Программная инженерия

4 Раздел. Модели качества процессов конструирования ПО

5 Раздел. Архитектура ПО

Расписание:

11.01 Пн	14:30-18:25	Г-2082	Лекция
12.01 Вт	14:30-18:25	Г-2082	Лекция
13.01 Ср	14:30-18:25	Г-2023	Лабораторная работа
14.01 Чт	14:30-18:25	Г-2023	Лабораторная работа
21.01 Чт	18:30-21:40	Г-2023	Экзамен

4. Модели качества процессов конструирования ПО

По литературе 1, 5

+

Статья: Модели зрелости процесса тестирования ПО. Вячеслав Панкратов http://www.osp.ru/os/2007/02/4108132/

Основные тезисы:

- Типичная разработка ПО в России была долгое время ориентирована на программистов-одиночек
- Интереса к индустриальному производству не было из-за высокой стоимости и низкого платежеспособного спроса на сложные программные комплексы(информационные системы)
- Разработка ПО велась спонтанно – не уделялось должного внимания организации самого процесса: планированию, тестированию, межгрупповому взаимодействию, управлению конфигурацией.
- Качество ПО напрямую зависит от качества процесса его производства. Управляя процессом производства и контролируя показатели эффективности всех его технологических этапов, можно влиять на качество производимого продукта
- Важно гарантировать высокое качество вашего процесса конструирования ПО. Такую гарантию дает сертификат качества процесса, подтверждающий его соответствие принятым международным стандартам.

Основные тезисы:

- Подобные международные стандарты дают представление (описывают) свою модель обеспечения качества
- Примеры: ISO9001:2000, ISO/IEC15504 и модель зрелости процесса конструирования ПО (Capability Maturity Model — CMM) Института программной инженерии при американском унив-те Карнеги-Меллон.
- Модель стандарта ISO 9001:2000 ориентирована на процессы разработки из любых областей человеческой деятельности.
- Стандарт ISO/IEC 15504 специализируется на процессах программной разработки и отличается более высоким уровнем детализации. Объем этого > 500 страниц. Значительная часть идей ISO/IEC 15504 взята из модели CMM.
- Базовое понятие модели CMM ***зрелость* компании**.
- **Незрелой** называют компанию, где процесс конструирования ПО и принимаемые решения зависят только от таланта конкретных разработчиков ⇒ высока вероятность превышения бюджета или срыва сроков окончания проекта.

Основные тезисы:

- **В зрелой компании** работают ясные процедуры управления проектами и построения программных продуктов. По мере необходимости эти процедуры уточняются и развиваются. Оценки длительности и затрат разработки точны, основываются на накопленном опыте.
- Также в компании имеются и действуют корпоративные стандарты на процессы взаимодействия с заказчиком, процессы анализа, проектирования, программирования, тестирования и внедрения ПП. Все это создает среду, обеспечивающую качественную разработку ПО.
- CMM фиксирует **критерии для оценки зрелости компании** и предлагает рецепты для улучшения существующих в ней процессов.
- В CMM не только сформулированы условия, необходимые для достижения минимальной организованности процесса, но и **даются рекомендации по дальнейшему совершенствованию процессов**.
- В CMM есть 5 уровней зрелости и предусмотрен плавный, поэтапный подход к совершенствованию процессов — можно поэтапно получать подтверждения об их улучшении после каждого уровня зрелости.

Пять уровней зрелости модели CMM

Зрелость технологического процесса - это степень ясности определения, управления, измерения, контроля и выполнения конкретного технологического процесса. Зрелость свидетельствует, с одной стороны, о мощности (richness) процесса программирования в организации, и, с др. стороны, о степени его применимости (адаптируемости) к проектам организации.

CMM разрабатывалась для программной индустрии, которая остается ее основным пользователем, но, в силу своей универсальности, модель находит сегодня применение и для оценки зрелости бизнес-процессов во многих других областях.

Описание уровней(лит-ра 1)

- **Начальный** — процесс разработки не формализован, носит хаотический характер, не может строго планироваться и отслеживаться. Определены лишь немногие из процессов, и успех проектов зависит от конкретных исполнителей.
- **Повторяемый** — для перехода на этот уровень необходимо внедрить формальные процедуры для выполнения основных элементов процесса конструирования; установлены основные процессы управления проектами: отслеживание затрат, сроков и функциональности. Упорядочены некоторые процессы, необходимые для того, чтобы повторить предыдущие достижения на аналогичных проектах. Основное отличие от уровня 1 состоит в том, что выполнение процесса планируется и контролируется.

Описание уровней(лит-ра 1)

- **Определенный** — процессы разработки ПО и управления проектами описаны и внедрены в единую систему процессов компании. Во всех проектах используется стандартный для организации процесс разработки и поддержки программного обеспечения, адаптированный под конкретный проект. Основное отличие от уровня 2 заключается в том, что элементы процесса уровня 3 планируются и управляются на основе единого стандарта компании.
- **Управляемый** — собираются детальные количественные данные по функционированию процессов разработки и качеству конечного продукта. Анализируется значение и динамика этих данных. Основное отличие от уровня 3 состоит в более объективной, количественной оценке продукта и процесса.
- **Оптимизируемый** — постоянное улучшение процессов основывается на количественных данных по процессам и на пробном внедрении новых идей и технологий. Основное отличие от уровня 4 заключается в том, что технология создания и сопровождения программных продуктов планомерно и последовательно совершенствуется.

Область ключевых процессов (ОКП)

Каждый уровень CMM характеризуется *областью ключевых процессов* (ОКП), причем считается, что каждый последующий уровень включает в себя все характеристики предыдущих уровней. Н-р для 3-го уровня зрелости рассматриваются ОКП 3-го уровня, ОКП 2-го уровня и ОКП 1-го уровня. Область ключевых процессов образуют процессы, которые при совместном выполнении приводят к достижению определенного набора целей. Н-р, ОКП 5-го уровня образуют процессы:

- предотвращения дефектов;
- управления изменениями технологии;
- управления изменениями процесса.

Если все цели ОКП достигнуты, компании присваивается сертификат данного уровня зрелости. Если хотя бы одна цель не достигнута, то компания не может соответствовать данному уровню CMM.

Примерные вопросы по CMM

- С какой целью разрабатывалась CMM?
- Какие ОКП характеризуют каждый уровень?
- Методы оценивания зрелости?
- Какие шаги должна предпринять компания чтобы перейти от одного уровня зрелости к другому?
- Как(где) в России можно получить сертификат? Какие шаги надо предпринять и сколько это стоит?
- Для чего нужно получать официальное подтверждение соответствия одному из уровней ?
- Примеры внедрения этой модели в российских компаниях. Может быть статистика по странам.
- Что мешает (в России) компаниям достигнуть более высокого уровня зрелости?
- Как взаимосвязана методология разработки(н-р RUP) и уровень зрелости компании?
- Достоинства и недостатки CMM.

5. Архитектура ПО

По 10 разделу 2 литературы

Архитектурное проектирование

Архитектурным проектированием называют первый этап процесса проектирования, на котором определяются подсистемы, а также структура управления и взаимодействия подсистем.

Целью архитектурного проектирования является описание *архитектуры программного обеспечения.*

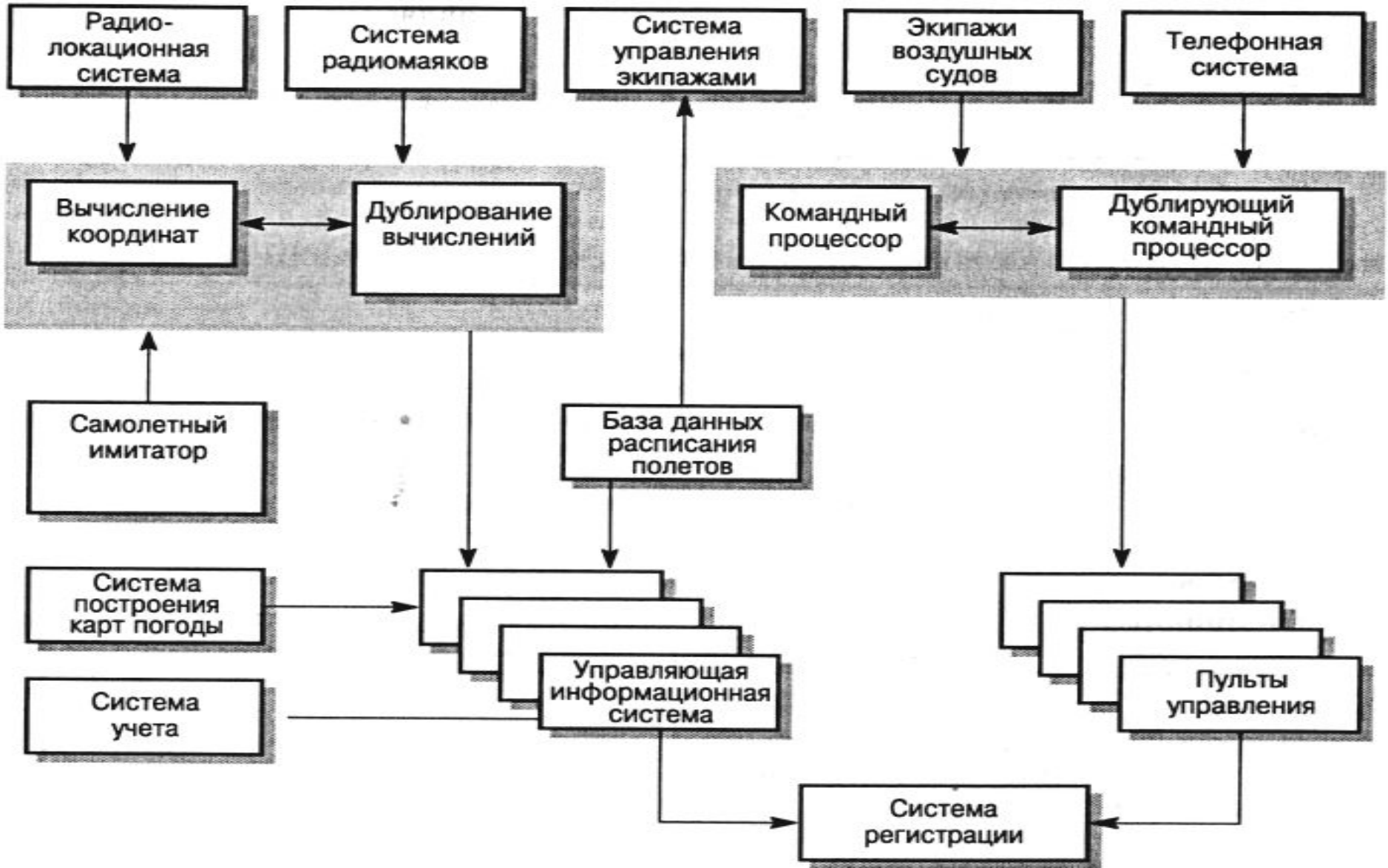
Архитектурное проектирование служит соединяющим звеном между процессом проектирования и процессом разработки требований к создаваемой системе.

Архитектурная декомпозиция системы необходима для структуризации и организации системной спецификации.

Модель системной архитектуры часто является отправной точкой для создания спецификации различных частей системы.

В процессе архитектурного проектирования разрабатывается базовая структура системы, т.е. определяются основные компоненты системы и взаимодействия между ними.

Пример архитектурной декомпозиции - система управления воздушными полетами. Архитектура система разбивается на несколько взаимодействующих подсистем.



Этапы архитектурного проектирования

1. ***Структурирование системы.*** Программная система структурируется в виде совокупности относительно независимых подсистем. Также определяются взаимодействия между подсистемами.
2. ***Моделирование управления.*** Разрабатывается базовая модель управления взаимоотношениями между частями системы.
3. ***Модульная декомпозиция.*** Каждая определенная на первом этапе подсистема разбивается на отдельные модули. Здесь определяются типы модулей и типы их взаимосвязей.

Результат - документ, отображающий архитектуру системы, состоящий из набора графических схем представлений моделей системы с соответствующим описанием. В описании должно быть указано, из каких подсистем состоит система и из каких модулей слагается каждая подсистема.

Понятия подсистема и модуль

Подсистема – это система (т.е. удовлетворяет "классическому" определению "система"), операции (методы) которой не зависят от сервисов, предоставляемых другими подсистемами. Подсистемы состоят из модулей и имеют определенные интерфейсы, с помощью которых взаимодействуют с другими подсистемами.

Модуль – компонент системы, который предоставляет один или несколько сервисов для других модулей. Модуль может использовать сервисы, поддерживаемые другими модулями. Модуль не рассматривается как независимая система. Модули обычно состоят из ряда других, более простых компонентов.

Четких различий между подсистемами и модулями нет

Архитектурные модели.

1. **Статическая структурная модель**, в которой представлены подсистемы или компоненты, разрабатываемые в дальнейшем независимо.
2. **Динамическая модель** процессов, в которой представлена организация процессов во время работы системы.
3. **Интерфейсная модель,** которая определяет сервисы, предоставляемые каждой подсистемой через общий интерфейс.
4. **Модели отношений,** в которых показаны взаимоотношения между частями системы, например поток данных между подсистемами.

Для описания архитектур лучше использовать неформальные модели и системы нотации, например унифицированный язык моделирования UML

Архитектура системы

может строиться в соответствии с определенной архитектурной моделью

- Важно знать эти модели, их недостатки, преимущества и возможности применения.
- Архитектура системы влияет на производительность, надежность, удобство сопровождения и другие характеристики системы. Поэтому модели архитектуры, выбранные для данной системы, могут зависеть от нефункциональных системных требований.

Архитектуру больших систем невозможно описать с помощью какой-либо одной модели. При разработке отдельных частей больших систем можно использовать разные архитектурные модели.

Архитектура и нефункциональные требования к ПО

Производительность. Необходимо уменьшить количество подсистем с критическими операциями и взаимодействием между ними. Лучше использовать крупномодульные компоненты.

Защищенность. Архитектура должна иметь многоуровневую структуру, в которой наиболее критические системные элементы защищены на внутренних уровнях, а проверка безопасности этих уровней осуществляется на более высоком уровне.

Безопасность. За операции, влияющие на безопасность системы, должно отвечать как можно меньше подсистем. Такой подход позволяет снизить стоимость разработки и решает проблему проверки надежности.

Архитектура и нефункциональные требования к ПО

Удобство сопровождения. Архитектуру системы следует проектировать на уровне мелких структурных компонентов, которые можно легко изменять. Программы, создающие данные, должны быть отделены от программ, использующих эти данные. Следует также избегать структуры совместного использования данных.

Надежность. Архитектура с включением избыточных компонентов, чтобы можно было заменять и обновлять их, не прерывая работу системы.

Некоторые из перечисленных требований противоречат друг другу.

Н-р, для того чтобы повысить производительность, необходимо использовать крупномодульные компоненты, в то же время сопровождение системы намного упрощается, если она состоит из мелких структурных компонентов.

Если необходимо учесть оба требования, следует искать компромиссное решение. Один из способов решения подобных проблем - применение различных архитектурных моделей для разных частей системы.

1 этап - структурирование системы

Система разбивается на несколько взаимодействующих подсистем.

три стандартные модели:

- **модель репозитория(репозитария)**
- **модель клиент/сервер**
- **модель абстрактной машины.**

Модель репозитория

Для того чтобы подсистемы, составляющие систему, работали эффективнее, между ними должен идти обмен информацией. Обмен можно организовать двумя способами.

1. Все совместно используемые данные хранятся в **центральной базе данных**, доступной всем подсистемам. Модель системы, основанная на совместном использовании базы данных, часто называют ***моделью репозитория.***
2. Каждая подсистема имеет собственную базу данных. Взаимообмен данными между подсистемами происходит посредством передачи сообщений.

Большинство систем, обрабатывающих большие объемы данных, организованы на основе модели репозитория. *Такая* модель подойдет к приложениям, в которых данные создаются в одной подсистеме, а используются в другой.

Архитектура CASE-средств включает в себя следующие компоненты

(подробнее о компонентах в разделе CASE)

В данной модели репозитории является пассивным элементом, а управление им возложено на подсистемы, использующие данные из репозитория.

Особенности, преимущества и недостатки репозитория

1. Очевидно, что совместное использование больших объемов данных эффективно, поскольку не требуется передавать данные из одной подсистемы в другие.
2. С другой стороны, подсистемы должны быть согласованы с моделью репозитория данных. Это всегда приводит к необходимости компромисса между требованиями, предъявляемыми к каждой подсистеме. Компромиссное решение может понизить их производительность.
3. Подсистемам, в которых создаются данные, не нужно знать, как эти данные используются в других подсистемах.
4. Поскольку в соответствии с согласованной моделью данных генерируются большие объемы информации, модернизация таких систем проблематична.

Особенности, преимущества и недостатки репозитория

5. В системах с репозиторием такие средства, как резервное копирование, обеспечение безопасности, управление доступом и восстановление данных, централизованы, поскольку входят в систему управления репозиторием.
6. В модели репозитория ко всем подсистемам применяется одинаковая политика. Но к разным подсистемам могут предъявляются разные требования.
7. Модель совместного использования репозитория прозрачна: если новые подсистемы совместимы с согласованной моделью данных, их можно непосредственно интегрировать в систему.
8. Сложно разместить репозитории на нескольких машинах, поскольку могут возникнуть проблемы, связанные с избыточностью и нарушением целостности данных.

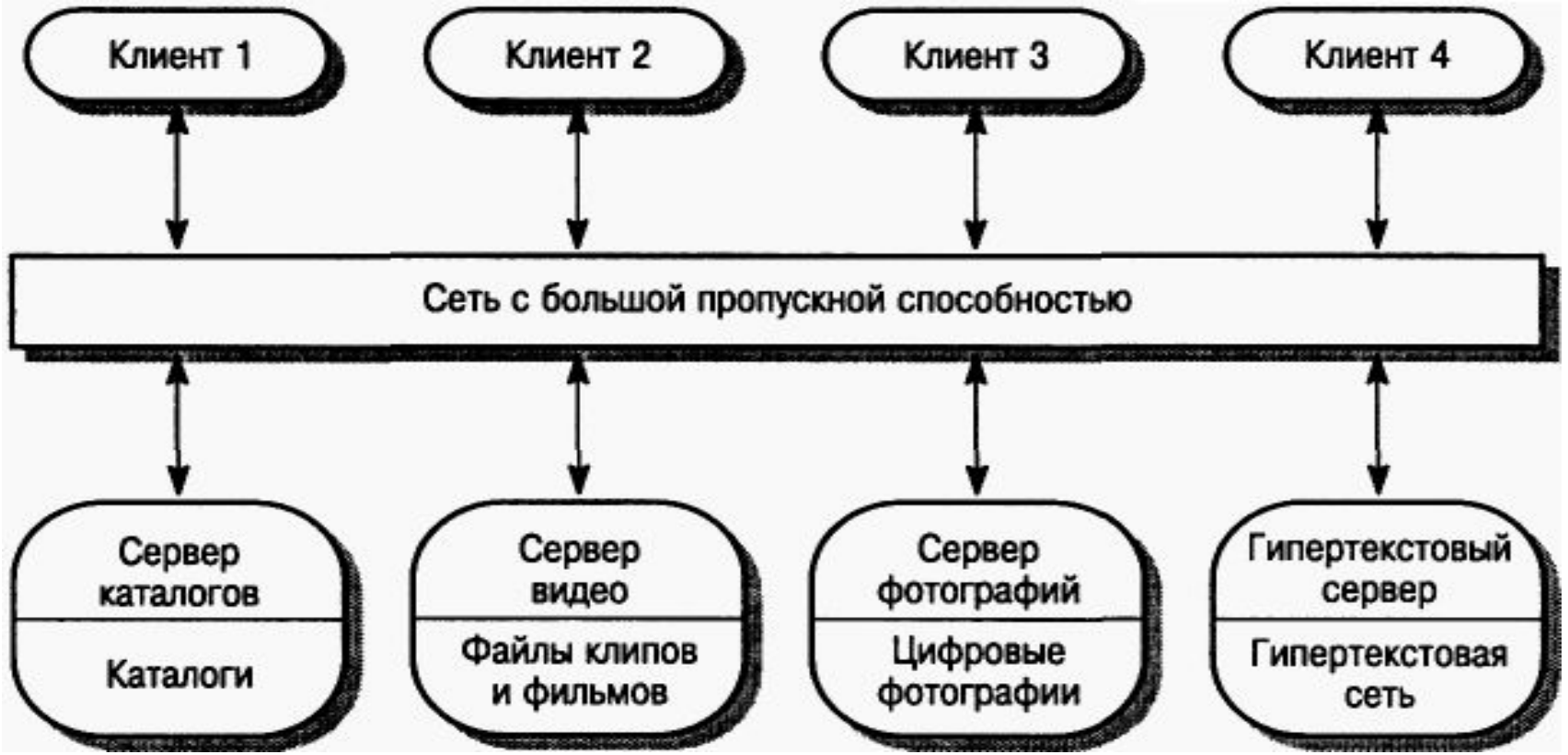
Модель клиент/сервер

– это модель распределенной системы, в которой показано распределение данных и процессов между несколькими процессорами.

Модель включает три основных компонента.

1. Набор автономных серверов, предоставляющих сервисы другим подсистемам.
2. Набор клиентов, которые вызывают сервисы, предоставляемые серверами. В контексте системы клиенты являются обычными подсистемами. Допускается параллельное выполнение нескольких экземпляров клиентской программы.
3. Сеть, посредством которой клиенты получают доступ к сервисам.

Пример использования модели клиент/сервер - Архитектура библиотечной системы фильмов и фотографий



Клиенты должны знать имена доступных серверов и сервисов, которые они предоставляют. В то же время серверам не нужно знать ни имена клиентов, ни их количество. Клиенты получают доступ к сервисам, предоставляемым сервером, посредством удаленного вызова процедур.

Модель клиент/сервер

Наиболее важное преимущество модели клиент/сервер состоит в том, что она является распределенной архитектурой.

Данную модель эффективно использовать в сетевых системах с множеством распределенных процессоров. В систему легко добавить новый сервер и интегрировать его с остальной частью системы или же обновить серверы, не воздействуя на другие части системы.

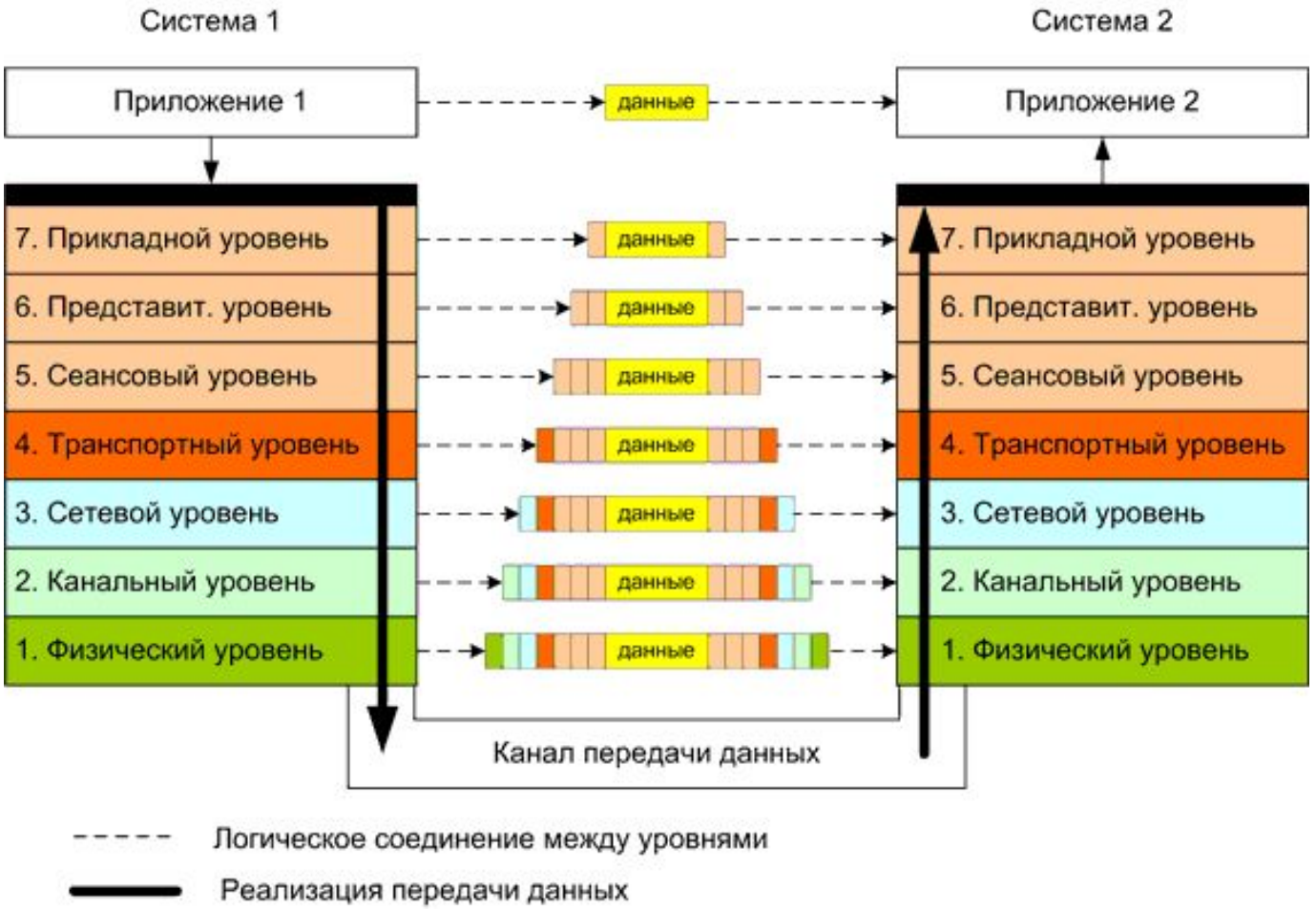
Модель архитектуры абстрактной машины

Модель архитектуры абстрактной машины (иногда называемая многоуровневой моделью) моделирует взаимодействие подсистем.

Она организует систему в виде набора уровней, каждый из которых предоставляет свои сервисы.

Каждый уровень определяет ***абстрактную машину***, машинный язык которой (сервисы, предоставляемые уровнем) используется для реализации следующего уровня абстрактной машины.

Модель абстрактной машины на примере модели OSI



*** OSI (Open System Interconnection - взаимодействие открытых систем) - международная программа стандартизации обмена данными между компьютерными системами на основе семиуровневой модели протоколов передачи данных в открытых системах. Модель предложена Международной организацией по стандартизации ISO (International Standards Organization).**

Особенности, преимущества и недостатки абстрактной машины

- обеспечивает пошаговое развитие систем – при разработке какого-либо уровня предоставляемые им сервисы становятся доступны пользователям.
- такая архитектура легко изменяема и переносима на разные платформы.
- Изменение интерфейса любого уровня повлияет только на смежный уровень.
- Недостатком многоуровневого подхода является довольно сложная структура системы. Основные средства, такие как управление файлами, необходимые всем абстрактным машинам, предоставляются внутренними уровнями. Поэтому сервисам, запрашиваемым пользователем, возможно, потребуется доступ к внутренним уровням абстрактной машины. Такая ситуация приводит к разрушению модели, так как внешний уровень зависит не только от предшествующего ему уровня, но и от более низких уровней.

2 этап – моделирование управления

В модели структуры системы (разрабатывается на 1 этапе) показаны все подсистемы, из которых она состоит.

Для того чтобы подсистемы функционировали как единое целое, необходимо управлять ими.

В структурных моделях нет никакой информации по управлению.

Разработчик архитектуры должен организовать подсистемы согласно некоторой модели управления, которая дополняла бы имеющуюся модель структуры.

В моделях управления на уровне архитектуры проектируется поток управления между подсистемами.

Модель управления дополняет структурные модели.

Два основных типа управления в ПС

Централизованное управление.

Одна из подсистем полностью отвечает за управление, запускает и завершает работу остальных подсистем. Управление от первой подсистемы может перейти к другой подсистеме, однако потом обязательно возвращается к первой.

Управление, основанное на событиях.

Здесь вместо одной подсистемы, ответственной за управление, на внешние события может отвечать любая подсистема. События, на которые реагирует система, могут происходить либо в других подсистемах, либо во внешнем окружении системы.

Централизованное управление

одна из систем управляет работой других подсистем

Модель вызова-возврата.

Это модель организации вызова программных процедур "сверху вниз", управление начинается на вершине иерархии процедур и через вызовы передается на более нижние уровни иерархии. применима только в последовательных системах.

Модель диспетчера.

Применяется в параллельных системах. Один системный компонент назначается диспетчером и управляет запуском, завершением и координированием других процессов системы. Процесс может протекать параллельно с другими процессами.

Модель вызова-возврата

Подобная модель встроена в языки программирования Ada, Pascal и C. Управление переходит от программы, расположенной на самом верхнем уровне иерархии, к подпрограмме более нижнего уровня. Затем происходит возврат управления в точку вызова подпрограммы. За управление отвечает та подпрограмма, которая выполняется в текущий момент; она может либо вызывать другие подпрограммы, либо вернуть управление вызвавшей ее подпрограмме.

Модель диспетчера для системы реального времени (параллельная система)

Центральный контроллер управляет выполнением множества процессов, связанных с датчиками и исполнительными механизмами. Контроллер в зависимости от переменных состояния системы, определяет моменты запуска или завершения процессов. Он проверяет, генерируется ли в остальных процессах информация, для того чтобы затем обработать ее или передать другим процессам на обработку. Обычно контроллер работает постоянно, проверяя датчики и другие процессы или отслеживая изменения состояния, поэтому данную модель иногда называют моделью с обратной связью.

Управление основанное на событиях

управление основано на внешних событиях.

Событие это **не только бинарный сигнал (да-нет). Сигнал может принимать некоторый диапазон значений.**

Различие между событием и обычными входными данными: планирование события выходит за рамки управления процессом, обрабатывающим это событие. Для обработки события подсистеме необходим доступ к информации состояния, однако такая информация обычно не определяется потоком управления.

Модели передачи сообщений.

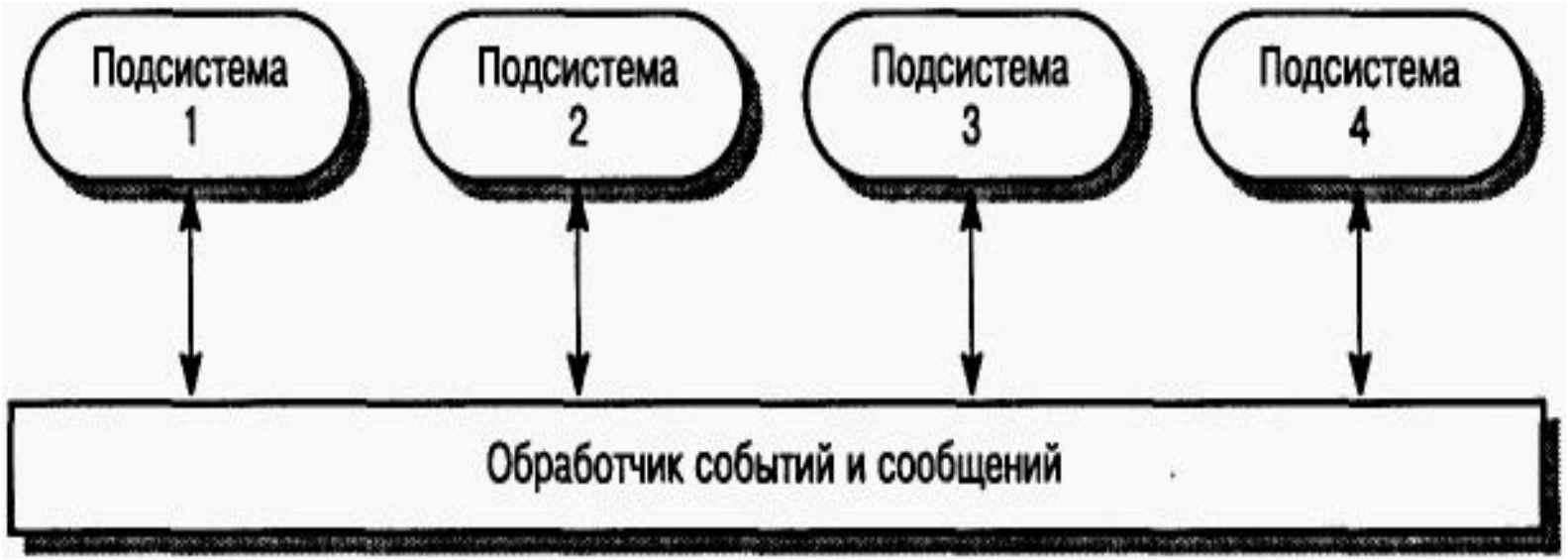
В этих моделях событие представляет собой передачу сообщения всем подсистемам. Любая подсистема, которая обрабатывает данное событие, отвечает на него.

Эффективны при интеграции подсистем, распределенных на разных компьютерах, которые объединены в сеть.

Модели, управляемые прерываниями.

Такие модели обычно используются в системах реального времени(СРВ), где внешние прерывания регистрируются обработчиком прерываний, а обраб-тся другим системным компонентом. Используются в СРВ со строгими временными требованиями.

В модели передачи сообщений подсистемы реагируют на определенные события.



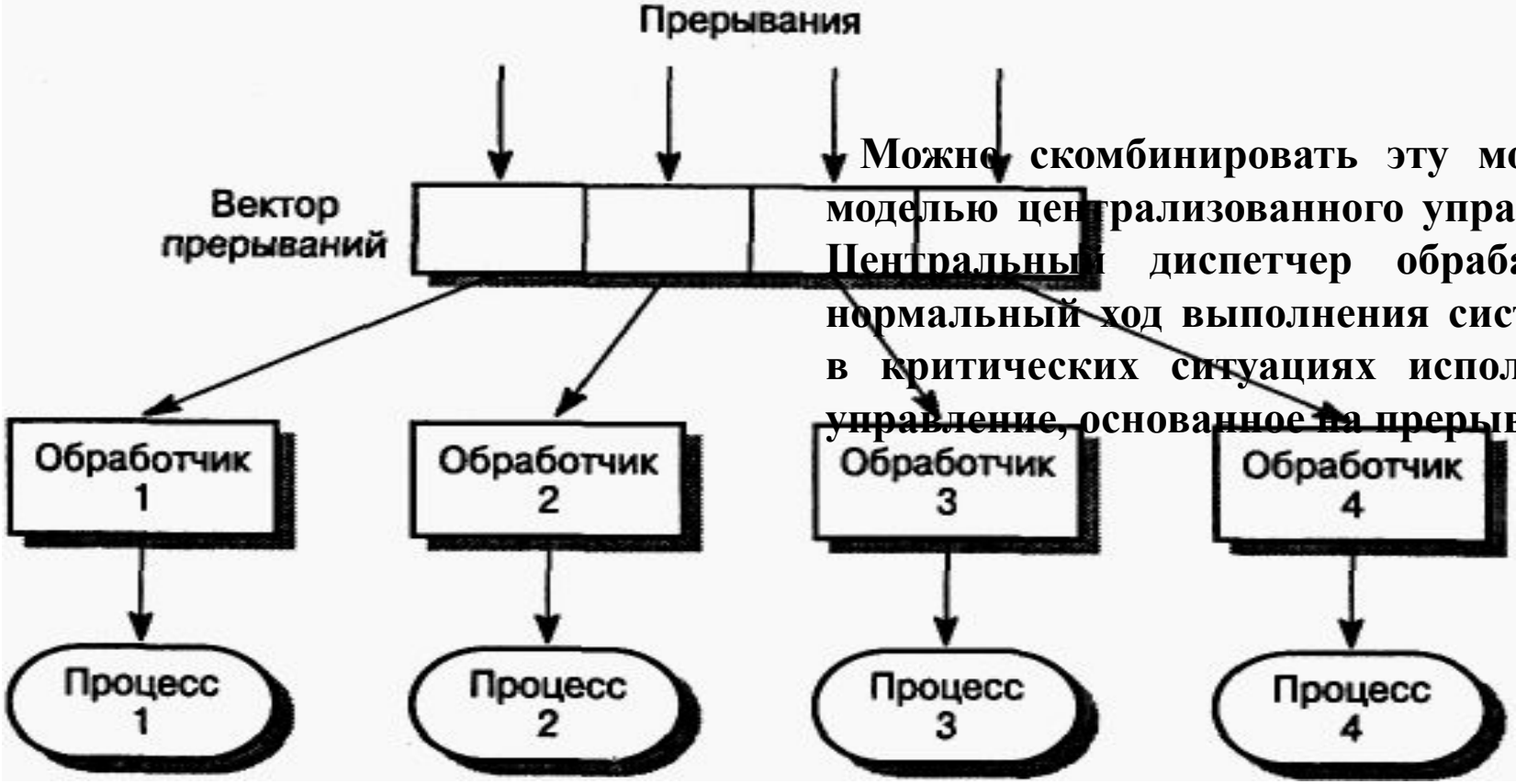
Между моделью передачи сообщений и моделью централизованного управления существует отличие: алгоритм управления не встроен в обработчик сообщений и событий. Подсистемы определяют, какие события им требуются, а обработчик сообщений и событий следит, чтобы данные события были отправлены именно им.

Обработчик события всегда поддерживает двухточечное взаимодействие. Поэтому подсистемы могут явно отправить сообщение другой подсистеме.

Модель управления, основанная на прерываниях

Чтобы обеспечить быструю реакцию на события, необходимо использовать управление, основанное на прерываниях.

Данная модель используется только в жестких СРВ, где требуется немедленная реакция на определенные события.



Можно скомбинировать эту модель с моделью централизованного управления. Центральный диспетчер обрабатывает нормальный ход выполнения системы, а в критических ситуациях используется управление, основанное на прерываниях.

Преимущество - мгновенная реакция системы на происходящие события, недостаток – сложность программирования и аттестации системы.

3 этап - модульная декомпозиция

После этапа разработки системной структуры в процессе проектирования следует этап декомпозиции подсистем на модули.

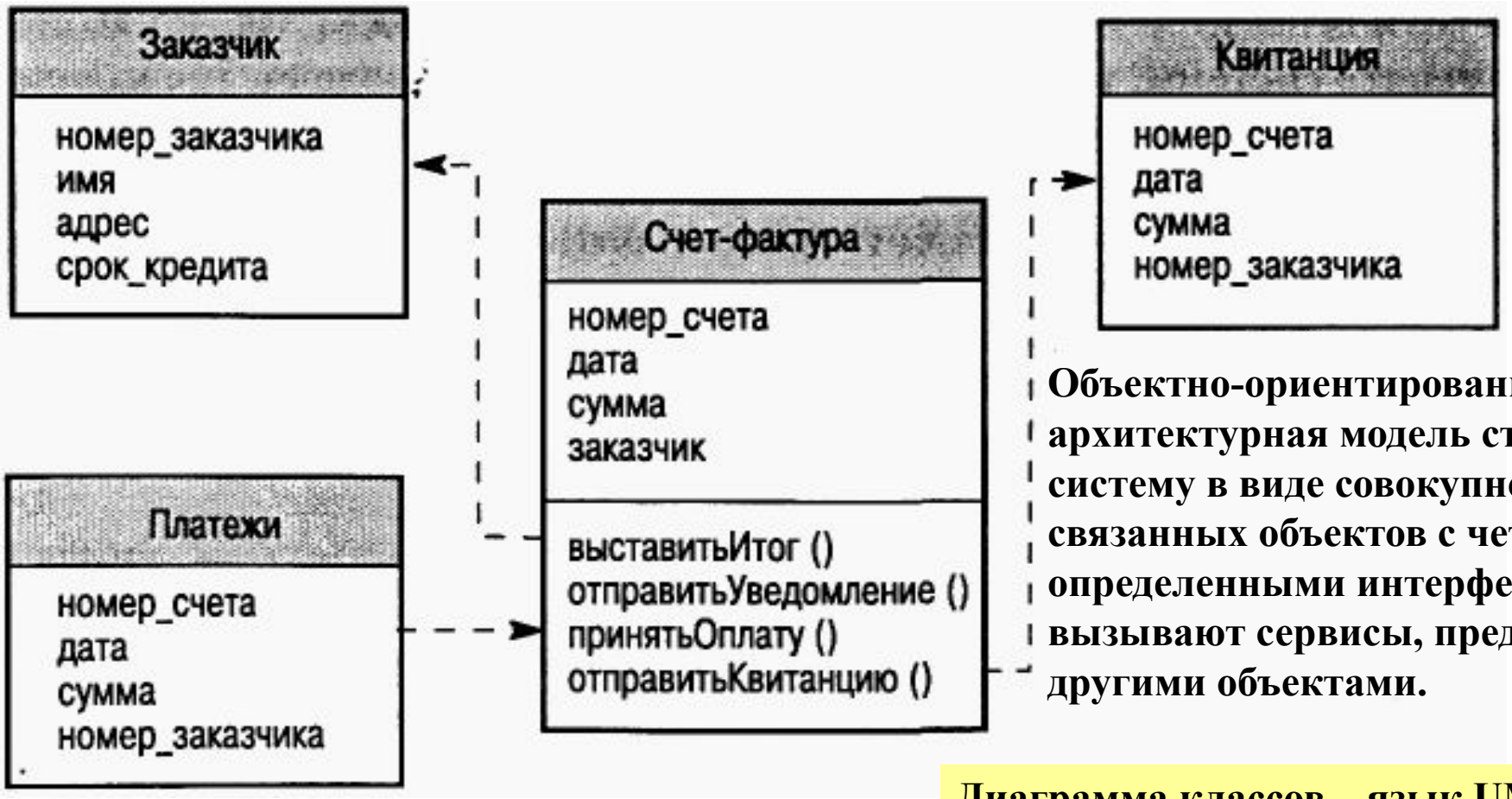
Между разбивкой системы на подсистемы и подсистем на модули нет принципиальных отличий. Однако компоненты модулей обычно меньше компонентов подсистем, поэтому можно использовать специальные модели декомпозиции.

Две модели, используемые на этапе модульной декомпозиции подсистем:

1. ***Объектно-ориентированная модель.*** Система состоит из набора взаимодействующих объектов.
2. ***Модель потоков данных.*** Система состоит из функциональных модулей, которые получают на входе данные и преобразуют их некоторым образом в выходные данные. Такой подход часто называется конвейерным.

Объектная модель системы обработки счетов

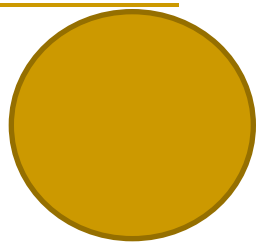
В объектно-ориентированной модели модули представляют собой объекты с собственными состояниями и определенными операциями над этими состояниями.



Объектно-ориентированная архитектурная модель структу систему в виде совокупности с связанных объектов с четко определенными интерфейсами вызывают сервисы, предостав другими объектами.

Диаграмма классов – язык UML

Примерные вопросы по теме



Архитектура ПО. Проблемно-зависимые архитектуры.
Какие типы проблемно-зависимых архитектур бывают
Примеры/состав/особенности/достоинства/
недостатки