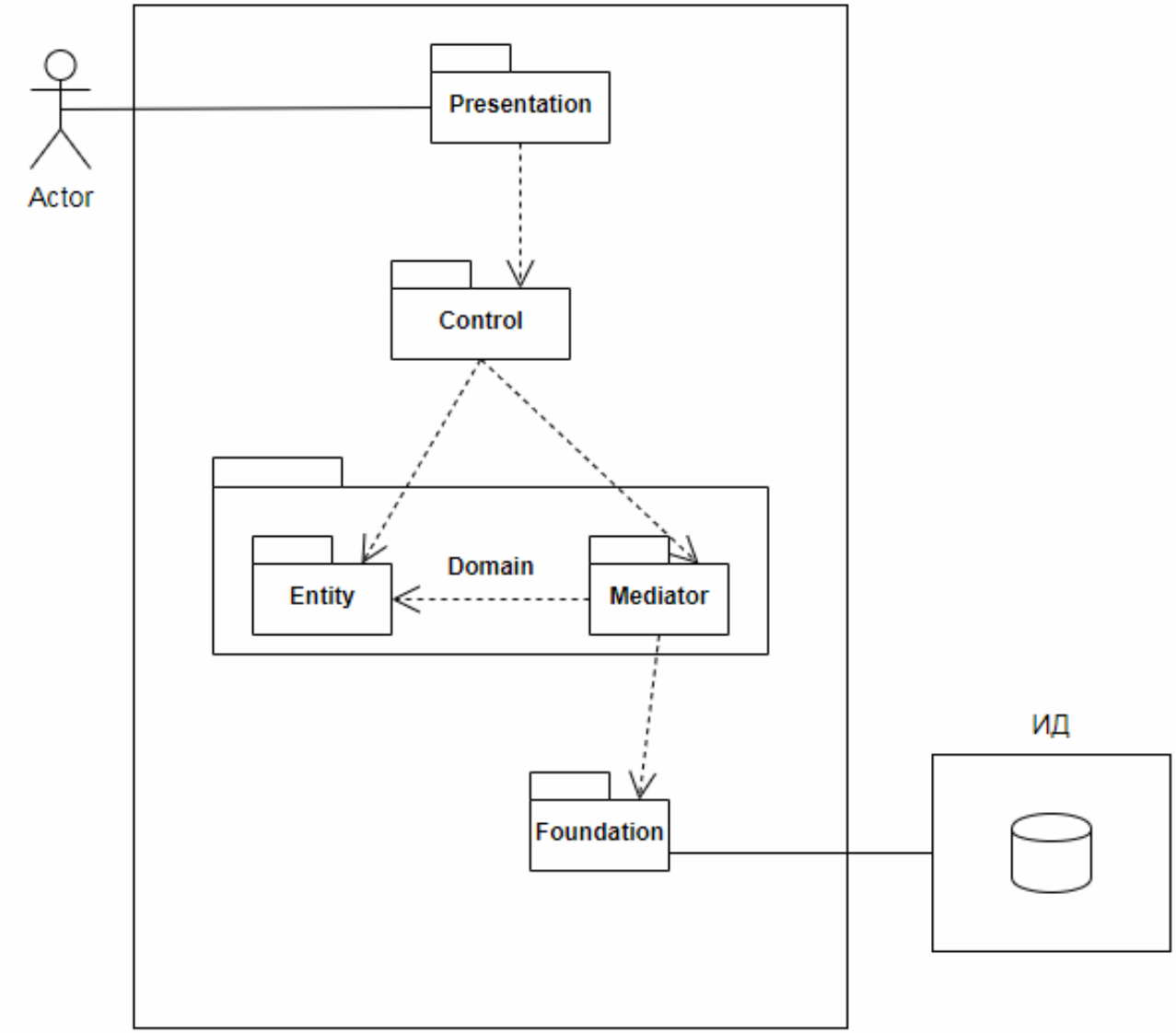
- Раздельная разработка GUI, базы данных и логики модели;
- Изменение GUI (или нескольких GUI) происходит при сохранности Model и Control;
- Изменения Control и Model производятся без изменения GUI;
- Есть возможность выполнения Model без GUI (например, для тестирования или в случае пакетной обработки).

PCMEF

- **PCMEF (Presentation, Control, Mediator, Entity, Foundation)** – уровневый структурный шаблон, организованный в виде вертикальной иерархии (представление, управление, посредник, сущность, основание)

- **Presentation** представляет собой GUI в виде стандартных классов (или их производных).
- В **Control** хранится логика, обработка событий, алгоритмы и сеансы;
- В **Entity** хранятся бизнес-объекты (контейнеры) из БД в ОП;
- **Mediator** является посредником между Entity и Foundation вне Control;
- **Foundation** осуществляет коммуникацию с базой данных или web-сервисами.

Шаблон PCMEF



Принципы PCMEF

1. Нисходящие зависимости, в которых участвуют интерфейсы, абстрактные классы и доминирующие классы.
2. Восходящие уведомления между подписчиками и издателями. Когда издатель в более низком уровне изменяет состояние, он посылает уведомление своим подписчикам.
3. Принцип соседней связи. Пакет должен непосредственно связываться только со своими соседними пакетами. Для не соседних объектов используется делегирование.
4. Явная ассоциация. Принцип явной ассоциации реализуется в пакетах, явно разрешающих передачу сообщений между классами.
5. Устранение циклов. Принцип устранения циклов гарантирует, что циклические зависимости между уровнями, пакетами и классами в пределах пакетов устранены.
6. Именование классов. Даёт возможность определить по имени класса, какому пакету он принадлежит (например, EInvoice – класс пакета Entity). Тот же принцип для интерфейсов (например, ICInvoice – интерфейс пакета Control).
7. Использование пакета знакомств.

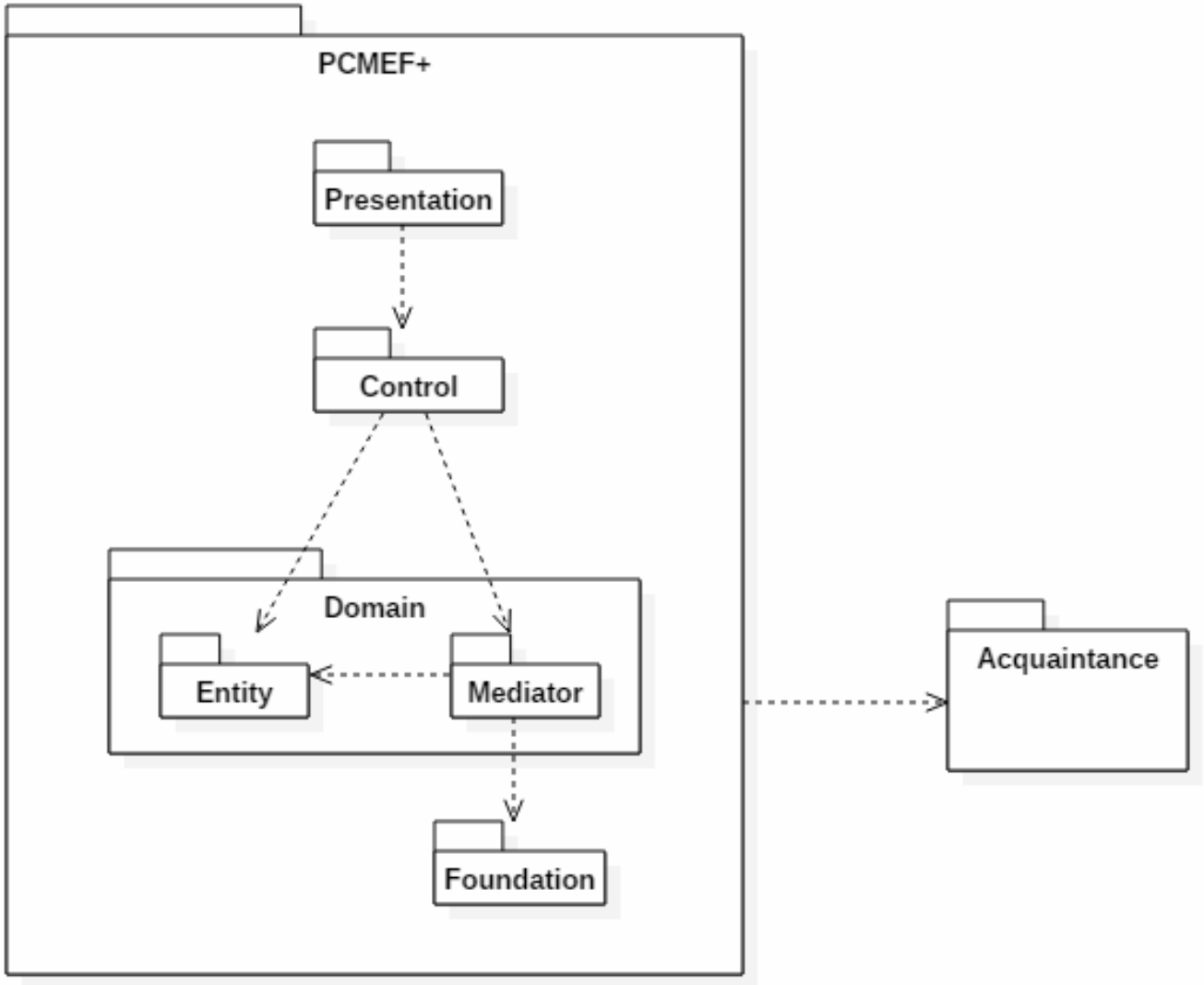
Пакет знакомств

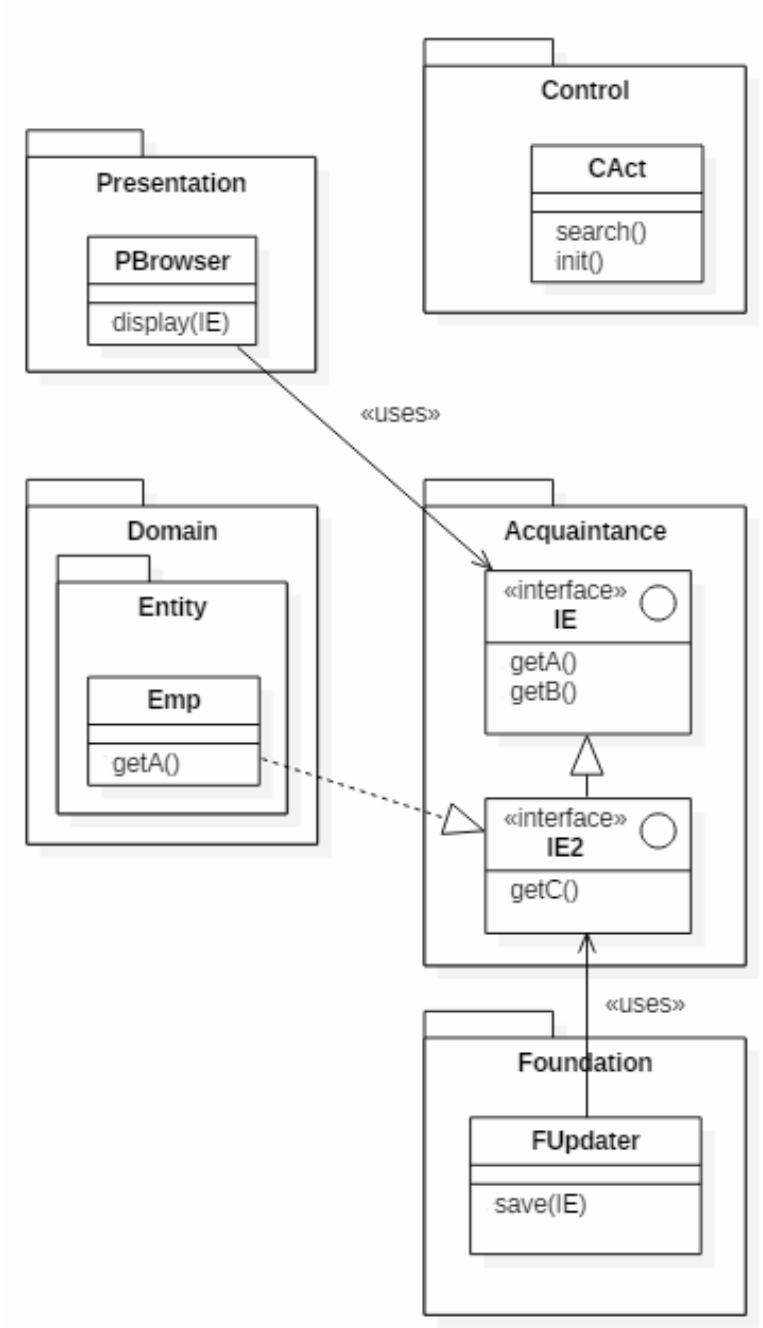
- Знакомство возникает тогда, когда внутри метода сообщение посылается объекту, являющемуся параметром этого метода.
- Для не соседних уровней используется **пакет знакомств**.

- **Пакет знакомств** — это автономный пакет, состоящий только из интерфейсов.

- Низкий уровень является устойчивым, его трудно изменить, но при этом легко расширить.

Расширение шаблона PCMEF до PCMEF+





PCMEF+

- пакет Acquaintance (знакомство) отделяет зависимости знакомств в изолированный пакет, который может управляться независимо.
- Пакет Acquaintance состоит только из интерфейсов.

- IE1 (интерфейс пакета Acquaintance с методами getA() и getB()) расширен интерфейсом IE2 (интерфейс пакета Acquaintance с методом getC()), который в свою очередь реализован классом Emp.

Зависимости в PCMEF+

Зависимости между пакетами будут следующие:

- **Foundation** использует интерфейсы Acquaintance, реализованные в пакетах Domain, Control и Presentation;
- **Domain** использует интерфейсы Acquaintance, реализованные в пакетах Control и Presentation;
- **Entity** использует интерфейсы Acquaintance, реализованные в пакете Presentation;
- **Presentation** использует интерфейсы Acquaintance, реализованные в пакетах Domain и Foundation;
- **Control** использует интерфейсы Acquaintance, реализованные в пакете Foundation.

Варианты развёртывания PCMEF

1. Двухъярусная структура (толстый клиент). На клиенте находятся все пакеты, а на сервере хранится база данных.
2. Одноярусная структура (всё находится на 1 ПК).
3. Тонкий клиент. Все уровни, кроме Presentation, находятся на сервере.
4. Трёхуровневая структура. Некоторые объекты Domain (предметная область) и Foundation (основание) перемещаются в средний уровень. Остальные пакеты хранятся на клиенте. База данных хранится на сервере.
5. Многоярусные структуры
   - В распределенной системе некоторые PCMEF-уровни могут быть отображены EJB- или .NET-компонентами и развернуты отдельно.
   - Mediator хранится на сервере (хранится у клиента только в случае очень толстого клиента).
   - Presentation хранится у клиента (толстый клиент) или на сервере в случае, если это web-сервер.
   - Control хранится на сервере или на клиенте (плохо, если этот пакет хранится и там, и там).

Литература

– Орлов С.А. Технологии разработки программного обеспечения. - Спб.: Питер. - 2002 г.
– Гамма Э. и др. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.:Питер. - 2009 г.
– Мацяшек Л.А., Лионг Б.Л. Практическая программная инженения. - М.: Бином. - 2009 г.