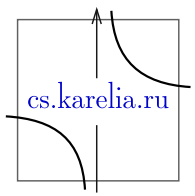
Петрозаводский государственный университет
Институт Математики и информационных технологий
Кафедра Информатики и математического обеспечения

Проектирование программных систем. Шаблоны архитектуры.

Лектор:

к.т.н., доцент Богоявленский Ю. А.
ybgv@cs.karelia.ru



Разделы

- Процесс синтеза программной системы. Проектирование архитектуры
- Шаблоны архитектуры с хранилищем данных и клиент-сервер
- Шаблоны многоуровневой архитектуры. Архитектура канала и фильтра
- Шаблоны управления

Процесс синтеза программной системы. Проектирование архитектуры. Содержание

- Анализ и синтез
- Процесс синтеза
  - Информационные потоки.
  - Качество ПО.
- Проектирование архитектуры.
  - Этапы.
  - Влияние требуемых характеристик.
  - Блочные диаграммы.
  - Основные виды деятельности
- Структурирование системы
  - Введение.
  - Шаблоны архитектуры.
- Шаблон модель-представление-контролёр.
  - Описание.
  - Схема структуры ПС.

Анализ и синтез

Технологический цикл разработки любого инженерного изделия включает анализ и синтез. Аналогичные действия можно выделить и в разработке программной системы.

Анализ

В ходе анализа ищется ответ на вопрос: «Что должна делать будущая система?» Именно на этой стадии закладывается фундамент успеха всего проекта.

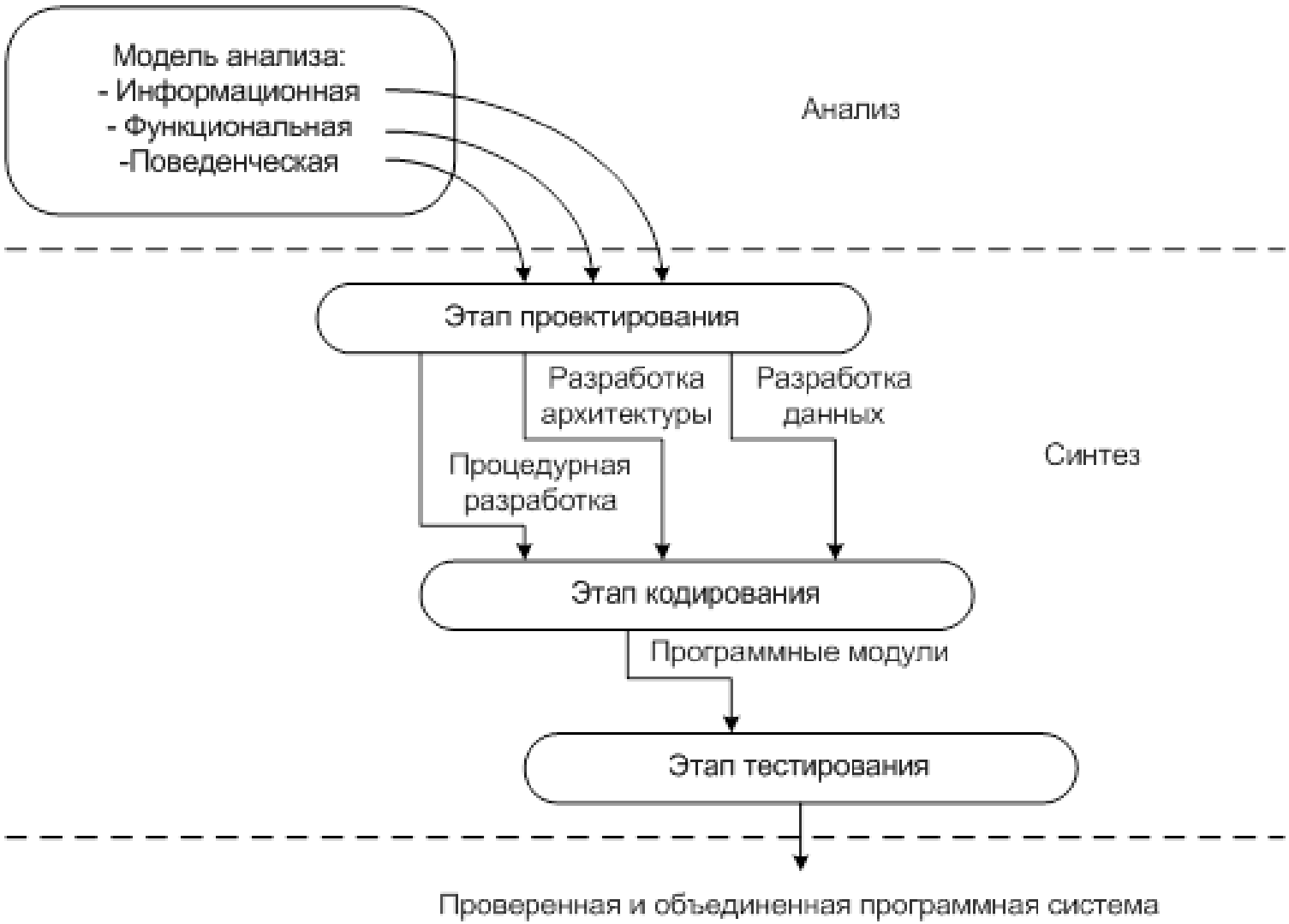
Синтез

В процессе синтеза формируется ответ на вопрос: «Каким образом система будет реализовывать предъявляемые к ней требования?»

Выделяют три этапа синтеза:

- проектирование программной системы;
- кодирование программной системы;
- тестирование программной системы.

Схема процесса синтеза



Процесс синтеза: информационные потоки

Этап проектирования питает требования к ПС, представленные информационной, функциональной и поведенческой моделями анализа.

- **Модели анализа** поставляют этапу проектирования исходные сведения для работы.
- **Информационная модель** описывает информацию, которую, по мнению заказчика, должна обрабатывать ПС
- **Функциональная модель** определяет перечень функций обработки.
- **Поведенческая модель** фиксирует желаемую динамику системы (режимы ее работы).

На выходе этапа проектирования — модель данных, модель архитектуры и модели подсистем архитектуры ПС.

Процесс синтеза: информационные потоки

- **Модель данных** — это результат преобразования информационной модели анализа в структуры данных, которые потребуются для реализации программной системы.
- **Модель архитектуры** выделяет основные подсистемы структуры, фиксирует связи между ними и задает принципы взаимодействия подсистем.
- **Модели подсистем** описывают организацию подсистем, то есть определяют их содержание.

Далее создаются тексты программных модулей, проводится тестирование для объединения и проверки ПС. На проектирование, кодирование и тестирование приходится более 75% стоимости разработки ПС. Принятые здесь решения оказывают решающее воздействие на успех реализации ПС и легкость, с которой ПС будет сопровождаться.

Процесс синтеза: качество ПО

Следует отметить, что решения, принимаемые в ходе проектирования, делают его стержневым этапом процесса синтеза. Важность проектирования можно определить одним словом — **качество**. Проектирование — этап, на котором «выращивается» качество разработки ПС.

Качество

это совокупность характеристик объекта, относящихся к его способности удовлетворить установленные и предполагаемые потребности.

Справедлива следующая аксиома разработки: ***может быть плохая ПС при хорошем проектировании, но не может быть хорошей ПС при плохом проектировании.*** Проектирование обеспечивает нас такими представлениями ПС, качество которых можно оценить. Проектирование — единственный путь, обеспечивающий правильную трансляцию требований заказчика в конечный программный продукт.

Проектирование архитектуры: этапы

Проектирование

это итерационный процесс, при помощи которого требования к ПС транслируются в инженерные представления ПС.

Вначале эти представления дают только концептуальную информацию (на высоком уровне абстракции), последующие уточнения приводят к формам, которые близки к текстам на языках программирования.

Обычно в проектировании выделяют две ступени:

- **Архитектурное проектирование**, в ходе которого формируются абстракции высокого уровня.
- **Детальное проектирование**, в ходе которого уточняются абстракции высокого уровня, добавляются подробности алгоритмического уровня.

Кроме того, во многих случаях выделяют **интерфейсное проектирование**, цель которого — сформировать графический

Проектирование архитектуры: влияние требуемых характеристик

Различным требованиям к интегральным характеристикам системы должны соответствовать разные варианты архитектуры.

- Если главным требованием является **производительность** системы, следует спроектировать архитектуру, которая локализует критические операции в пределах небольшого количества подсистем с минимальным числом взаимодействий между ними. Для сокращения внешних коммуникаций подсистемы должны быть крупными, с большой степенью автономности.
- Для обеспечения максимальной **защищенности** архитектура должна быть многоуровневой, причем самые ценные подсистемы следует размещать на внутренних уровнях, с высокой степенью защиты.

Проектирование архитектуры: влияние требуемых характеристик

- При ориентации на максимальную **безопасность**, поддержку безопасности нужно сосредоточить в одной подсистеме. Это уменьшит стоимость проектирования и позволит применить эффективный механизм проверки надежности.
- Для создания **устойчивости к отказам** в архитектуру вводятся резервные подсистемы. Их наличие позволяет выполнять замену отказавших элементов без остановки системы.
- **Удобство сопровождения** повышается при проектировании архитектуры из малых подсистем, которые легче проверять и обновлять. Для облегчения диагностики ошибок желателен отказ от глобальных структур данных.

Поиск компромиссного решения — типичная задача архитектурного проектирования, поскольку, как правило, требуется максимизация многих параметров системы.

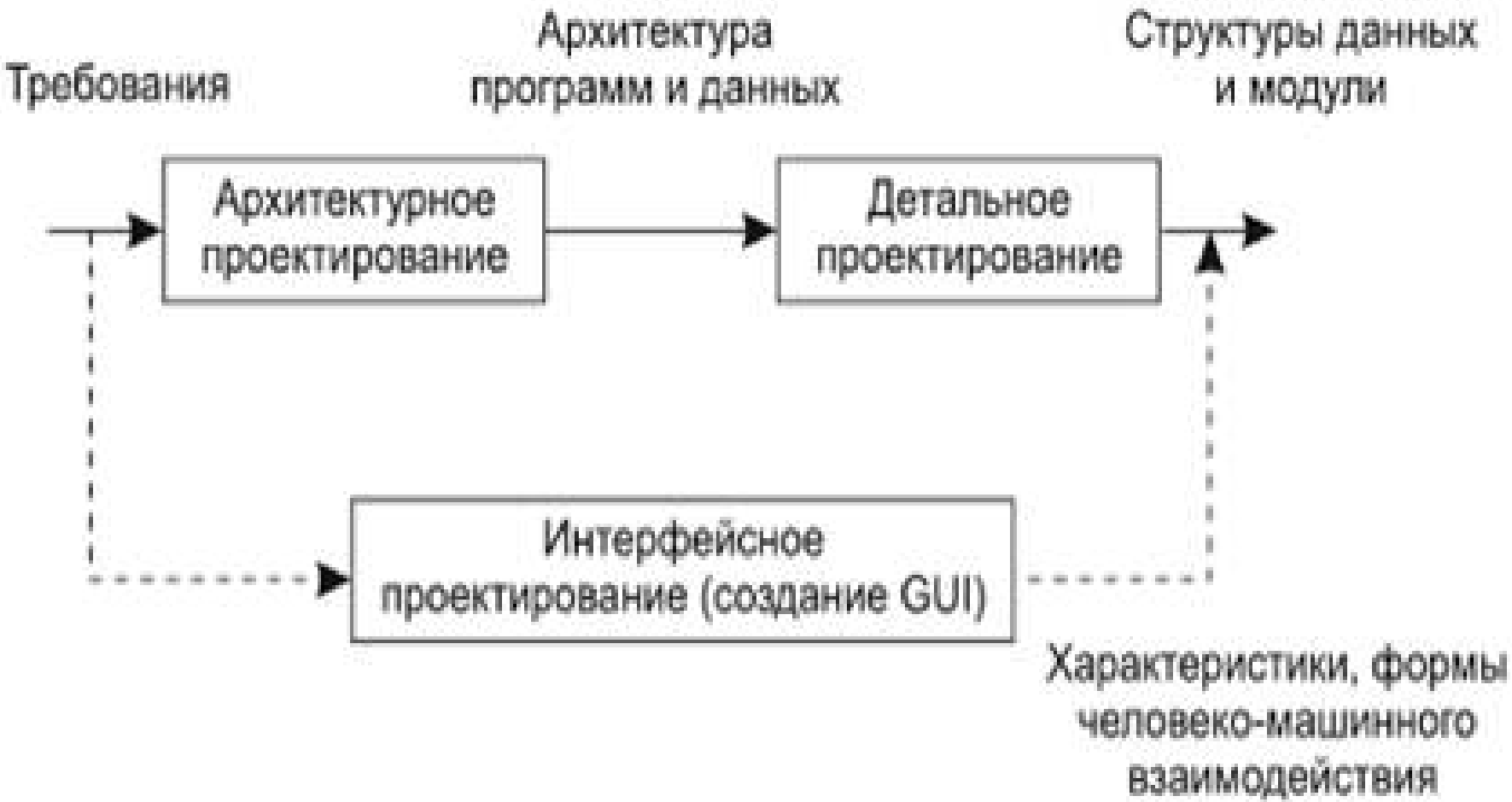
Проектирование архитектуры: блочные диаграммы

Архитектура системы часто моделируется с помощью простых блочных диаграмм. Каждый блок в диаграмме представляет подсистему. Стрелки показывают передачу данных и управляющих сигналов между компонентами. Преимуществами этой диаграммы:

- Она полезна для обсуждения системы с заинтересованными лицами и для планирования проекта, потому что не обременена деталями.
- Не специалист, но заинтересованное лицо сможет понять главные идеи организации системы, не вдаваясь в детали.
- Менеджер видит ключевые компоненты, которые должны быть разработаны, и начинает понимать, каких сотрудников надо привлечь к разработке.

Скудность блочной диаграммы можно компенсировать документированием архитектуры, сформировав подробное архитектурное описание, которое поясняет компоненты системы, их интерфейсы, соединения и облегчает понимание и развитие системы.

Информационные связи процесса проектирования



Проектирование архитектуры: основные виды деятельности

Во время архитектурного проектирования системные архитекторы должны принимать такие решения, которые глубоко затрагивают систему и процесс ее разработки. Основываясь на своих знаниях и опыте, они решают широкий спектр задач. Все эти задачи группируются вокруг трех типов базисной деятельности:

1. **Структурирование системы**. Система разбивается на несколько подсистем, где под подсистемой понимается независимый программный компонент. Определяются взаимодействия подсистем.
2. **Моделирование управления**. Формируется стратегия управления частями системы.
3. **Декомпозиция подсистем на модули**. Каждая подсистема разбивается на модули. Определяются типы модулей и межмодульные соединения.

Структурирование системы: введение

В ходе архитектурного проектирования создается структурная организация системы, которая будет отвечать всем функциональным и нефункциональным требованиям. В ходе этого процесса необходимо учесть следующие моменты:

- Успех процесса зависит от типа разрабатываемой системы, подготовки и опыта системного архитектора, а также от конкретных требований к системе.
- Каждая программная система уникальна, но системы в одной и той же прикладной области часто имеют сходные архитектуры.
- Во встроенных системах и системах для персональных компьютеров обычно имеется только один процессор, и не нужно проектировать распределенную архитектуру.
- Современные большие системы являются распределенными системами, в которых системное программное обеспечение распределено по многим компьютерам.

Структурирование системы: шаблоны архитектуры

Архитектура программной системы может быть основана на определенном архитектурном паттерне.

Архитектурный паттерн (шаблон)

это описание типовой организации системы. Архитектурные паттерны фиксируют сущность архитектуры, которая использовалась в различных программных системах.

Архитектурный паттерн можно рассматривать как обобщенное описание хорошей практики, опробованной и проверенной в различных системах и средах. Таким образом, архитектурный паттерн должен описать системную организацию, которая была успешна в предыдущих системах.

Структурирование системы: шаблоны архитектуры

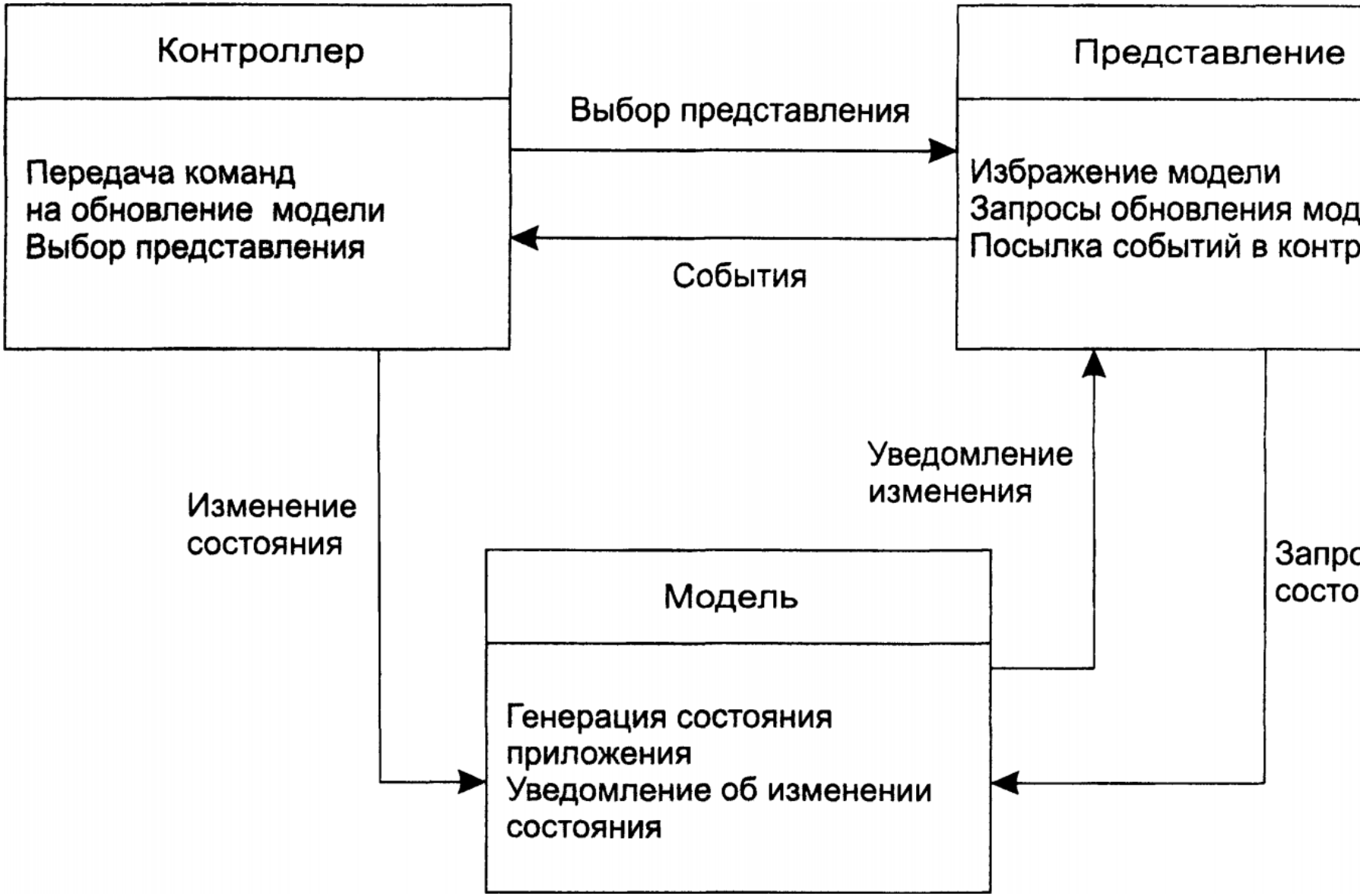
Поскольку паттерн является образцом типового решения, с которым знакомятся и которое могут применить многие архитекторы, его описание должно содержать развернутую характеристику. Обычно описание состоит из следующих разделов:

- **Имя паттерна** - должно характеризовать его суть;
- **Описание** - краткое и понятное описание функциональных возможностей и структуры паттерна, структура поясняется графической диаграммой;
- **Пример** - приводится пример типового применения паттерна, поясняемый диаграммой;
- **Когда используется** - описываются условия применения, дается обобщенная характеристика предметных областей;
- **Преимущества** - перечисляются преимущества применения;
- **Недостатки** - указываются слабые стороны данного паттерн-решения.

MVC: описание

Имя паттерна	Модель-представление-контролер (MVC)
Описание	Отделяет представление системы и взаимодействие с системой от данных системы. Система разделяется на три логических компонента, которые взаимодействуют друг с другом. Компонент ***Модель*** управляет системными данными и операциями над данными. Компонент ***Представление*** отображает данные для пользователя. Компонент ***Контроллер*** взаимодействует с пользователем, инициирует операции в модели и управляет работой представления.
Когда используется	Когда требуется несколько вариантов обработки и представления данных или когда неизвестны требования к взаимодействию и представлению данных
Преимущества	Позволяет изменять данные независимо от их представления. Изменение данных, сделанное в одном представлении, отображается во всех остальных представлениях
Недостатки	Избыточность программного кода при простой модели данных и простых схемах взаимодействия

MVC: схема структуры ПС



Шаблоны архитектуры с хранилищем данных и клиент-сервер. Содержание

Паттерн Хранилище данных

Часто подсистемы разделяют данные, находящиеся в общем хранилище. Как правило, данные оргпнизованы в БД, что требует наличия СУБД.

Архитектурный паттерн хранилища данных описывает организацию взаимодействия подсистем через БД.

- Описание:
  - В нейтральном хранилище находятся все данные системы. Эти данные доступны всем подсистемам. Подсистемы взаимодействуют друг с другом косвенно, через хранилище.
- Когда используется:
  - Когда требуется создавать и долгое время хранить большие объемы информации. Удобно использовать в системах, порядок работы которых определяется данными.
- Недостатки:
  - Трудности размещения хранилища на нескольких компьютерах. Возможно понижение скорости доступа к данным. Проблемы хранилища прямо влияют на всю систему.

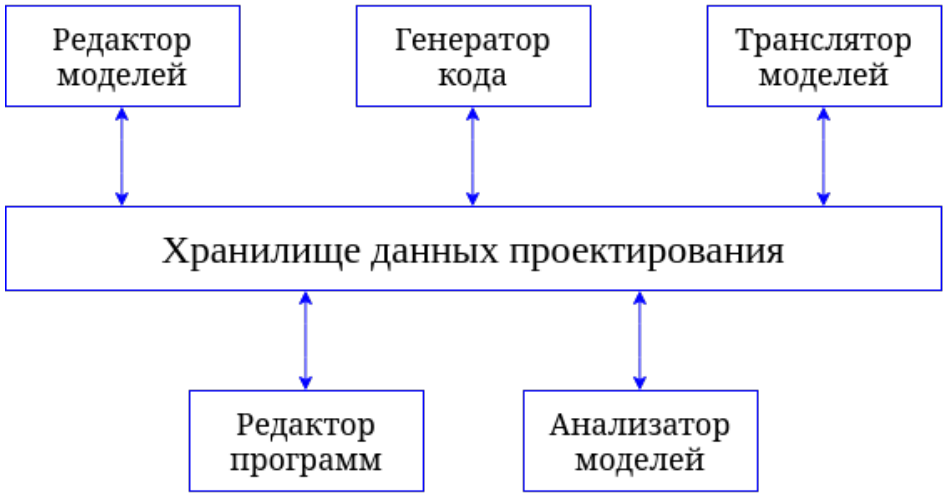
Паттерн Хранилища данных

- Преимущества:

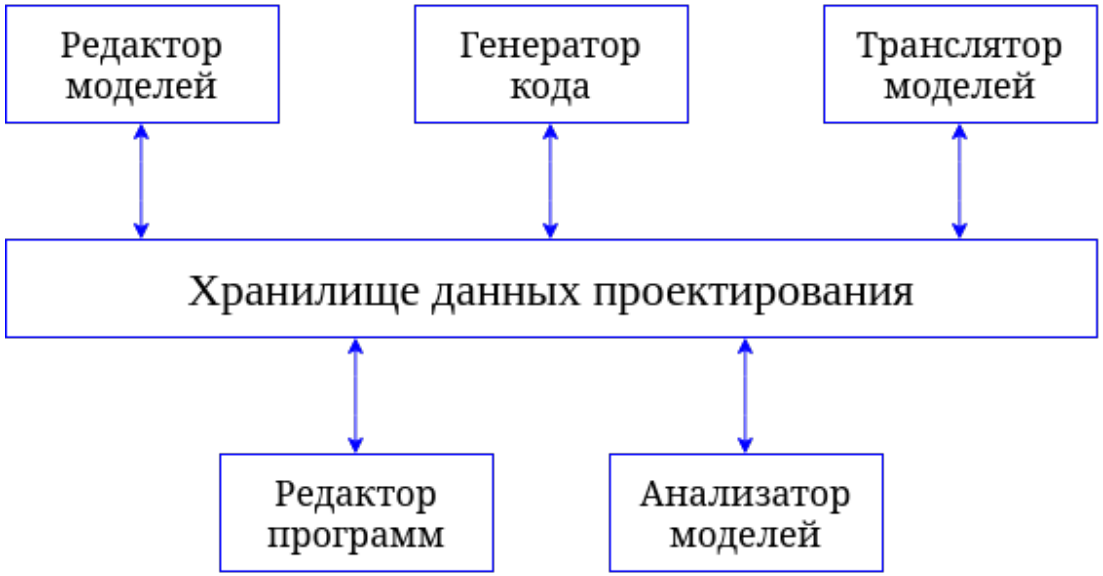
1. Предоставляет подсистемам простую схему для получения устойчиво существующих объектов и управления их жизненным циклом.
2. Убирает необходимость в технической поддержке целостности объектов, разных вариантов технологий СУБД и даже разных источников данных.
3. Скрывает сложность механизма доступа к устойчивым объектам.
4. Обеспечивает функциональную независимость подсистем.

Пример

- Архитектура Case-системы на основе хранилища показана на рисунке. Здесь каждая подсистема генерирует данные, которые доступны другим подсистемам.



Пример



Этот паттерн удобен для систем, где одни компоненты генерируют данные, а другие используют их (например, систем управления, информационных систем, среды для разработки программ и т. д.)

Пример

- На рисунке демонстрируется архитектура системы автоматизации разработки ПО на базе паттерна хранилища. Обрамление хранилища набором утилит позволяет организовать эффективную обработку больших объемов данных, находящихся в совместном использовании.
- Еще одна проблема возникает при необходимости размещения хранилища на нескольких компьютерах, поскольку появляются трудности в обеспечении резервирования и целостности данных.Чаще всего хранилище является ведомым элементом системы, который управляется другими подсистемами.

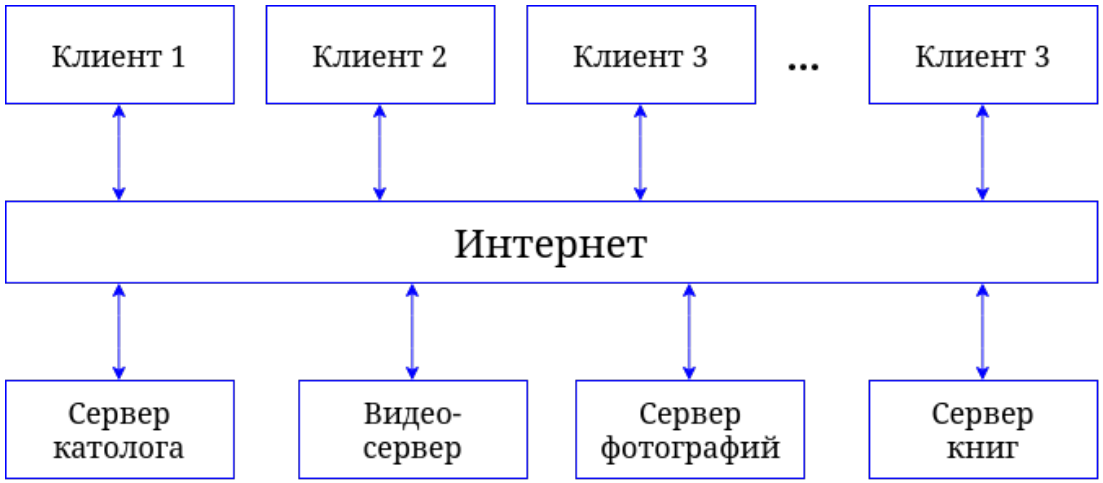
Клиент-серверная архитектура

- Описание:
  - Функциональность системы обеспечивается набором услуг (сервисов). Каждый сервис располагается на своем сервере. Клиенты являются пользователями этих сервисов. Для получения услуги клиент обращается к серверу.
- Когда используется:
  - Когда услуги должны быть доступны из разных мест.
  - Когда требуется гибкий механизм перестройки системы по загружаемым начальным данным.

Клиент-серверная архитектура

- Преимущества:
  - Предоставление клиентам различных услуг через сеть.
  - Устраняется необходимость тиражирования реализации услуг среди серверов.
- Недостатки:
  - Возможно понижение скорости доступа к данным из-за проблем в сети.
  - Поломка сервера лишает клиента услуги.

Клиент-серверная архитектура сетевой библиотеки



Система здесь организована в виде набора сервисов, находящихся на серверах, и клиентов, которые обращаются к серверам и используют сервисы.

Клиент-серверная архитектура сетевой библиотеки

Главные элементы этой архитектуры:

- Набор серверов, предлагающих услуги клиентам (например, серверы услуг печати, серверы для записи и хранения файлов, серверы-переводчики).
- Набор клиентов, которые обращаются к сервисам серверов. В роли клиентов выступают клиентские программы, часто запускаемые одновременно на различных компьютерах.
- Сеть, которая позволяет клиентам обращаться к сервисам. Чаще всего клиент-серверные системы реализованы как распределенные системы, соединенные с помощью интернет-протокола.

«Тонкий» клиент

- В случае «тонкого» клиента вся обработка и управление данными выполняются на сервере. На клиентском компьютере запускаются только функции приема и отображения данных — реализуется пользовательский интерфейс.
- Главный недостаток модели «тонкого» клиента — большая загруженность сервера и сети (все вычисления выполняются на сервере).

«Толстый» клиент

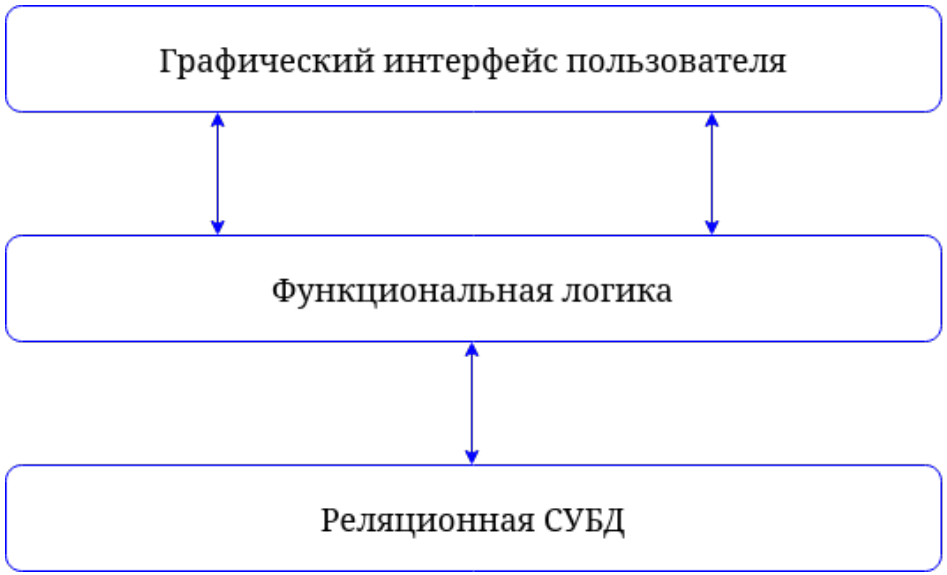
- При «толстом» клиенте сервер только управляет данными. На клиентском компьютере организована обработка данных и взаимодействие с пользователем системы.
- Модель «толстого» клиента максимально использует вычислительную мощность клиентских компьютеров: на них перемещаются и операции обработки, и операции представления. Примером «толстых» клиентов являются банкоматы.

Клиент-серверная архитектура сетевой библиотеки

- Эта система предлагает клиентам электронные версии книг, фильмы и фотографии. В системе имеется несколько серверов, осуществляющих обработку, хранение и выдачу различных типов данных. Аудио- и видеоинформацию следует передавать синхронно, в реальном масштабе времени, что обеспечивает видеосервер. Фотографии позволяет просматривать с высоким разрешением отдельный сервер.
- Сервер каталога обслуживает запросы поиска, а сервер книг — запросы книжных изданий. В качестве клиентов рассматриваются экземпляры веб-браузера пользователей.

Трехъярусная архитектура клиент-сервер

- В обычной, двухъярусной системе клиент-сервер могут возникнуть существенные проблемы с размещением на аппаратуре трех логических ярусов — представления, обработки и хранения данных. Чтобы избежать этих проблем, применяют трехъярусную архитектуру клиент-сервер:



В этой архитектуре ярусам представления, обработки и хранения данных соответствуют отдельные подсистемы.

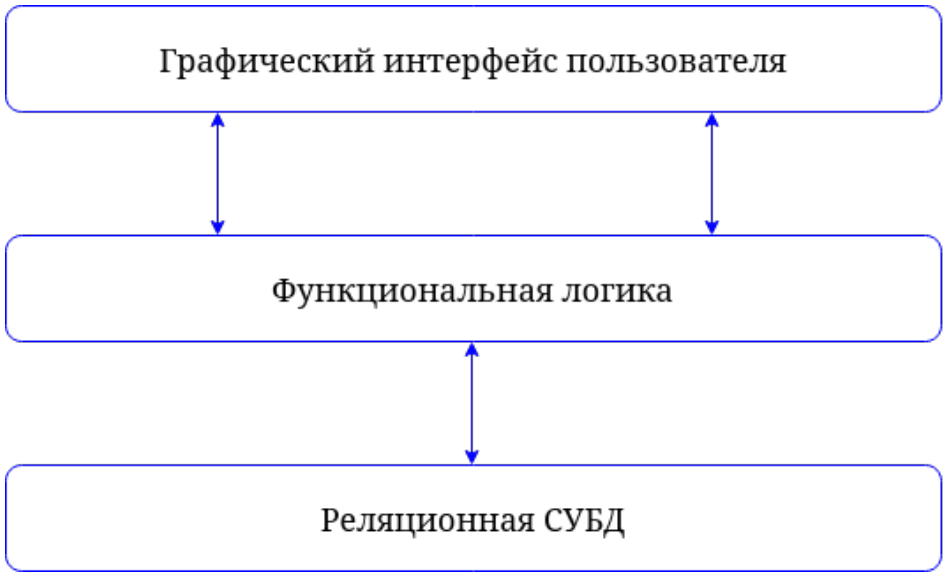
Трехъярусная архитектура клиент-сервер

- Ярус графического интерфейса пользователя запускается на машине клиента.
- Функциональную логику образуют модули, осуществляющие функциональные обязанности системы. Этот ярус запускается на сервере приложения.
- Реляционная СУБД хранит данные, требуемые ярусу функциональной логики. Этот ярус запускается на втором сервере — сервере базы данных.
- Если же требования к системе возрастут, достаточно просто разделить эти логические серверы и выполнять их на разных машинах.

Трехъярусная архитектура клиент-сервер

Преимущества трехъярусной модели:

- Упрощается такая модификация яруса, которая не влияет на другие ярусы.
- Отделение прикладных функций от функций управления БД упрощает оптимизацию всей системы.



Шаблоны многоуровневой архитектуры. Архитектура канала и фильтра. Содержание

- Многоуровневая архитектура.
  - Описание
  - Реализация
  - Пример
  - Достоинства
  - Недостатки
- Архитектура канала и фильтра.
  - Описание
  - Пример
  - Достоинства
  - Недостатки

Многоуровневая архитектура

Имя	**Многоуровневая архитектура**
Описание	Система организуется в виде набора уровней. С каждым уровнем связывается определенная функциональность. Каждый уровень предлагает услуги вышестоящему уровню и использует услуги нижестоящего уровня. Структуру паттерна поясняет рис. 6. 8
Пример	Архитектура многоуровневой библиотеки диссертационного фонда показана на рис. 6.9
Когда используется	1. Когда разработка обеспечивается чередой команд, причем каждая команда создавала функциональность одного уровня. 2. Когда новые возможности создаются на базе существующих систем. 3. Когда требуется многоуровневая защищенность
Преимущества	1. Позволяет замещать целые уровни (при условии сохранения интерфейса). 2. Для повышения надежности в каждый уровень можно добавить дополнительные возможности (например, аутентификацию)
Недостатки	1. Трудно обеспечить ясное разделение уровней. Текущий уровень может взаимодействовать не с соседом снизу, а с более низкими уровнями. 2. Возможно понижение производительности, поскольку запрос услуги может последовательно обрабатываться на каждом уровне

Многоуровневая архитектура

Паттерн многоуровневой архитектуры предлагает организовать функциональность в виде отдельных уровней

Каждый уровень реализуется с использованием средств и сервисов, обеспечиваемых уровнем, который расположен непосредственно под ним

Используется когда:

1. разработка обеспечивается чередой команд, каждая команда создавала функциональность одного уровня;
2. новые возможности создаются на базе существующих систем;
3. требуется многоуровневая защищенность.

Реализация многоуровневой архитектуры

Многоуровневая архитектура

- Нижний уровень - средства системной поддержки (взаимодействие с операционной системой, база данных);
- Второй уровень - средства прикладной функциональности и сервисные утилиты, обслуживающие разные подсистемы приложения;
- Третий уровень - управляет пользовательским интерфейсом и обеспечивает авторизацию пользователя;
- Верхний уровень - предлагает средства пользовательского интерфейса.

Реализация многоуровневой архитектуры

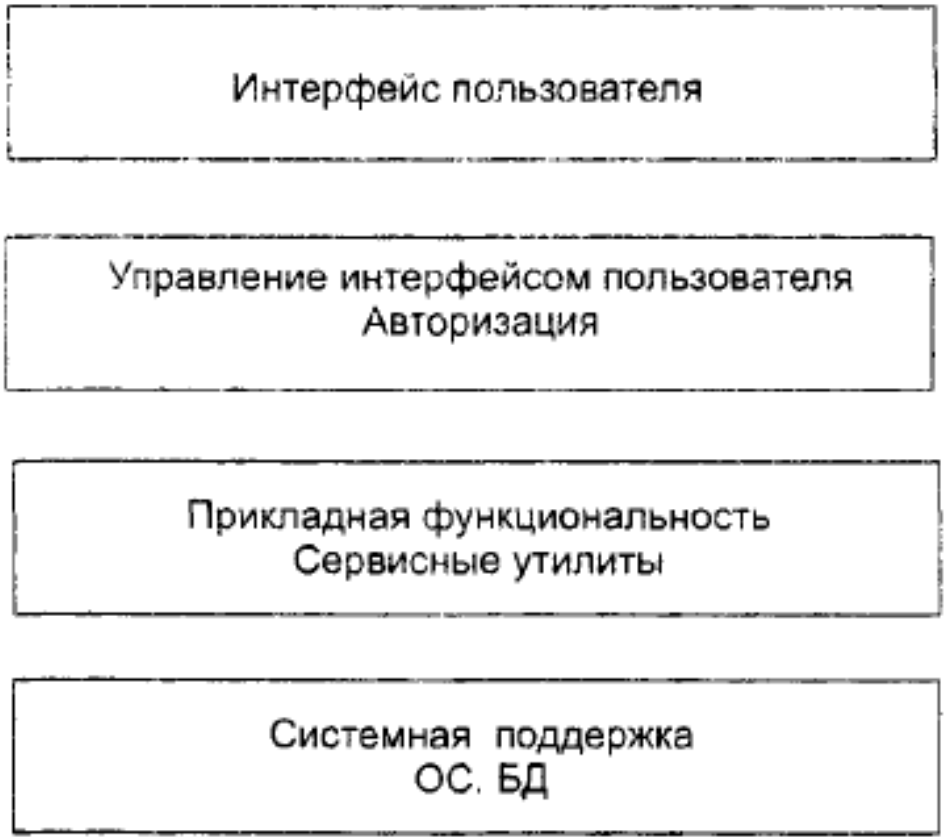


Рис. 6.8. Многоуровневая архитектура

Архитектура библиотеки диссертационного фонда

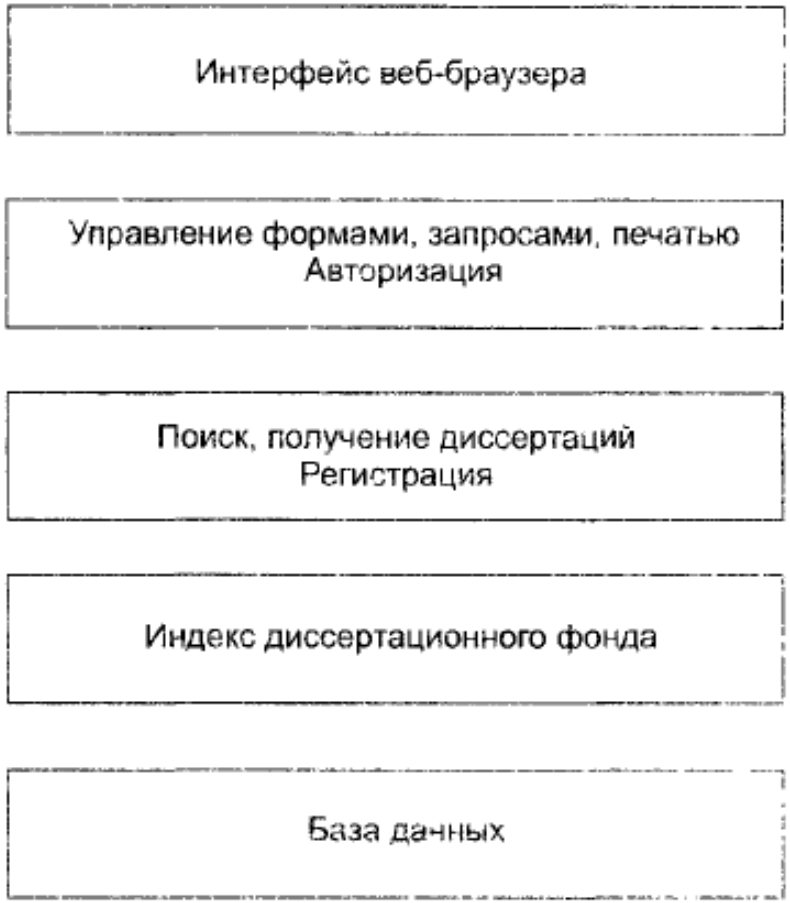


Рис. 6.9. Архитектура библиотеки диссертационного фонда

Достоинства архитектуры

- Изменяемость
  - изменения могут быть легко локализованы: при сохранении интерфейса можно модифицировать содержание целого уровня;
  - изменение интерфейса уровня затрагивает лишь смежный уровень.
- Переносимость
  Перенос системы в другую операционную среду требует замены только нижнего уровня

Недостатки архитектуры

- Трудно обеспечивать ясное разделение уровней: текущий уровень может взаимодействовать не с соседом снизу, а с более низкими уровнями
- Возможно понижение производительности, поскольку запрос услуги может последовательно обрабатываться на каждом уровне

Архитектура канала и фильтра

Имя	**Канал и фильтр**
Описание	Обработка данных в системе организуется с помощью обрабатывающих компонентов (фильтров) — преобразователей. Каждый фильтр выполняет один тип преобразования данных. Для обработки между компонентами устанавливаются потоки данных (каналы)
Пример	Архитектура системы угловой стабилизации ЛА на основе паттерна канала и фильтра показана на рис. 6.10
Когда используется	Когда создается система обработки, управляемая потоком данных. Процесс обработки образуется последовательностью этапов, в которой результаты предыдущего этапа являются входными данными последующего этапа
Преимущества	1. Простота системы. 2. Поддержка механизма повторного использования. 3. Прозрачное наращивание количества преобразований
Недостатки	Должен быть согласован формат данных, перемещаемых между взаимодействующими преобразователями. Каждый преобразователь должен «разбирать» входные данные и «собирать» результаты в соответствии с согласованным форматом. Это увеличивает накладные расходы. Нельзя применять преобразователи с несовместимыми структурами данных

Архитектура канала и фильтра

Паттерн канала или фильтра предлагает рассматривать архитектурную структуру как воплощение диаграммы потоков данных: обработка выполняется функциональными преобразователями (именуются фильтрами), данные между ними переносятся по каналам.

Каждый фильтр выполняет один тип преобразования данных

Когда используется?

При создании системы обработки, управляемой потоком данных. Процесс обработки - последовательность этапов: результаты предыдущего этапа - входные данные последующего

Архитектура канала и фильтра

- Потоки данных перемешаются от одного фильтра к другому и преобразуются по мере движения по последовательности фильтров;
- Каждый шаг обработки - это преобразование, реализуемое фильтром;
- Фильтр выделяет из потока только те данные, которые может обработать;
- Сформированный результат фильтр возвращает в поток;
- Потоки входных данных перемещаются через эти фильтры до тех пор, пока не будут преобразованы в выходные данные

Пример: архитектура системы угловой стабилизации ЛА

Использование системы в приложении с пакетной обработкой.

Система управляет угловым положением летательного аппарата (ЛА).

С измерительных устройств поступает информация о текущих значениях углов по трем каналам.

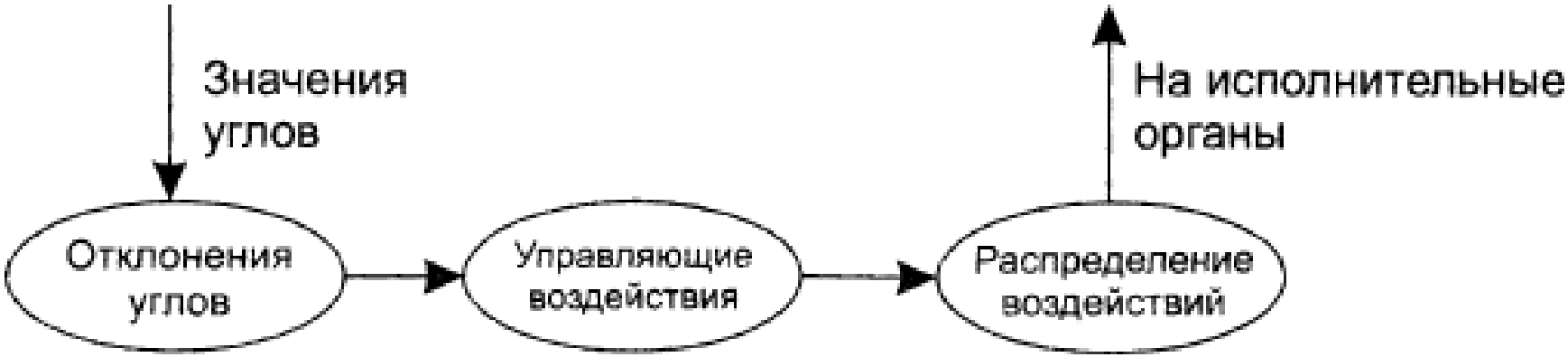


Рис. 6.10. Архитектура системы угловой стабилизации ЛА

Достоинства и недостатки паттерна

К достоинствам относятся:

- простота системы;
- поддержка механизма повторного использования;
- прозрачное наращивание количества преобразований.

Недостаток - нужен формат данных, распознаваемый всеми фильтрами.

Каждое преобразование следует:

- либо согласовывать со смежными фильтрами форматы данных,
- либо иметь стандартный формат для всех данных Каждый фильтр должен выполнять грамматический разбор входных данных, синтезировать выходные данные в соответствующем формате (вычислительная нагрузка возрастает)

В систему нельзя интегрировать фильтры, работающие с несовместимыми форматами данных.

Заключение

Диалоговые системы на основе каналов и фильтров трудны для написания из-за потребности представления данных в виде потока.

- Графическим интерфейсам пользователя присущи более сложные форматы и стратегия управления, основанная на событиях.
- Это трудно транслировать в форму потоков данных.

Шаблоны управления. Содержание

- Моделирование управления.
- Паттерны централизованного управления, паттерн вызов-возврат.
- Паттерн вызов-возврат.
- Паттерн менеджера.
- Паттерны событийного управления, паттерн широковещательного управления.
- Паттерн широковещательного управления.
- Паттерн управления на основе прерываний.

Моделирование управления

Структурная организация системы показывает составляющие ее части — внутренние подсистемы. Для того чтобы подсистемы функционировали как единое целое, ими надо управлять. В структурных моделях нет никаких сведений по управлению, поэтому архитектор должен ввести модель управления, которая дополняла бы имеющуюся модель структуры. В моделях архитектурного управления проектируется поток управления между подсистемами.

Известны два основных типа управления в программных системах: **централизованное управление и событийное управление.**

Типовые решения управления представляются **паттернами управления**.

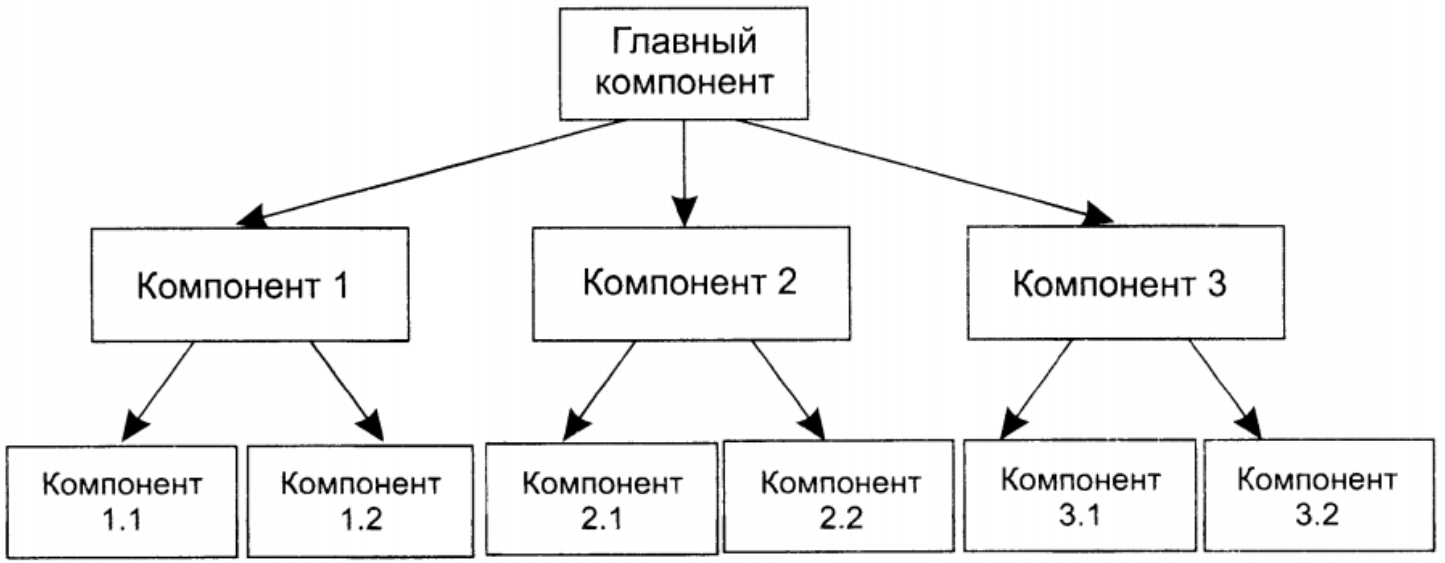
Паттерны централизованного управления, паттерн вызов-возврат

В паттернах централизованного управления одна из подсистем назначается главной и управляет работой других подсистем. Различают две разновидности паттернов, зависящие от режима работы управляемых подсистем (последовательная работа, параллельная работа).

Паттерн вызов-возврат. Это известная схема организации вызова программных компонентов «сверху вниз», в которой управление начинается на вершине иерархии компонентов и через вызовы передается на более низкие уровни иерархии. Данный паттерн применим только в системах с последовательным режимом обработки.

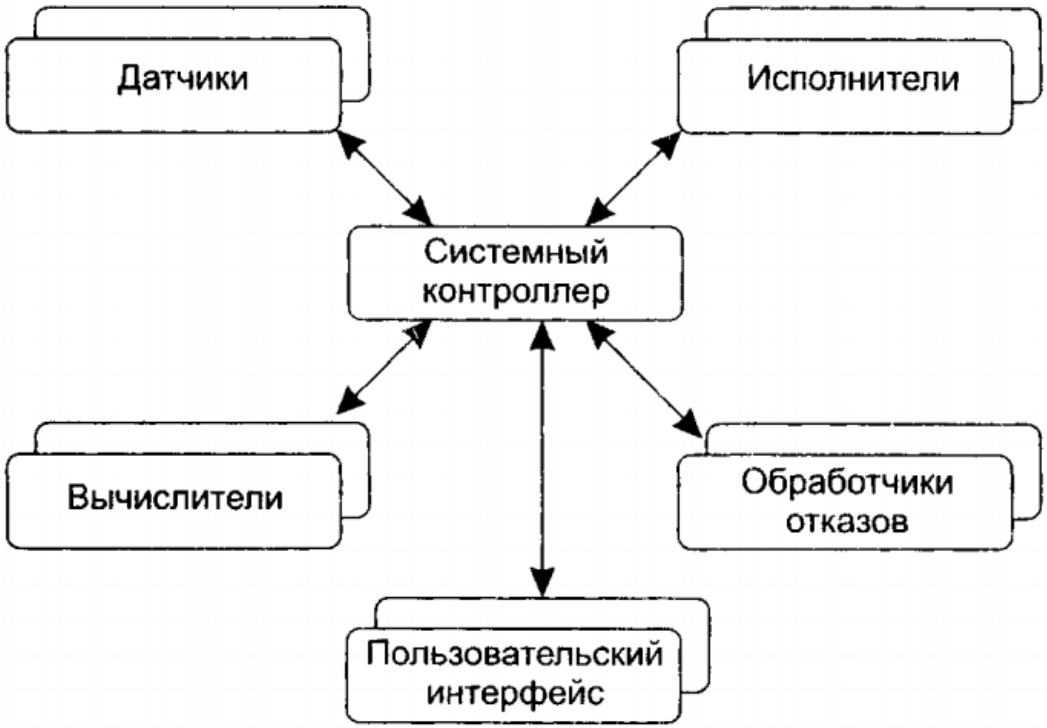
Паттерн вызов-возврат

Имя	**Вызов-возврат**
Описание	Вызов компонентов осуществляется «сверху вниз», то есть управление начинается на вершине иерархии компонентов и через многократные вызовы передается на нижние уровни иерархии
Пример	Иллюстрация организации управления приведена на рис. 6.11
Когда используется	Применима только в системах последовательной обработки, то есть в таких системах, в которых процессы должны протекать последовательно
Преимущества	Простой анализ потоков управления. Такие системы легче проектировать и тестировать
Недостатки	Сложно обрабатывать исключительные ситуации



Паттерн менеджера

Паттерн менеджера. Применяется в системах параллельной обработки. Один компонент назначается менеджером и управляет запуском, финализацией и координацией работы других компонентов. Работа «другого» компонента может протекать параллельно работе третьего компонента.



Система реального времени.

Паттерны событийного управления, паттерн широковещательного управления

При централизованном подходе управление системой обычно зависит от ее состояния. При событийном подходе управление системами основано на внешних событиях (событиях внешней среды) и на аварийных событиях, возникших внутри системы. Рассмотрим два паттерна событийного управления системами.

Паттерн широковещательного управления. Здесь каждая подсистема уведомляет обработчика о своем интересе к конкретным событиям. Когда событие происходит, обработчик пересылает его подсистеме, которая может обработать это событие. Функции управления в обработчик не встраиваются.

Паттерн широковещательного управления

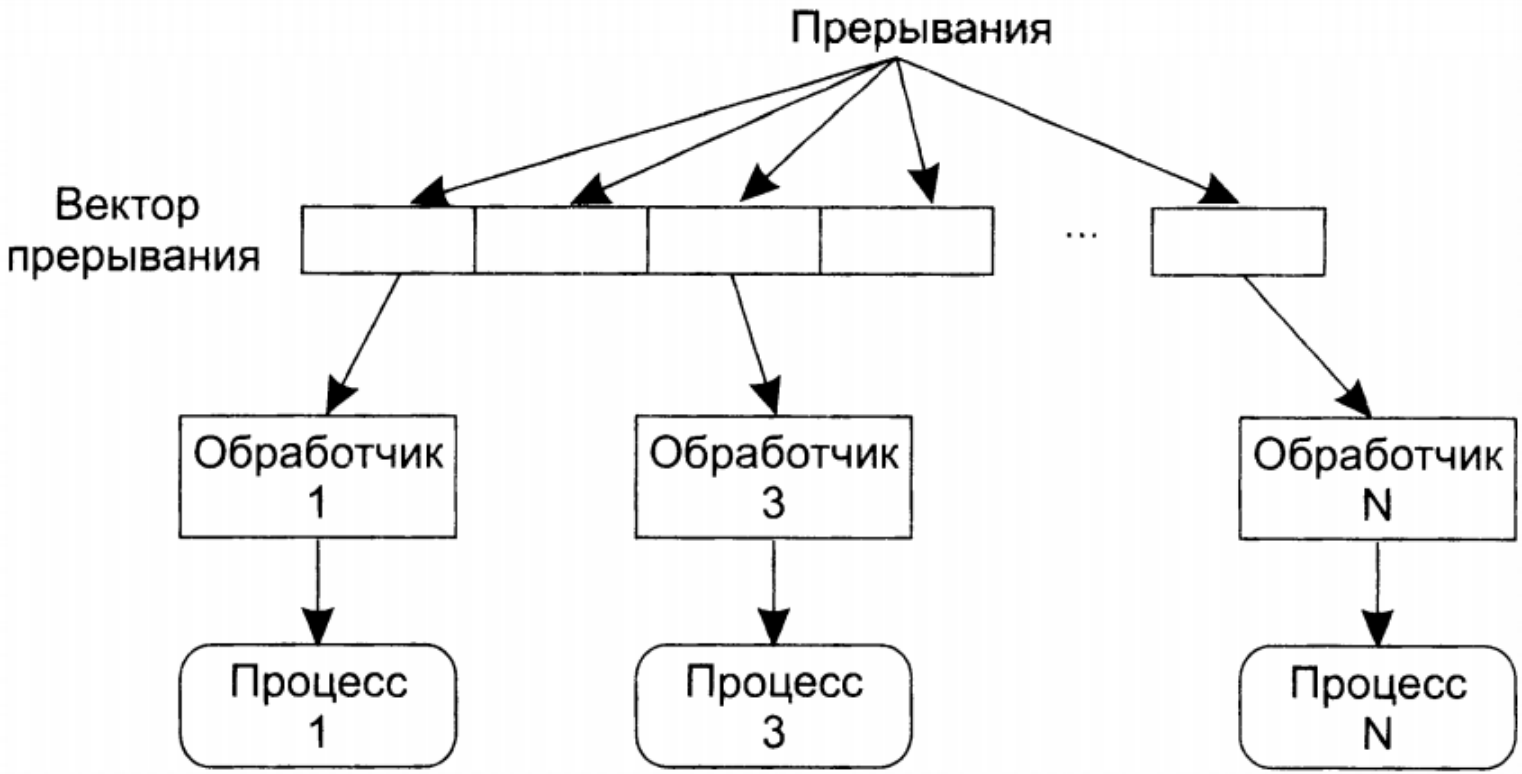
Имя	Широковещательное управление
Описание	Все события и сообщения поступают в обработчик, который передает их конкретным адресатам. Адресатом является подсистема, предварительно объявившая о своем интересе к событию. Адресат, который обрабатывает данное событие, отвечает на него.
Когда используется	Данный подход эффективен при сетевой интеграции подсистем, распределенных на разные компьютеры.
Преимущества	Простота изменения системы, легкость ее размещения.
Недостатки	Возможны конфликты в реакции на внешние события.

Паттерн управления на основе прерываний

Предполагается применение набора обработчиков, оперативно реагирующих на прерывания — внешние события.

Имя	Управление на основе прерываний
Описание	В системе релизован набор обработчиков. Каждая категория прерываний обслуживается своим обработчиком. Обработчик регистрирует прерывание и передает управление заранее определенному процессу
Когда используется	В системах реального времени. Можно комбинировать с паттерном менеджер, управляющим при нормальной работе.
Преимущества	Быстрая реакция системы иа внешние события.
Недостатки	Система сложна в программировании и проверке. При тестировании системы затруднительно имитировать все прерывания. Число прерываний ограничено используемой аппаратурой (после достижения предела, связанного с аппаратными ограничениями, никакие другие прерывания не обрабатываются)

Паттерн управления на основе прерываний



Система, управляемая прерываниями.

Спасибо за внимание.