Проектирование программных средств

Лекция 3

Системный анализ

Спецификация требований

- Итогом системного анализа является выработка требований к программному продукту
- Спецификация требований служит исходным документом при проектировании программного средства

Проектирование

- Целью этапа проектирования является построение ***модели разрабатываемого программного продукта***, удовлетворяющей спецификации требований
- В процессе проектирования разрабатывается ***логика*** решения проблем, выявленных на этапе системного анализа

Проектирование

- Результатом этапа проектирования является ***проект*** – набор документов, описывающих модель программного средства, а также ряд сопутствующих документов (детальные планы работ, экономические расчеты и т.д.)

Проектирование

- Для представления модели программного средства используются различные нотации:
  - *блок-схемы,*
  - *ER-диаграммы,*
  - *UML-диаграммы,*
  - *DFD-диаграммы,*
  - *макеты*

Стадии проектирования

- В зависимости от класса создаваемого ПО, проектирование может выполняться как «вручную», так и с использованием различных средствами автоматизации
- Обычно выделяют три стадии проектирования:
  - ***эскизное проектирование***
  - ***детальное проектирование***
  - ***проектирование интерфейса***

Имеет своей целью структурирование разрабатываемого ПС, выделение отдельных относительно независимых подсистем, связанных с решением отдельных подзадач. Завершается созданием эскизного проекта, специфицирующего эти подсистемы и интерфейс между ними.

ЭСКИЗНОЕ ПРОЕКТИРОВАНИЕ

Эскизное проектирование

- Разрабатываемый программный продукт рассматривается как
  - *часть системы обработки информации, включающей аппаратную и программную составляющие*
  - *набор взаимодействующих подсистем*
- На этой стадии определяется ***архитектура системы***, т.е. ее структура и характер взаимодействия отдельных частей

Понятие архитектуры ПС

- Под архитектурой ПС понимают набор ее внутренних структур, которые видны с различных точек зрения и состоят из
  - *архитектурных компонентов,*
  - *связей и возможных взаимодействий между компонентами,*
  - *доступных извне свойств этих компонентов*

Понятие компонента

- Под ***архитектурным компонентом*** в этом определении понимается достаточно произвольно выделенный структурный элемент ПС, который решает некоторые подзадачи в рамках общих задач системы и взаимодействует с остальными частями системы через определенный интерфейс

Роль архитектуры

- Выбор архитектуры ПС задает способ реализации требований на высоком уровне абстракции
- Архитектура ПС почти полностью определяет ее:
  - *надежность,*
  - *переносимость,*
  - *удобство сопровождения*

Роль архитектуры

- Архитектура значительно влияет на эргономику и эффективность ПО, которые, однако, сильно зависят и от реализации отдельных компонентов
- Значительно меньше влияние архитектуры на функциональность — обычно заданную функциональность можно реализовать, использовав совершенно различные архитектуры

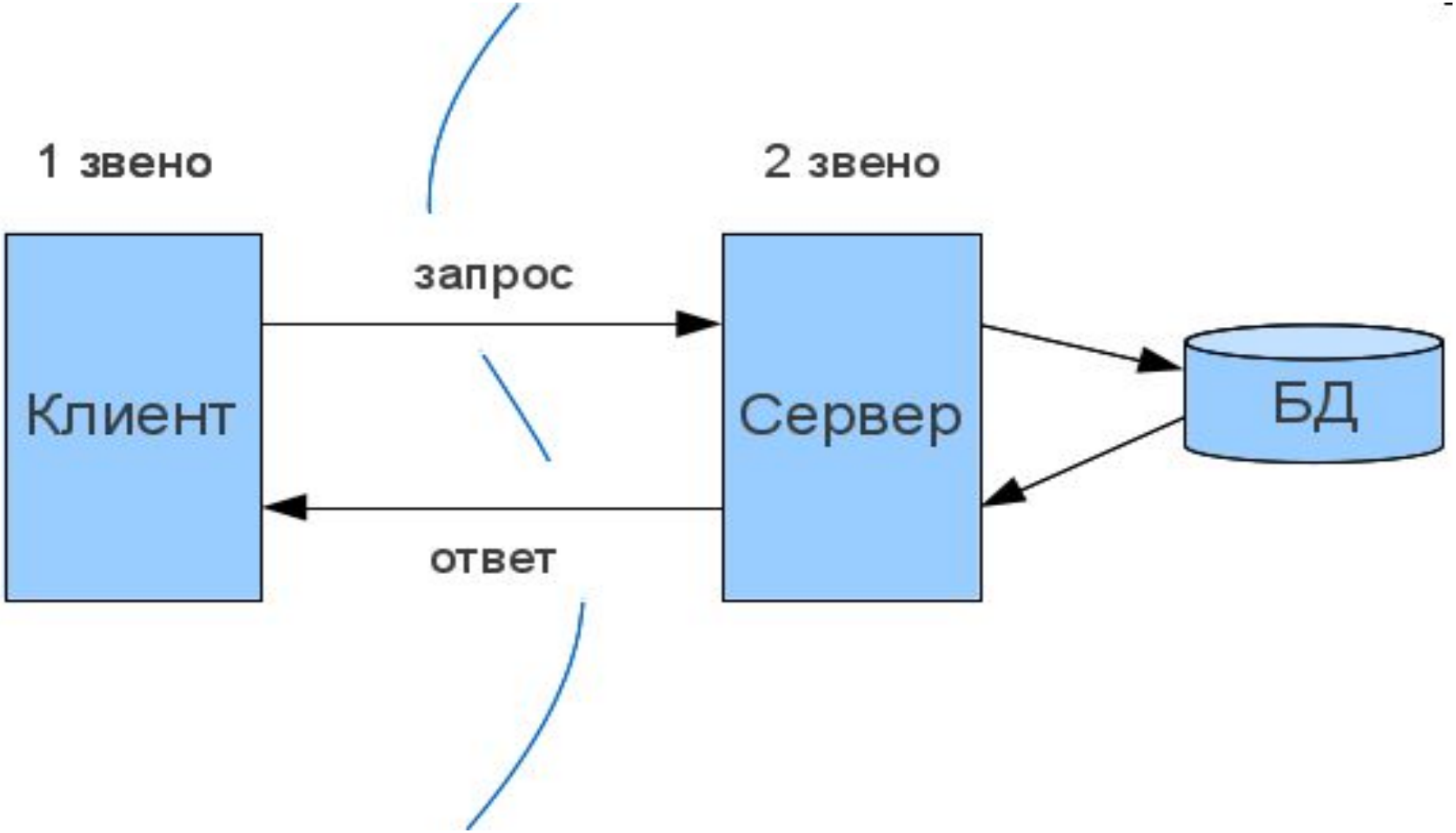
Примеры архитектур

- Клиент-серверная архитектура (Client-server)
- Архитектура распределенных вычислений (Distributed Computing)
- Одноранговая архитектура (Peer-to-peer)
- Архитектура каналов и фильтров (Pipes and filters)
- Расширяемая архитектура (PlugIns)
- Монолитная система (Monolithic System)
- Многоуровневая архитектура (Multitiered)
- Сервис-ориентированная архитектура (Service-oriented)
- Компонентная архитектура (Search-oriented)

Клиент-серверная архитектура

- Архитектура приложения, в котором задания распределены между поставщиками услуг (серверами) заказчиками услуг (клиентами)
- Клиенты и серверы могут взаимодействовать через компьютерную сеть или выполняться на одном компьютере
- Используется для распределенных систем и в большинстве бизнес-приложений

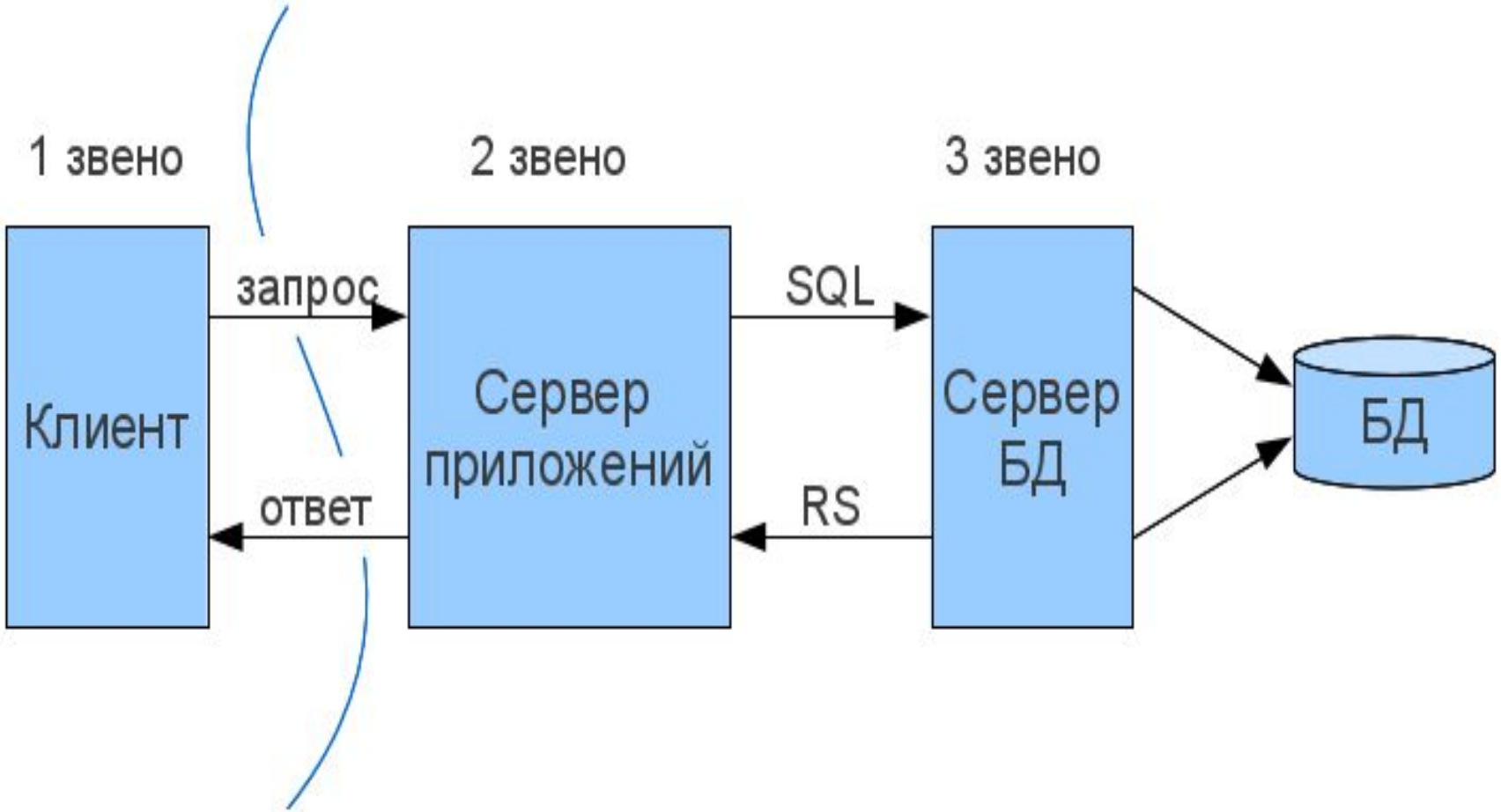
Клиент-серверная архитектура



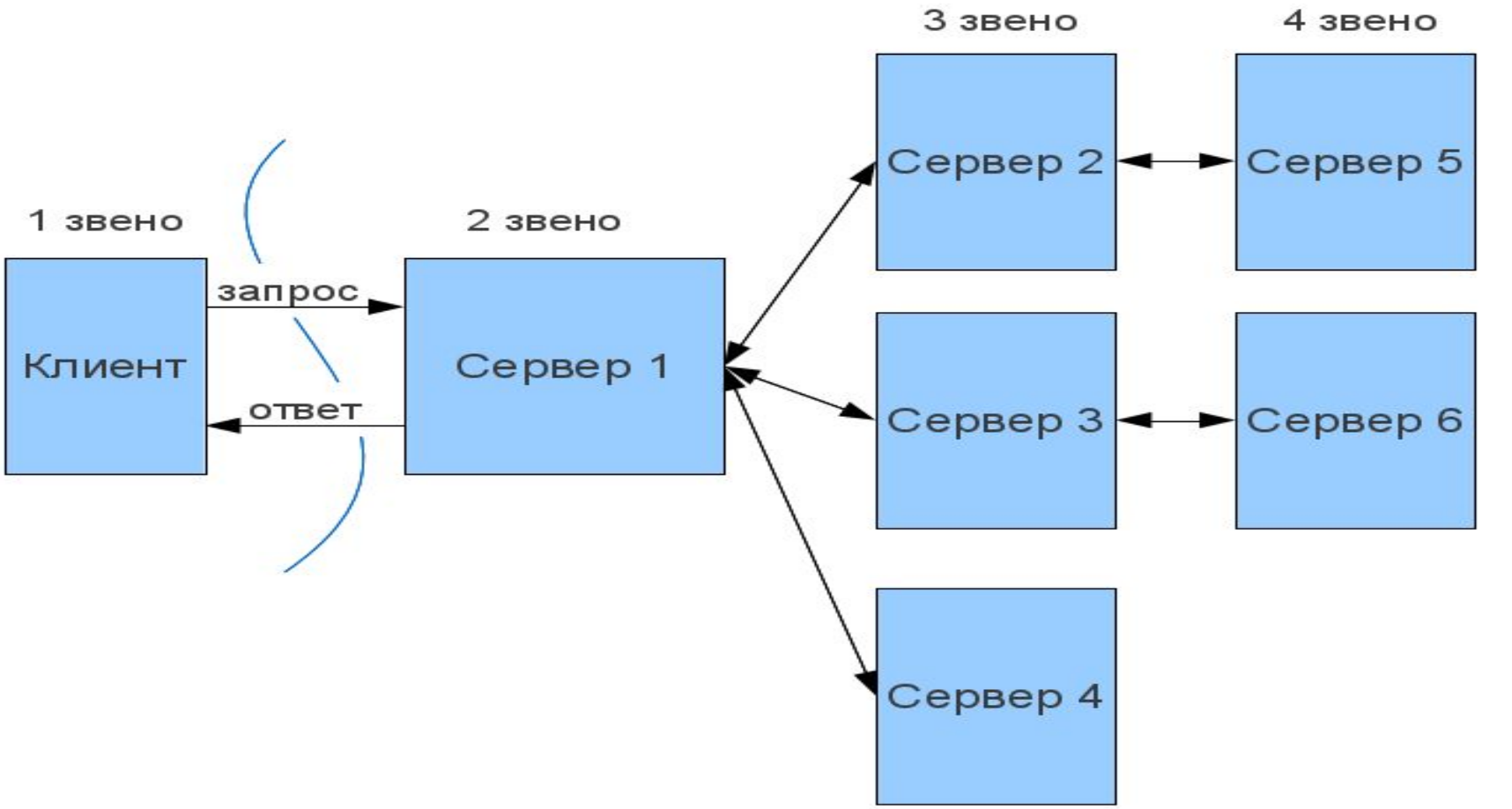
Многоуровневая архитектура

- Разновидность архитектуры клиент-сервер, в которой функция обработки данных вынесена на один или несколько отдельных серверов
- Обеспечивается естественное расслоение задач системы на наборы подзадач, которые можно было бы решать последовательно
- Компоненты системы разделяются на несколько уровней так, что компоненты данного уровня могут использовать для своей работы только соседей или компоненты предыдущего уровня

Трехуровневая архитектура



Многоуровневая архитектура



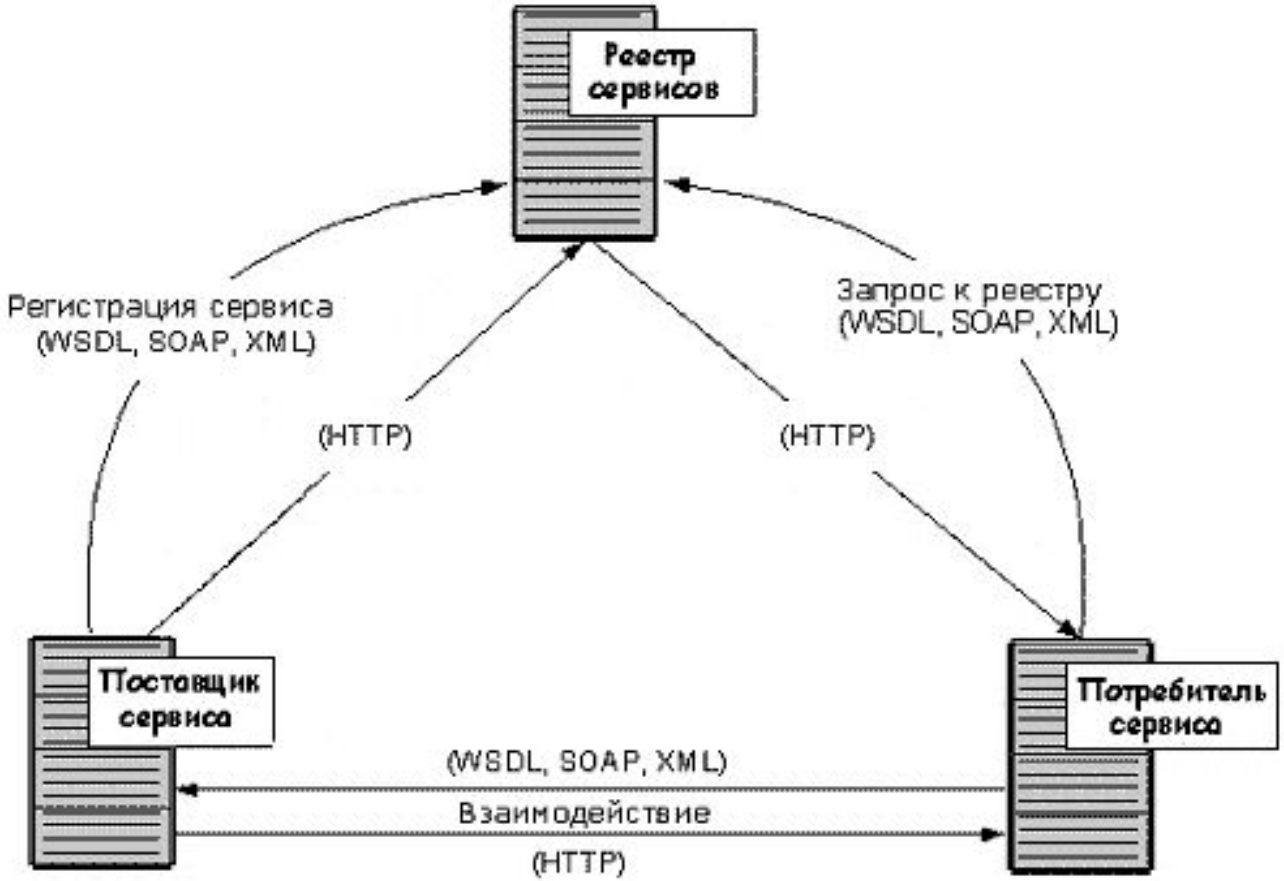
Сервис-ориентированная архитектура

- Такой подход к разработке приложений для управления предприятием, которой предусматривает разложение программных процессов на отдельные услуги, с которыми далее может работать сеть – обеспечивать их поиск и предоставление
- Каждая услуга обладает функциональными возможностями, которые могут быть приспособлены к потребностям предприятия

Сервис-ориентированная архитектура

- SOA "подталкивает" к использованию для построения приложений подхода основанного на связывании уже существующих сервисов, а не на написании нового программного кода
- В самом общем виде SOA предполагает наличие трех основных участников: поставщика сервиса, потребителя сервиса и реестра сервисов
- Поставщик сервиса регистрирует свои сервисы в реестре, а потребитель обращается к реестру с запросом

Схема SOA

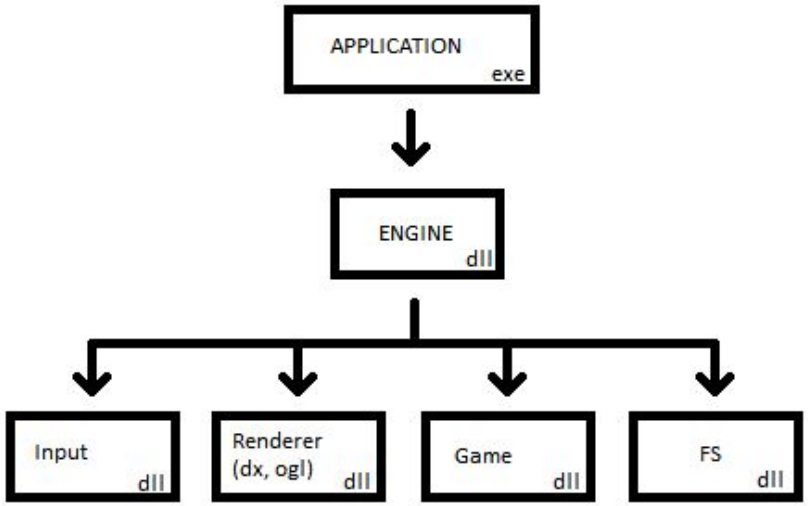

Реестр сервисов
Регистрация сервиса (WSDL, SOAP, XML)
Запрос к реестру (WSDL, SOAP, XML)
(HTTP)
(HTTP)
Поставщик сервиса
Потребитель сервиса
(WSDL, SOAP, XML)
Взаимодействие
(HTTP)

Компонентная архитектура

- Разрабатываемое приложение строится из набора готовых компонентов развертывания со строго определенным интерфейсом
- В данном случае под ***компонентом развертывания*** подразумевается практически независимая и заменяемая часть программной системы, которая соответствует конкретной функции (набору функций)
- Такой компонент может быть легко добавлен в систему или удален из нее

Компонентная архитектура

- Наиболее распространенными платформами для построения компонентной архитектуры являются:
  - Microsoft COM,
  - Sun Enterprise Java Beans,
  - CORBA,
  - Microsoft .Net



Архитектура хранилища данных

- Подсистемы разделяют данные, находящиеся в общей памяти и, как правило, представляющие собой базу данных
- Применима, если основные функции системы связаны с хранением, обработкой и представлением больших количеств данных

Проблемы выбора

- Выбор между той или иной архитектурой определяется в основном нефункциональными требованиями и необходимыми свойствами ПО с точки зрения удобства сопровождения и переносимости
- Для построения хорошей архитектуры надо учитывать возможные противоречия между требованиями к различным характеристикам и уметь выбирать решения, дающие приемлемые значения по всем показателям

Эффективность

- Так, для повышения эффективности в общем случае выгоднее использовать монолитные архитектуры, в которых выделено небольшое число архитектурных компонентов (в пределе — единственный компонент)
- Этим обеспечивается экономия как памяти, так и времени работы, поскольку имеется возможность оптимизировать работу алгоритмов в рамках одного компонента

Удобство сопровождения

- С другой стороны, для повышения удобства сопровождения, наоборот, лучше разбить систему на большое число отдельных небольших архитектурных компонентов, с тем чтобы каждый из них решал свою четко определенную часть общей задачи

Надежность

- С третьей стороны, для повышения надежности лучше использовать либо небольшой набор простых архитектурных компонентов, либо дублирование функций, сделав несколько компонентов ответственными за решение одной подзадачи
- При этом надо использовать достаточно сильно отличающиеся способы решения одной и той же задачи в разных архитектурных компонентах

Эскизный проект

- Результаты эскизного проектирования представляются в виде ***эскизного проекта*** – описания ее архитектурных составляющих и способов взаимодействия между ними
- Стадия эскизного проектирования не является строго обязательной и может быть исключена, если основные проектные решения определены ранее или достаточно очевидны

Целью детального проектирования является полная спецификация архитектурных составляющих программного средства. Выполняется параллельно с проектированием пользовательского интерфейса и завершается созданием технического проекта как основы для написания программного кода

ДЕТАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Детальное проектирование

- На стадии детального проектирования конкретизируются решения архитектурного уровня и производится:
  - *разработка иерархии классов и структуры базы данных,*
  - *детализация структуры отдельных архитектурных компонентов*
  - *построение алгоритмов для отдельных подзадач,*
  - *поиск и подбор готовых компонентов для реализации некоторых функций системы*

Программные модули

- ***Программный модуль*** – это любой фрагмент ПС, который программируется, компилируется и отлаживается отдельно от других частей программы, и тем самым, физически разделен с ними
- Компонент развертывания можно рассматривать как частный случай программного модуля, специально предназначенного для распространения и повторного использования

Взаимодействие модулей

- Объединение многих модулей в единую систему достигается через *интерфейсы модулей*
- Интерфейс модуля – это описание тех средств (структур данных и структур управления), которые данный модуль :
  - предоставляет для внешнего использования (*экспортирует*)
  - заимствует у других модулей (*импортирует*)

Принцип инкапсуляции

- Таким образом, модуль делится на две части:
  - внешнюю – интерфейс,
  - внутреннюю – реализацию
- Интерфейс модуля – это его представление как "чёрного ящика" с известными входами и выходами
- Объявленная в интерфейсе функциональность модуля реализуется в его внутренней, невидимой «извне», части

Скрытие информации

- Модуль, подобен айсбергу; лишь его верхушка - интерфейс - видна клиентам

Характеристики модуля

- Для оценки качества программного модуля обычно используют следующие его характеристики:
  - размер модуля,
  - связность (прочность) модуля,
  - сцепление с другими модулями,
  - рутинность модуля

Размер модуля

- *Размер* модуля измеряется числом содержащихся в нем операторов или строк
- Модуль не должен быть слишком маленьким или слишком большим
- Обычно рекомендуются программные модули размером порядка нескольких сотен операторов

Прочность модуля

- ***Прочность*** (*cohesion*) модуля – это мера его внутренних связей
- Чем выше прочность модуля, тем больший вклад в упрощение программы он может внести
- По степени прочности модули делятся на:
  - *прочные по совпадению,*
  - *функционально прочные,*
  - *информационно прочные*

Прочность по совпадению

- *Прочным по совпадению* называется такой модуль, между элементами которого нет осмысленных связей (повторяющийся фрагмент кода)
- Подобный класс программных модулей не рекомендуется для использования

Функциональная прочность

- *Функционально прочный* модуль – это модуль, выполняющий (реализующий) одну какую-либо определенную функцию
- При реализации этой функции такой модуль может использовать и другие модули
- Такой класс программных модулей рекомендуется для использования

Информационная прочность

- *Информационно прочный* модуль – это модуль, реализующий несколько операций над одной и той же структурой данных (информационным объектом), которая считается неизвестной вне этого модуля
- Информационно прочный модуль может реализовывать, например, абстрактный тип данных
- Такой класс программных модулей обладает высшей степенью прочности

Сцепление модуля

- ***Сцепление*** (*coupling*) модуля – это мера его зависимости по данным от других модулей
- Характеризуется способом передачи данных. Чем слабее сцепление модуля с другими модулями, тем сильнее его независимость от других модулей
- Виды сцепления:
  - *сцепление по содержимому,*
  - *сцепление по общей области,*
  - *параметрическое сцепление*

Виды сцепления модулей

- *Сцепление по содержимому*. Таким является сцепление двух модулей, когда один из них имеет прямые ссылки на содержимое другого модуля (например, на константу, содержащуюся в другом модуле)
- Такое сцепление модулей недопустимо

Виды сцепления модулей

- *Сцепление по общей области* – это такое сцепление модулей, когда несколько модулей используют одну и ту же область памяти
- Не рекомендуется использовать

Виды сцепления

- *Параметрическое сцепление* – это случай, когда данные передаются модулю либо при обращении к нему как значения его параметров, либо как результат его обращения к другому модулю для вычисления некоторой функции
- Рекомендуется для использования

Рутинность модуля

- *Рутинность* модуля – это его независимость от предыстории обращений к нему
- Модуль будем называть *рутинным*, если результат обращения к нему зависит только от значений его параметров и не зависит от предыстории обращений к нему
- Иначе модуль называется *зависящим от предыстории.*

Модульность

- В результате модульной декомпозиции разрабатываемое программное средство должно быть представлено в виде системы, разбитой на множество модулей с и низкой степенью зацепления и сильной связностью
- Слабая степень зацепления является признаком хорошо структурированности и в сочетании с сильной связностью обеспечивает хорошие показатели качества программного средства

Спецификация модуля

- Модульная структура программы должна включать и совокупность спецификаций модулей, образующих эту программу
- Спецификация программного модуля содержит
  - *синтаксическую спецификацию* его входов, позволяющую построить на используемом языке программирования синтаксически правильное обращение к нему;
  - *функциональную спецификацию* модуля (описание семантики функций, выполняемых этим модулем по каждому из его входов).

Имеет целью разработку пользовательского интерфейса, обеспечивающего эргономичность разрабатываемого программного средства. Выполняется совместно с детальным проектированием. Результаты фиксируются в техническом проекте

ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА

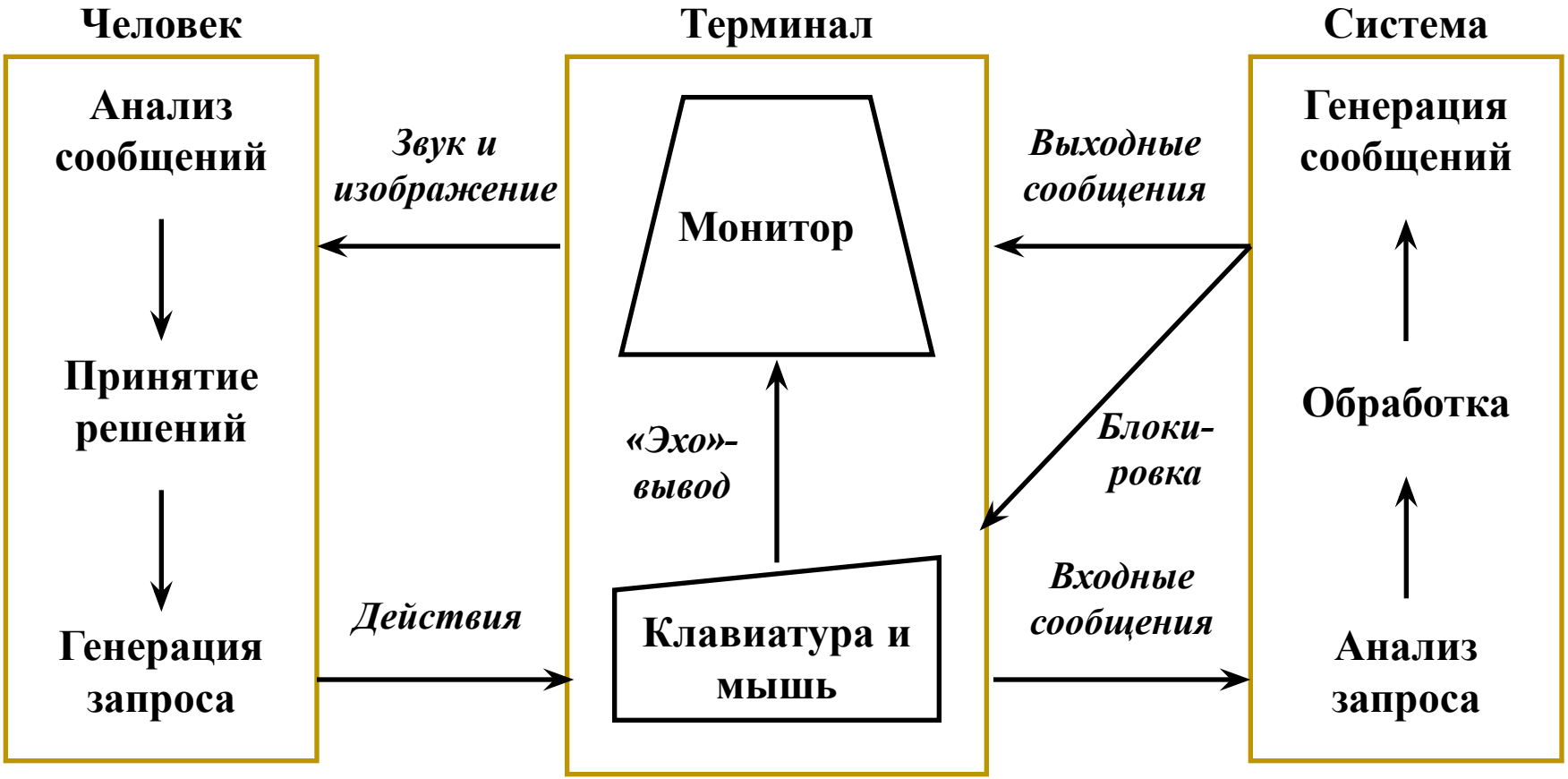
Пользовательский интерфейс

- *Пользовательский интерфейс* представляет собой совокупность программных и аппаратных средств, обеспечивающих взаимодействие пользователя с компьютером
- Основу такого взаимодействия составляют *диалоги* – регламентированный обмен информацией между человеком и компьютером, осуществляемый в реальном масштабе времени и имеющий целью координацию их действий

Диалоги

- Каждый диалог состоит из отдельных процессов ввода-вывода, которые физически обеспечивают связь пользователя и компьютера
- Обмен информацией осуществляется передачей сообщений и управляющих сигналов, а именно:
  - входных сообщений, генерируемых человеком с помощью средств ввода;
  - выходных сообщений, которые генерируются компьютером в виде текстов, звуковых сигналов и/или изображений

Схема диалога



Элементы интерфейса

- Пользовательский интерфейс объединяет в себе все элементы и компоненты программы, которые способны оказывать влияние на его взаимодействие с программным обеспечением
- К этим элементам относятся:
  - *набор задач пользователя, которые он решает при помощи системы;*
  - *используемая системой метафора (например, Рабочий стол в MS Windows);*
  - *элементы управления системой;*

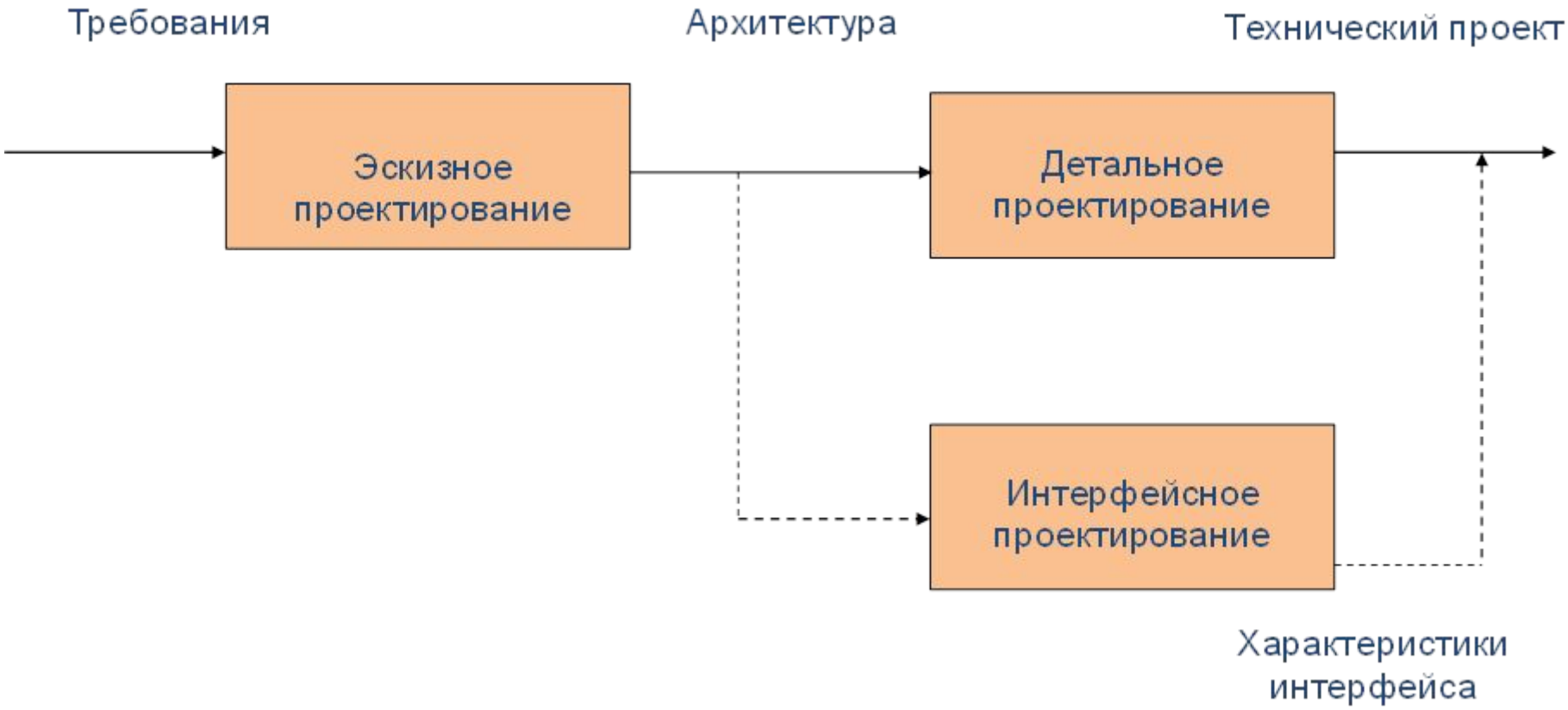
Элементы интерфейса

- *навигация между блоками системы;*
- *визуальный дизайн экранов программы;*
- *отображаемая информация и ее форматы;*
- *устройства и технологии ввода данных;*
- *поддержка принятия решений в конкретной предметной области;*
- *порядок использования программы и документация на нее.*

Технический проект

- Результаты детального и интерфейсного проектирования представляются в техническом проекте (*Software Design Document*)
- Технический проект должен также содержать оценку экономической эффективности системы и перечень мероприятий по подготовке к ее внедрению

Процесс проектирования



ШАБЛОНЫ ПРОЕКТИРОВАНИЯ

Шаблоны проектирования

- *Шаблоны проектирования* (*паттерн*, англ. design pattern) — это многократно применяемая конструкция, предоставляющая решение общей проблемы проектирования в рамках конкретного контекста
- Паттерн не является законченным образцом проекта, который может быть прямо преобразован в код, скорее это описание или образец для того, как решить задачу, таким образом, чтобы это можно было использовать в различных ситуациях

Преимущества шаблонов

- Каждый отдельный шаблон описывает решение целого класса абстрактных проблем
- Шаблоны позволяют унифицировать терминологию, названия модулей и элементов проекта.
- Шаблон проектирования позволяет, повторно использовать удачное решение
- И, наконец, шаблоны независимы от применяемого языка программирования

Критика шаблонов

- Слепое применение шаблонов, без осмысления причин и предпосылок выделения каждого отдельного шаблона, замедляет профессиональный рост программиста
- Знакомиться со списками шаблонов надо только после достижения определенного уровня профессионализма

Классификация шаблонов

- *Шаблоны анализа (analysis patterns)* – представляют собой типовые решения при моделировании сложных взаимоотношений между понятиями некоторой предметной области
- Делятся на шаблоны *функционального* и *объектно-ориентированного анализа*
- Используются на этапе анализа (предметной области, системного, требований)

Классификация шаблонов

- *Архитектурные шаблоны (architectural styles, architectural patterns)*
- Такие образцы представляют собой типовые способы организации системы в целом или крупных подсистем; задают некоторые правила выделения компонентов и реализации взаимодействий между ними
- Используются на стадии эскизного проектирования

Архитектурные шаблоны

- Конвейер обработки данных (data flow).
  - *Пакетная обработка (batch sequential)*
  - *Каналы и фильтры (pipe-and-filter)* – утилиты UNIX
- Вызов-возврат (call-return)
  - *Процедурная декомпозиция* – основная схема построения программ для языков C, Pascal, Ada
  - *Абстрактные типы данных (abstract data types)* – библиотеки классов и компонентов
  - *Многоуровневая система (layers)* – протоколы сетей передачи данных
  - *Клиент-сервер* – основная модель бизнес-приложений

Архитектурные шаблоны

- Интерактивные системы
  - *Данные–представление–обработка (model-view-controller, MVC)*
  - *Представление–абстракция–управление (presentation-abstraction-control)* – интерактивная система на основе агентов, имеющих собственные состояния и пользовательский интерфейс
- Системы на основе хранилища данных
  - *Репозиторий (repository)* – выделяется общее хранилище данных - репозиторий
  - *Классная доска (blackboard*) – системы распознавания текста

Классификация шаблонов

- *Шаблоны проектирования (design patterns)*
- Определяют типовые проектные решения для часто встречающихся задач среднего уровня, касающиеся структуры одной подсистемы или организации взаимодействия двух-трех компонентов
- Применяются на стадии детального проектирования

Классификация шаблонов

- *Идиомы (idioms, programming patterns)*
- Идиомы являются специфическими для некоторого языка программирования способами организации элементов программного кода, позволяющими решить некоторую часто встречающуюся задачу
- Используются на этапе реализации (кодирования)

Классификация шаблонов

- *Образцы организации (organizational patterns) и образцы процессов (process patterns)*
- Образцы этого типа описывают успешные практики организации разработки ПО или другой сложной деятельности, позволяющие решать определенные задачи в рамках некоторого контекста, который включает ограничения на возможные решения.

Описание шаблонов

- Таким образом, шаблоны, понимаемые как образцы решения неких типовых задач могут применяться на всех стадиях разработки программных систем, способствуя сокращению этих сроков
- При описании шаблона выделяют четыре его составляющих:
  - *имя,*
  - *задача,*
  - *решение,*
  - *результаты*

Имя шаблона

- Сославшись на имя, можно сразу описать проблему, ее решения и последствия
- Присваивание шаблонам имен позволяет проектировать на более высоком уровне абстракции
- С помощью словаря шаблонов можно вести обсуждение с коллегами, упоминать шаблоны в документации, представлять тонкости системы

Задача

- Описание того, когда следует применять шаблон
- Формулируется задача и ее контекст (например, представить алгоритм в виде объектов)
- Иногда в описание задачи входит перечень условий, при выполнении которых имеет смысл применять шаблон

Решение

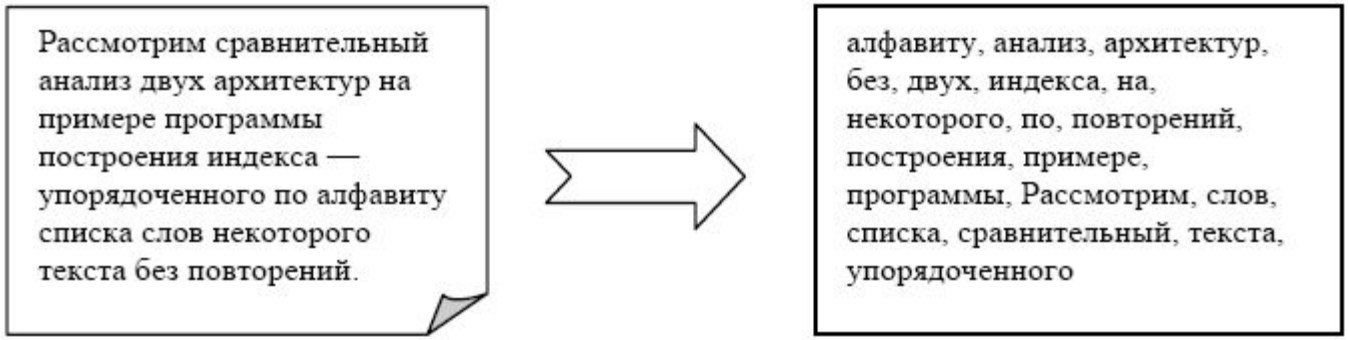
- Описание элементов решения, отношений между ними, функций каждого элемента
- При этом решение – не конкретный дизайн или реализация, а абстрактное описание задачи и того, как она может быть решена с помощью некоего весьма общего сочетания элементов (в случае проектирования, например, это могут быть объекты и классы)

Результаты

- Результаты - это следствия применения шаблона и разного рода компромиссы
- Иногда в результатах может быть описан выбор языка и реализации
- В случае проектирования к результатам относят влияние на степень гибкости, расширяемости и переносимости системы

Пример архитектурного шаблона

- Рассмотрим сравнительный анализ двух архитектур на примере индексатора — программы для построения индекса некоторого текста, т.е. упорядоченного по алфавиту списка его слов без повторений



Сценарии работы

- Выделим следующие сценарии работы или модификации программы:
  - Надо сделать так, чтобы индексатор мог работать в инкрементальном режиме, читая на входе одну фразу за другой и пополняя получаемый в процессе работы индекс
  - Надо сделать так, чтобы индексатор мог игнорировать предлоги, союзы, местоимения, междометия, частицы и другие служебные слова
  - Надо сделать так, чтобы индексатор мог обрабатывать тексты, подаваемые ему на вход в виде архивов
  - Надо сделать так, чтобы в индексе оставались только слова в основной грамматической форме — существительные в единственном числе и именительном падеже, глаголы в неопределенной форме и пр.

Архитектура «каналы-фильтры»

- Определим две возможных архитектуры индексатора для сравнительного анализа
- В качестве первой архитектуры рассмотрим разбиение индексатора на два компонента:
  - Один компонент принимает на свой вход входной текст, полностью прочитывает его и выдает на выходе список слов, из которых он состоит
  - Второй компонент принимает на вход список слов, а на выходе выдает его упорядоченный вариант без повторений

Архитектура «каналы-фильтры»

Архитектура «репозиторий»

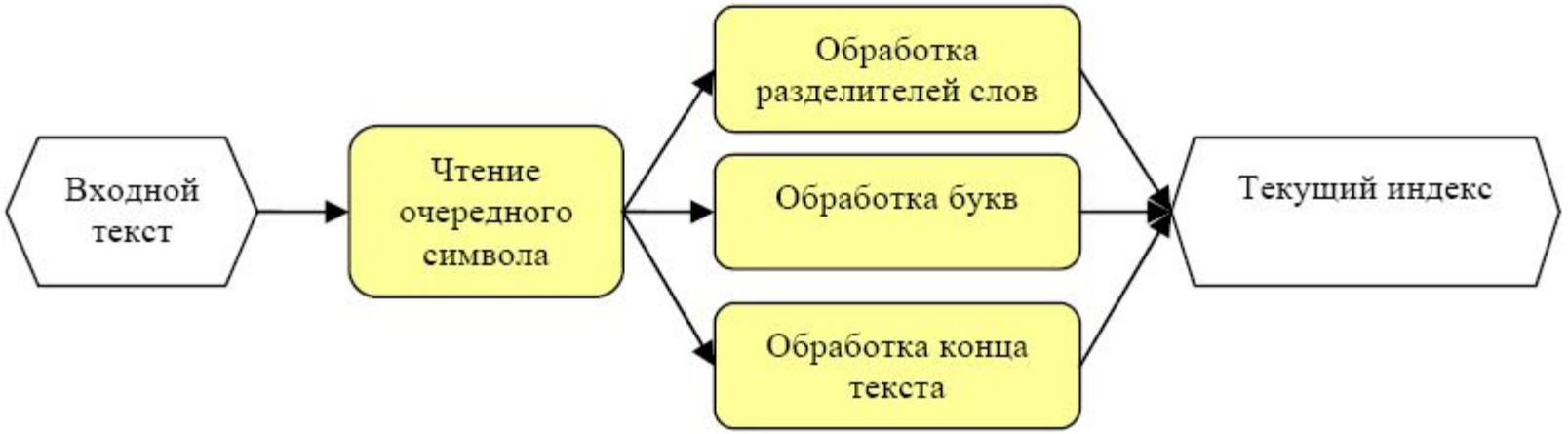
- Другой вариант архитектуры индексатора устроен следующим образом
- Имеется упорядоченный список без повторений всех слов, прочитанных до настоящего момента
- Имеются две переменные:
  - строка, хранящая последнее (быть может, не до конца) прочитанное слово
  - ссылка на то слово в подготовленном списке, которое лексикографически следует за последним словом (предшествующее этому слово в списке предшествует последнему прочитанному слову)

Архитектура «репозиторий»

- В дополнение к этим данным имеются следующие компоненты:
  - Первый читает очередной символ на входе и передает его на обработку одному из остальных. Если это разделитель слов (пробел, табуляция, перевод строки), управление получает второй компонент. Если это буква — третий. Если входной текст кончается — четвертый
  - Второй компонент закачивает ввод последнего слова — оно помещается в список перед тем местом, на которое указывает ссылка; после чего последнее слово становится пустым, а ссылка начинает указывать на первое слово в списке

Архитектура «репозиторий»

- Третий компонент добавляет прочитанную букву в конец последнего слова, после чего, быть может, перемещает ссылку на следующее за полученным слово в списке
- Четвертый компонент выдает полученный индекс на выход.



Сценарий а

- Этот сценарий прямо поддерживается второй архитектурой.
- Чтобы поддержать его в первой архитектуре, необходимо внести изменения в оба компонента так, чтобы первый компонент мог бы пополнять промежуточный список, читая входной текст фраза за фразой, а второй — аналогичным способом пополнять результирующий упорядоченный список, вставляя туда поступающие ему на вход слова.

Сценарий b

- Обе архитектуры не поддерживают этот сценарий
- В первой архитектуре необходимо изменить первый компонент или, лучше, вставить после него дополнительный фильтр, отбрасывающий вспомогательные части речи
- Во второй архитектуре нужно ввести дополнительный компонент, который перехватывает буквы, выдаваемые модулем их обработки и сигналы о конце слова от первого компонента, после чего он должен отсеивать служебные слова

Сценарий с

- Этот сценарий также требует изменений в обеих архитектурах
- В обоих случаях эти изменения одинаковы — достаточно добавить дополнительный компонент, декодирующий архивы, если они подаются на вход

Сценарий d

- Этот сценарий также не поддерживается обеими архитектурами
- Требуемые им изменения аналогичны требованиям второго сценария, только в этом случае дополнительный компонент-фильтр должен еще и преобразовывать слова в их основную форму и только после этого пытаться добавить результат к итоговому индексу

Сравнение двух архитектур

Архитектура	Сценарий a	Сценарий b	Сценарий c	Сценарий d
Каналы и фильтры	- -	+ + *	+ + *	+ + *
Репозиторий	+ + + +	+ + - + *	+ + + + *	+ + - + *

+ обозначает возможность не изменять компонент,

- — необходимость изменения компонента,

* — необходимость добавления одного компонента

Сравнение двух архитектур

- В целом первая архитектура на предложенных сценариях выглядит лучше второй.
- Единственный ее недостаток — отсутствие возможности инкрементально поставлять данные на вход компонентам.
- Если его устранить, сделав компоненты способными потреблять данные постепенно, эта архитектура станет почти идеальным вариантом, поскольку она легко расширяется — для решения многих дополнительных задач потребуется только добавлять компоненты в общий конвейер.

Сравнение двух архитектур

- Вторая архитектура, несмотря на выигрыш в инкрементальности, проигрывает в целом
- Основная ее проблема — слишком специфически построенный компонент-обработчик букв
- Необходимость изменить его в нескольких сценариях показывает, что нужно объединить обработчик букв и обработчик конца слов в единый компонент, выдающий слова целиком, после чего полученная архитектура не будет ничем уступать исправленной первой

Примеры шаблонов

- Abstract Factory - паттерн, позволяющий изменять поведение системы, варьируя создаваемые объекты, при этом сохраняя интерфейсы
- Adapter - паттерн, позволяющий преобразовать интерфейс объекта к тому, который требует клиент.

Примеры шаблонов

- Builder - паттерн, позволяющий абстрагировать процесс создания комплексных систем, путем выделения и обобщения классов, отвечающих за создание частей
- Bridge - паттерн, позволяющий отделить интерфейс от реализации и изменять их независимо

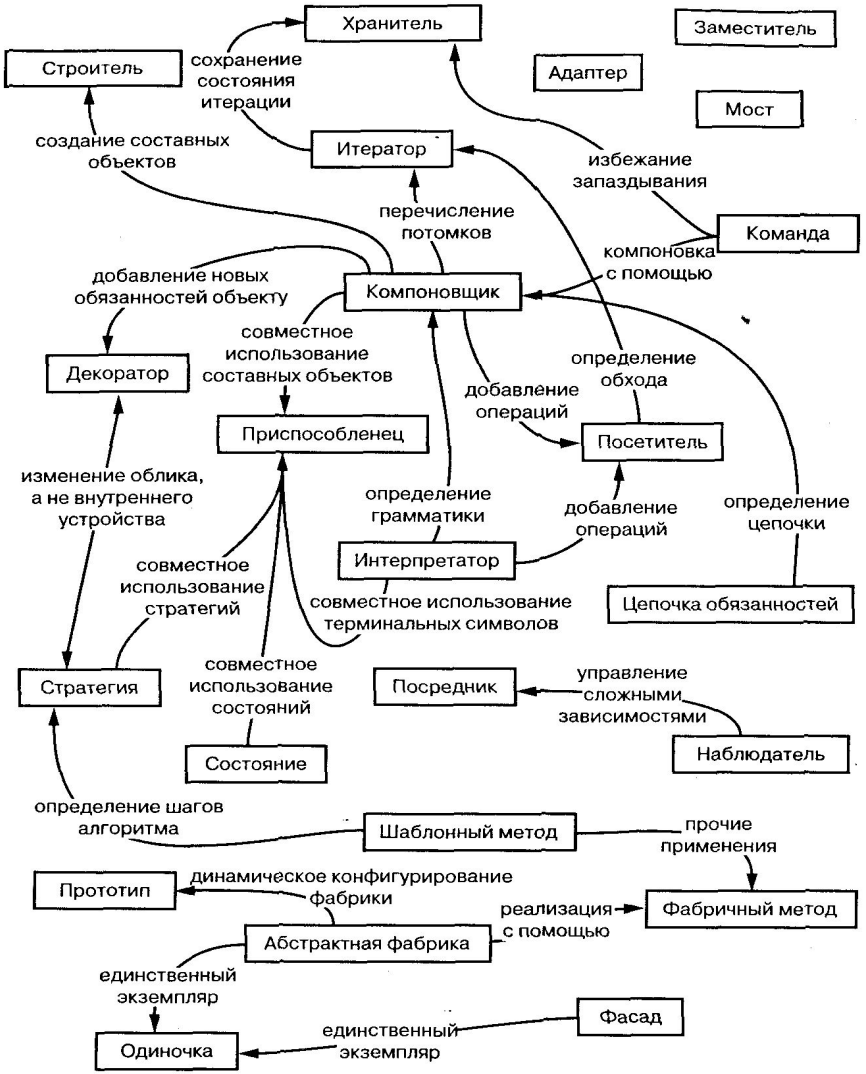
Примеры шаблонов

- Command - паттерн, инкапсулирующий запрос как объект, позволяя более гибко работать с запросами (параметризовать, архивировать, наделять поведением)
- Decorator - паттерн, позволяющий динамически добавлять обязанности объекту, путем включения его в "конверт", обладающий совместимым интерфейсом

Примеры шаблонов

- Facade - паттерн, позволяющий скрыть сложность системы путем сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы
- Другие примеры шаблонов приведены в документе «Каталог шаблонов»

Отношения между шаблонами проектирования



Конец лекции