Технология разработки программного обеспечения

Лекция 9.

Проектирование программного обеспечения

Разработка ПО

- Выявление требований.
- Анализ системы
- Проектирование программного обеспечения.
- Реализация программного обеспечения.
- Тестирование.
- Внедрение.

Проектирование программного обеспечения

- Основной **целью этапа анализа** было построение логической модели системы, отражающей функциональность, которую должна предоставлять система для удовлетворения требований пользователя.
- При переходе к проектированию ПО нужно решить две задачи:
  - ***определить преобразование модели анализа в модель проектирования***;
  - ***определить взаимосвязь между моделями – использовать их вместе или отдельно***.

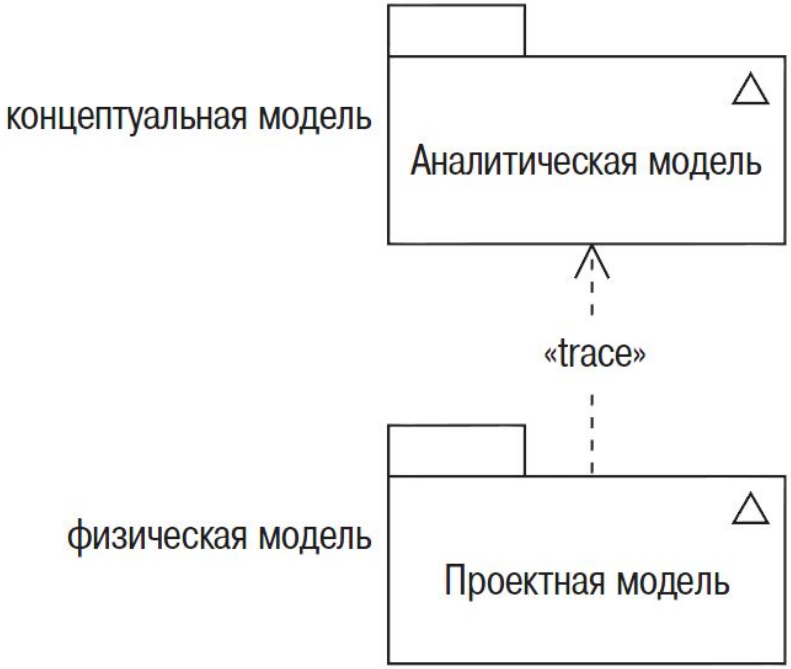
Цель этапа проектирования

- **Цель этапа проектирования** – определить в полном объеме, как будет реализовываться функциональность системы, выявленная на этапе анализа.
  - При анализе целью является поиск способов удовлетворения функциональных требований.
  - При проектировании целью является поиск способов удовлетворения не функциональных требований:
    - Эффективность
    - Безопасность.
    - …

- Одним из путей решения задачи проектирования является одновременное изучение
  - предметной области и
  - области решения ().
- Требования поступают из предметной области, и анализ можно рассматривать как ее исследование с точки зрения заказчиков системы.

- Проектирование предполагает объединение технических решений (библиотек классов, механизмов сохранения объектов и т. д.) для создания модели системы (проектной модели), которая может быть реализована в действительности.
- При проектировании принимаются решения по стратегическим вопросам, таким как сохранение и распределение объектов, в соответствии с которыми и создается проектная модель.

- Проектную модель можно рассматривать как уточнение модели анализа с добавлением деталей и конкретных технических решений.
- Проектная модель содержит все то же самое, что и модель анализа, но все ее элементы проработаны более основательно и должны включать подробности реализации.

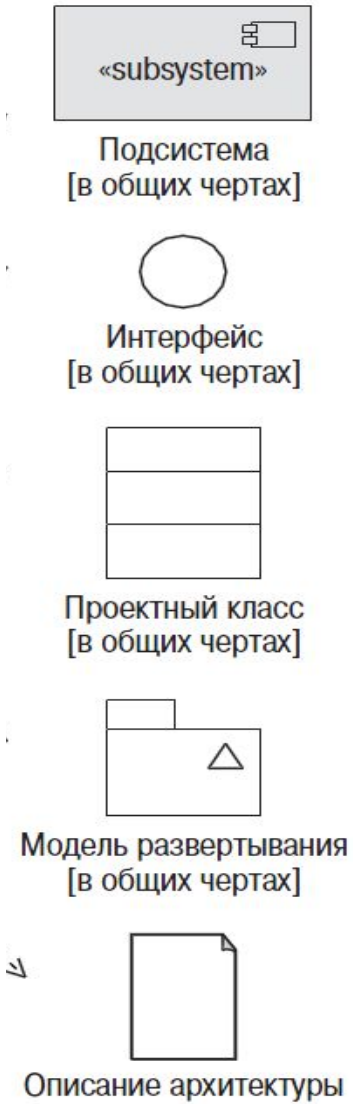


Работы по проектированию ПО

- Проектирование ПС включает:
  1. Проектирования архитектуры (Architectural design)
  2. Проектирование классов (Design a class)
  3. Проектирование вариантов использования (Design a use case)
- Они являются параллельными и итеративными.

Состав модели проектирования

- Результатом этапа проектирования являются следующие модели:
  1. подсистемы этапа проектирования;
  2. классы проектирования;
  3. интерфейсы;
  4. реализации вариантов использования – проектные;
  5. диаграммы развертывания.



Подсистемы

- Проектная модель содержит набор **проектных подсистем**.
- ***Подсистемы*** – это компоненты , которые могут включать различные типы элементов модели.
- Хотя некоторые ключевые интерфейсы уже могли быть определены во время анализа, при проектировании им уделяется намного больше внимания.
- Именно интерфейсы проектных подсистем объединяют систему в единое целое.
- Их роль при проектировании архитектуры очень большая, поэтому поиску и моделированию ключевых интерфейсов посвящается очень много

Интерфейсы

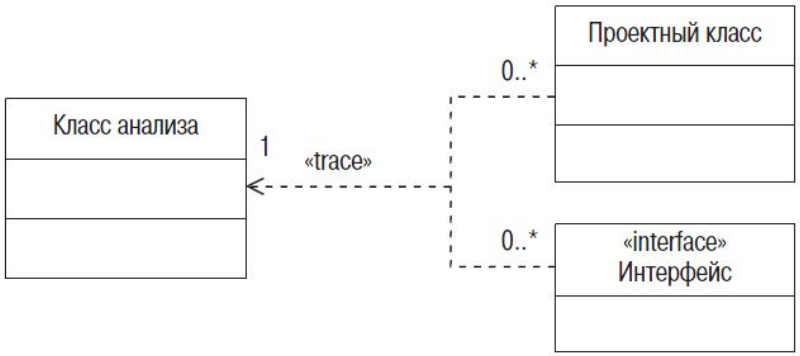
- Одним из ключевых результатов проектирования являются интерфейсы.
- Они позволяют разложить систему на подсистемы, которые могут разрабатываться параллельно.
- При проектировании также создается диаграмма развертывания в первом приближении, которая показывает распределение программной системы на физических вычислительных узлах.
- Такая диаграмма является очень важной и имеет стратегическое значение.

Классы проектирования

- Класс анализа это только эскиз класса, который будет программироваться
  - Например, он может содержать пару атрибутов и только несколько ключевых операций.
- Проектный класс непосредственно соответствует программируемому классу.
- Он должен быть точно определен – содержать полное описание:
  - всех атрибутов с указанием типов и начальных значений;
  - всех операции (включая возвращаемые типы и списки параметров).

Классы проектирования (2)

- Класс анализа может быть превращен в один или более интерфейсов или проектных классов.
  - Классы анализа являются высокоуровневым концептуальным представлением классов системы.
  - При проектировании, для реализации этих концептуальных классов может понадобиться один ил ых классов и/или и

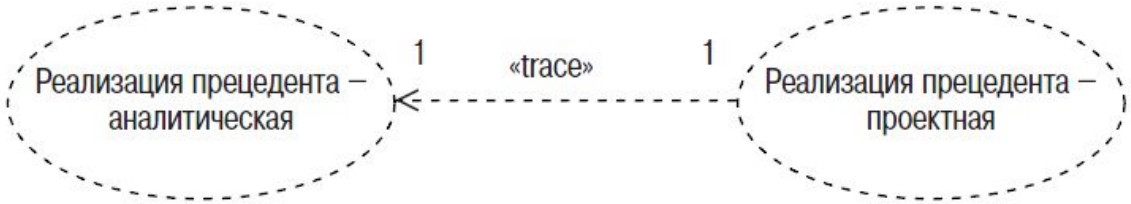


Реализация варианта использования на этапе проектирования

- «Реализация варианта использования на этапе проектирования» – это завершающий этап жизненного цикла реализации варианта использования.
- Он включает проектные классы как часть своей структуры и разрабатывается параллельно с ними.

Отношение между реализациями вариантов использования

- Между аналитическими и проектными реализациями вариантов использования устанавливается простое отношение «trace» «один-к-одному».
- При проектировании реализация варианта реализации просто становится более детализированной.



Связь между моделями анализа и проектирования

- В идеальном случае для программной системы можно было бы создавалась единственную модель, а средство моделирования позволяло бы работать и с моделью анализа и с моделью проектирования.
- Однако ни одна из систем UML-моделирования не позволяет создавать модели анализа и модели проектирования на основании одной базовой модели.
- Модели анализа сохраняются для больших, сложных или стратегически важных систем.

Стратегии работы с моделями анализа и проектирования

№	Стратегия	Результат
1	Модель анализа дополняется до модели проектирования	Имеется единственная проектная модель.
2	Модель анализа, дополняется до модели проектирования, а с помощью инструмента моделирования восстанавливается «представление анализа».	Имеется единственная проектная модель, но восстановленное инструментом моделирования модели анализа может быть неприемлемым.
3	Модель анализа замораживается. До проектной модели дополняется копия модели анализа.	Имеются две модели, но они не синхронизированы.
4	Поддерживаются две отдельные модели – анализа и проектная	Имеются две модели – они синхронизованы, но их обслуживание очень трудоемко

Нужно ли сохранять модель анализа?

- Модель анализа обеспечивают «общую картину» системы.
- В них может быть лишь от 1 до 10% классов **подробной проектной модели**, поэтому они более понятны.
- Их роль является важной для следующих действий:
  1. введения в проект новых людей;
  2. ***понимания системы спустя месяцы или годы после ее поставки;***
  3. понимания того, как система выполняет требования пользователей;
  4. ***планирования обслуживания и улучшения;***
  5. понимания логической архитектуры системы;
  6. привлечения внешних ресурсов к разработке системы.
- Если необходимо что-то из перечисленного выше,

- Обычно модели анализа должны сохраняться для любой большой, сложной, стратегически важной системы или системы с предположительно долгим сроком эксплуатации.
  - Это означает, что необходимо выбирать между стратегиями 3 и 4.
- Возможность рассинхронизации моделей анализа и проектирования моделей должна быть всегда очень тщательно продумана.
  - Если система небольшая (меньше 200 проектных классов), тогда непосредственно модель проектирования достаточно мала и понятна, поэтому необходимости в отдельной модели анализа, возможно, и нет.
  - Если система не имеет стратегического значения или недолговечна, то одновременная поддержка моделей анализа и проектирования – излишнее требование.

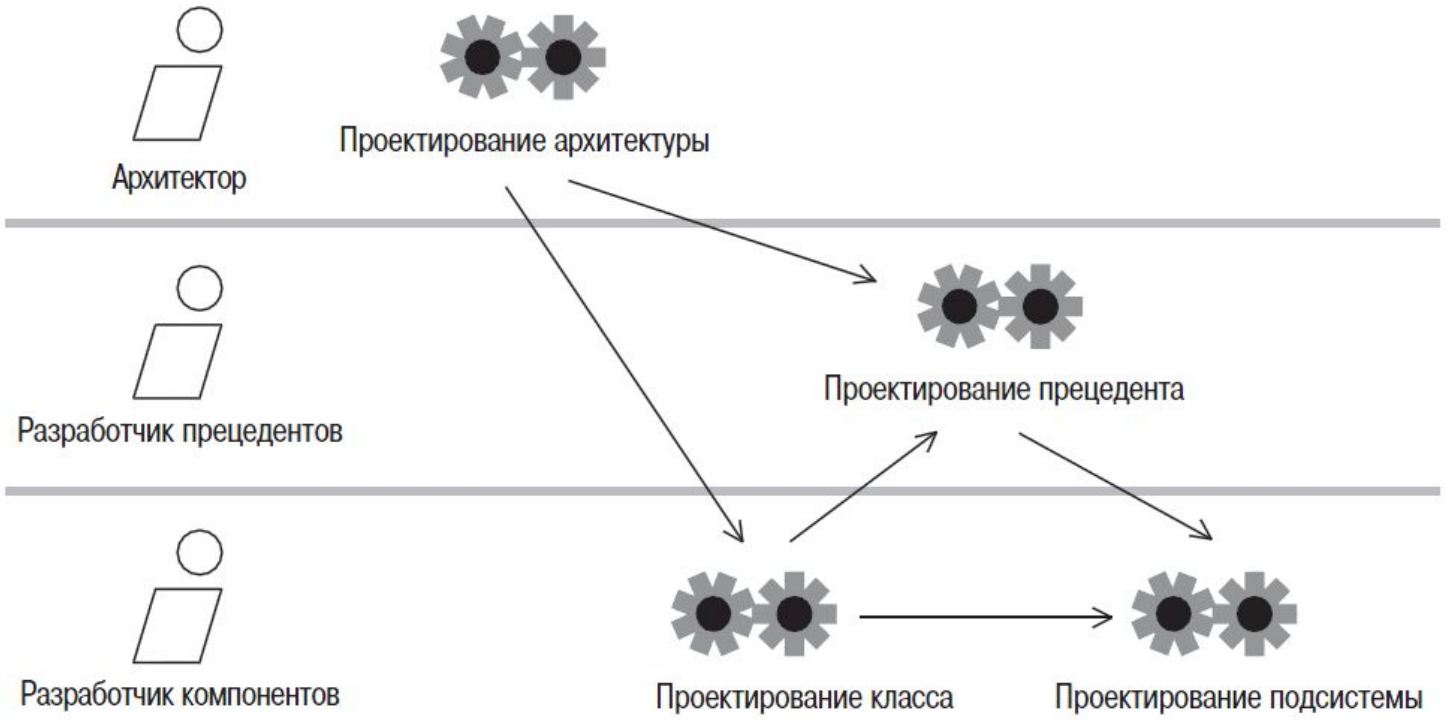
- В этом случае необходимо выбирать между стратегиями 1 и 2, и решающим фактором будут возможности используемого инструментального средства UML-моделирования.
- Некоторые инструменты моделирования сохраняют одну базовую модель и обеспечивают возможность применения фильтров и сокрытия информации для восстановления «аналитического» представления из проектной модели.
  - Это разумный компромисс для многих систем среднего размера, но для очень больших систем этого, скорее всего, недостаточно.
- **Следует помнить о том, что реальный срок службы многих систем существенно превышает предполагаемый!**

Участники этапа проектирования

- Основными участниками этапа проектирования являются:
  - архитектор,
  - разработчик вариантов использования,
  - разработчик компонентов.
- В большинстве ОО проектов роль архитектора выполняют один или несколько специалистов, но часто они же потом являются разработчиками прецедентов и компонентов.
- Один из подходов к проектированию состоит в том, чтобы определенные члены команды отвечали за отдельные части системы, начиная с этапа анализа и заканчивая этапом реализацией.

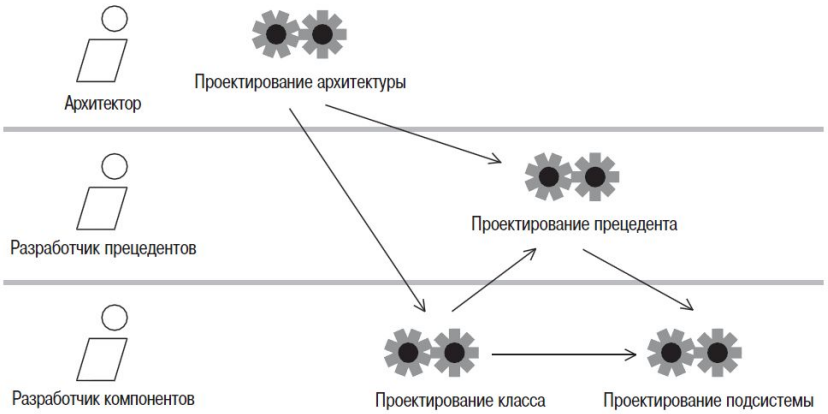
- Таким образом, специалист или команда, ответственные за создание конкретной части ОО модели анализа, будут дополнять ее до проектной модели и, возможно с помощью программистов, превращать ее в код.
- При таком подходе (в этом его основное преимущество) нет «перекладывания ответственности» друг на друга между аналитиками, проектировщиками и программистами, что распространено в ОО проектах.

Распределение работ по участникам этапа проектирования



Работы этапа проектирования

- Этапа проектирования включает следующие работы:
  1. Проектирование архитектуры.
  2. Проектирование классов.
  3. Проектирование вариантов использования.
  4. Проектирование подсистем.

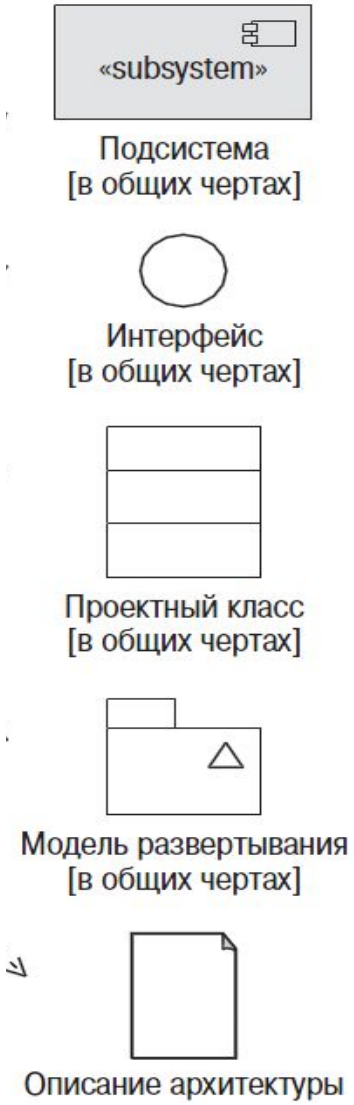


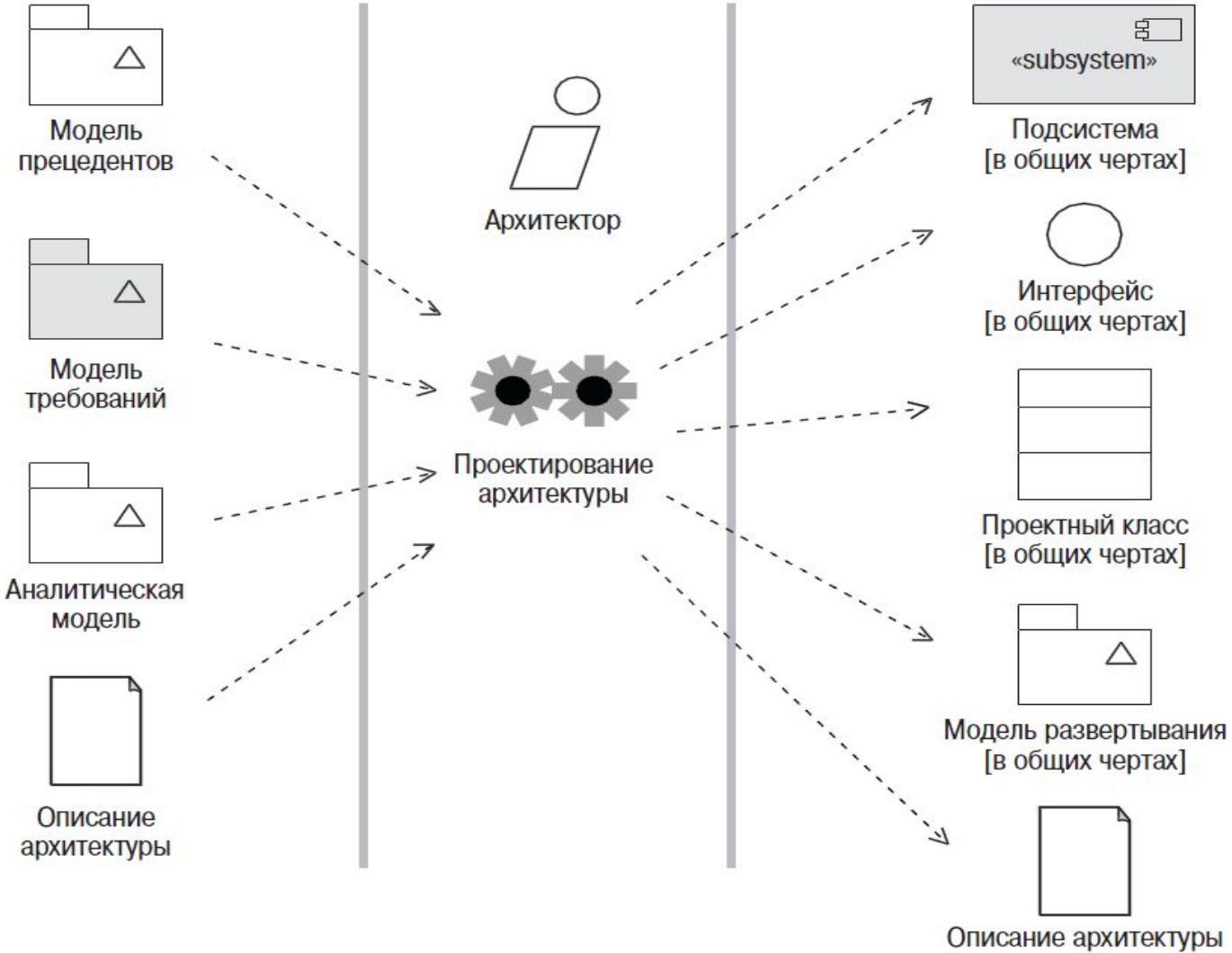
1. Проектирование архитектуры

- Процесс проектирования начинается с работы под названием «Проектирование архитектуры» (architectural design).
- Ее осуществляют один или более архитекторов.

Результаты процесса проектирования архитектуры

- В результате «Проектирования архитектуры» создается набор описаний:
  - подсистемы (в общих чертах);
  - интерфейсы (в общих чертах);
  - проектные классы;
  - модель развертывания;
  - описание архитектуры.




Модель прецедентов
Модель требований
Аналитическая модель
Описание архитектуры
Архитектор
Проектирование архитектуры
«subsystem»
Подсистема [в общих чертах]
Интерфейс [в общих чертах]
Проектный класс [в общих чертах]
Модель развертывания [в общих чертах]
Описание архитектуры

Проектирование архитектуры

- **Проектирование архитектуры** заключается в предварительном описании моделей, важных с архитектурной точки зрения, с целью создания общего плана архитектуры системы.
- На языке UML подсистемы описываются с помощью пакетов.
- Обозначенные в общих чертах описания являются входными данными для более детального проектирования, в процессе которого происходит их конкретизация

Начальное формирование подсистем

- Начальное формирование подсистем осуществляется путем идентификации групп элементов модели, имеющих прочные семантические связи.
- Кандидаты в подсистемы часто обнаруживаются со временем в ходе развития и совершенствования модели.
- Кандидаты в подсистемы обязательно должны отражать реальные семантические группы элементов, а не просто некоторое идеализированное (но выдуманное) представление логической архитектуры.

Выявление на основе статической модели

- Статическая модель – самый полезный источник кандидатов в подсистемы.
- Необходимо искать следующее:
  - связные группы классов на диаграммах классов;
  - иерархии наследования.

Выявление на основе вариантов использования

- В качестве источника кандидаты в подсистемы можно также рассматривать модель вариантов использования.
- Важно попытаться сделать кандидаты в подсистемы максимально связанными с точки зрения бизнес-процесса.
- Обычно варианты использования пересекают несколько кандидатов: один вариант использования может реализовываться классами из нескольких разных пакетов.
- Тем не менее один или более вариантов использования, поддерживающих определенный бизнес-процесс, или актера, или ряд взаимосвязанных вариантов использования, могут указывать на потенциального кандидата в подсистему.

Сокращение открытых членов кандидатов в подсистемы

- После идентификации ряда предполагаемых пакетов необходимо попытаться сократить до минимума количество открытых членов и зависимостей между пакетами.
- Это осуществляется путем:
  - перемещения классов из пакета в пакет,
  - добавления пакетов,
  - удаления пакетов.

Принципы формирования пакетов

- Основой хорошей структуры пакетов является
  - высокая связность *в рамках* пакета и
  - низкая связанность *между* пакетами.
- Пакет должен содержать группу тесно взаимосвязанных классов.
- Самую тесную взаимосвязь классов образует
  - наследование;
  - далее композиция;
  - агрегирование ;
  - ассоциации

- Классы, образующие иерархии наследования или композиции, являются основными кандидатами на размещение в одном пакете.
- Это обеспечит высокую связность в рамках пакета и, вероятно, более низкую связанность с другими пакетами.

Принцип простоты

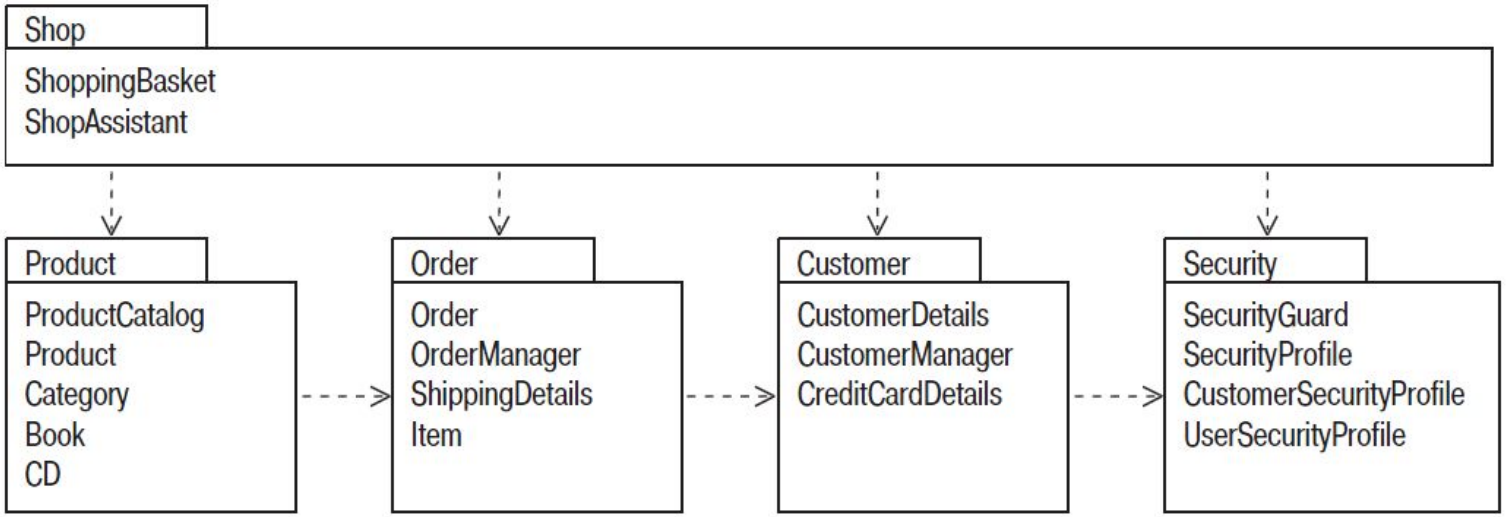
- И как всегда, при создании модели пакетов на этапе анализа необходимо придерживаться простоты.
- Важнее получить правильный набор пакетов, чем широко применять такие возможности, как обобщение пакетов и стереотипы зависимостей.
- Все это можно добавить позже и только в том случае, если это сделает модель более понятной.

- Отсутствие вложенных пакетов – один из гарантов простоты модели.
  - Чем глубже что-то помещено в структуру вложенных пакетов, тем более непонятным оно становится.
  - Иногда встречаются модели с очень глубоко вложенными пакетами, содержащими лишь один-два класса.
  - Такие модели больше похожи на стандартную функциональную декомпозицию, чем на объектную модель.

- В пакете должно быть от четырех до десяти классов анализа.
- Но могут быть и исключения из правил.
- Если нарушение этого правила делает модель более ясной, нарушайте его!
- Иногда в модель пакетов необходимо ввести пакеты с одним-двумя классами, чтобы избавиться от циклической зависимости.
- В таких случаях это вполне обоснованно.

Пример модели пакетов

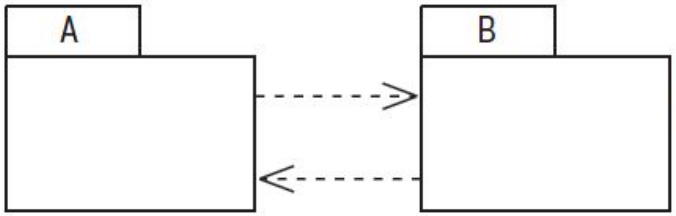
- Пример модели пакетов на этапе анализа для простой системы электронной коммерции.



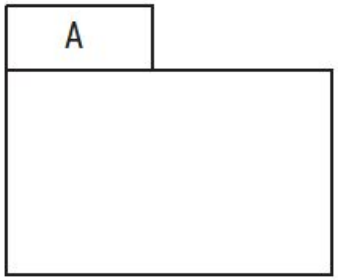
Циклические зависимости пакетов

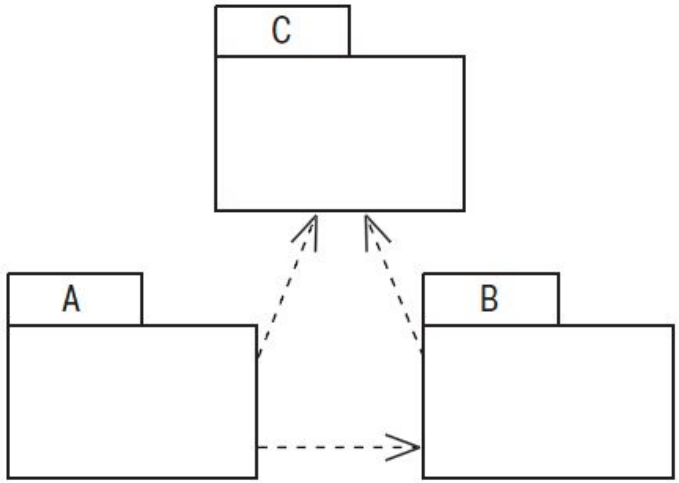
- В модели пакетов на этапе анализа необходимо избегать циклических зависимостей.
- Если пакет A некоторым образом зависит от пакета B, и наоборот, то это очень сильный аргумент в пользу объединения этих двух пакетов.
- Объединение пакетов (merging) является хорошим способом устранения циклических зависимостей.
- Но лучше, и это очень часто срабатывает, попытаться вынести общие элементы в третий пакет C и удалить цикл, перестроив отношения зависимостей.
- **Следует избегать циклических зависимостей между пакетами.**

Пример устранения циклической зависимости пакетов



Возможны более сложные циклы, в которых принимают участие три или более пакетов!





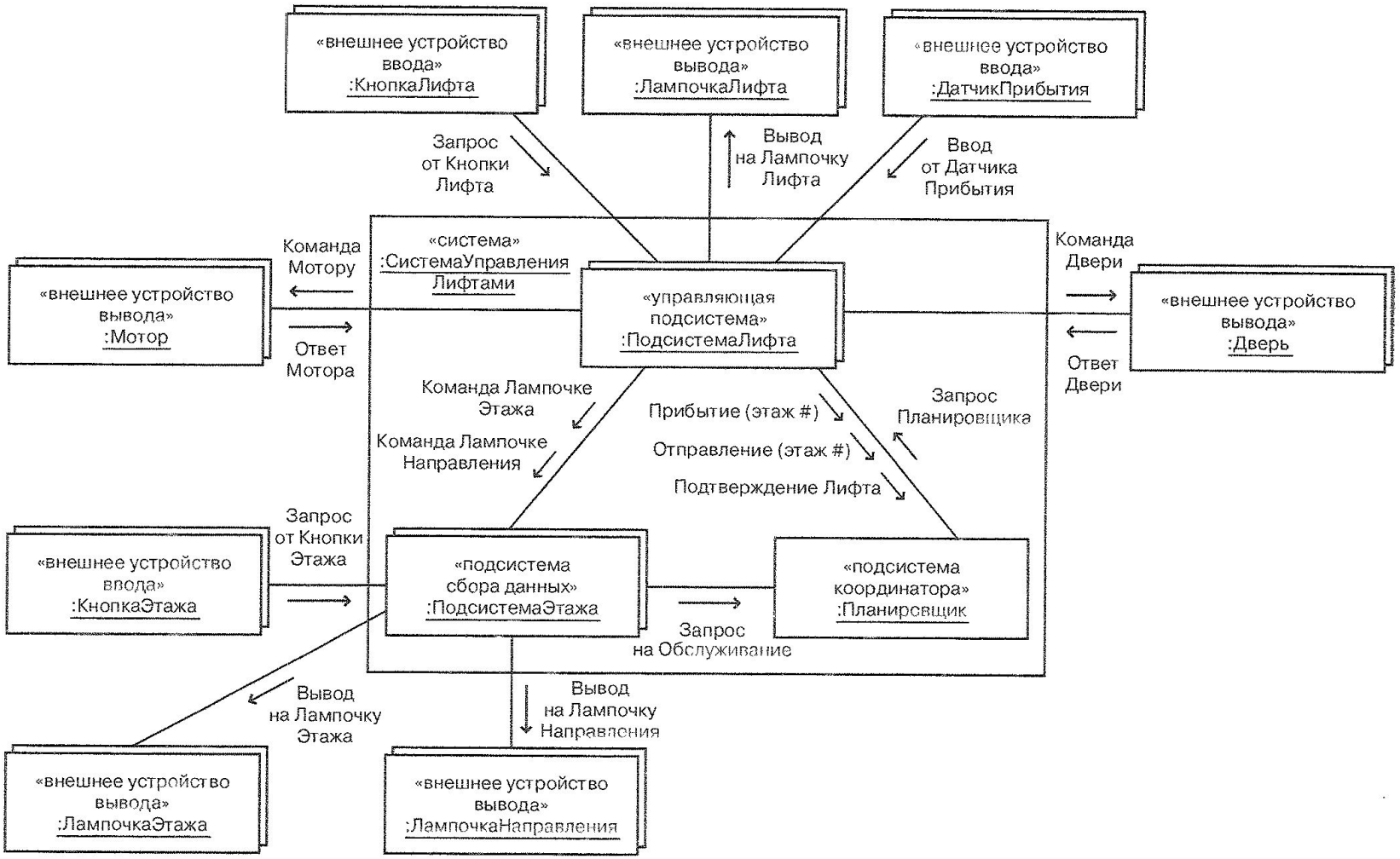
- В модели анализа порой невозможно создать диаграмму пакетов без нарушения прав доступа, поскольку в анализе часто используются двунаправленные отношения между классами.
- Предположим, имеется очень простая модель: один класс находится в пакете А, другой класс – в пакете В.
- Если класс пакета А имеет двунаправленное отношение с классом пакета В, тогда пакет А зависит от пакета В, но и пакет В зависит от пакета А.
- Имеем циклическую зависимость между двумя пакетами.

- Единственная возможность избавиться от этого нарушения – уточнить отношение между A и B, сделав его однонаправленным, либо поместить оба класса в один пакет.
- Таким образом, зависимости пакетов являются превосходным аргументом в пользу применения возможности навигации в аналитических моделях!
- С другой стороны, классы, действительно имеющие взаимные зависимости (а не зависимости, являющиеся результатом несовершенства модели), должны находиться в одном пакете.

Декомпозиция системы

- Система разбивается на подсистемы, состоящие из объектов, которые функционально зависят друг от друга.
- Принципы разделения
  - Подсистема должна минимально зависеть от других подсистем (минимальная связанность, coupling).
  - С другой стороны, степень связанности между объектами внутри одной подсистемы обязана быть высокой (максимальное сцепление, cohesion).
- Подсистему удобно рассматривать как составной объект или агрегат, содержащий более простые объекты.
- Подсистем одного типа может быть несколько.

Пример распределенной архитектуры: система управления лифтами



Декомпозиция системы (2)

- Во время анализа предметной области и декомпозиции системы акцент ставится на разделении обязанностей.
- Каждая подсистема выполняет некоторую основную функцию, которая мало зависит от функциональности прочих подсистем.
- Подсистему допустимо разложить на более мелкие подсистемы, обеспечивающие выполнение части ее функций.
- После того как будет определен интерфейс между подсистемами, их дальнейшее проектирование можно вести независимо.

- Подсистема обеспечивает сокрытие информации на более высоком уровне, чем объект.
- Разбиение системы на подсистемы следует начинать с изучения вариантов использования.
- Основой подсистемы служит модель взаимодействия объектов в ВИ, поскольку они связаны друг с другом.
- Степень такой связанности достаточно высока, в отличие от степени связанности с объектами из других вариантов использования (она мала или отсутствует).
- Объект, принимающий участие сразу в нескольких вариантах использования, следует отнести к какой-то одной подсистеме - обычно к той, с которой он теснее связан.
- Иногда подсистема включает объекты из разных ВИ.
- Чаще всего так бывает, когда в нескольких ВИ имеются общие, функционально связанные между собой объекты.

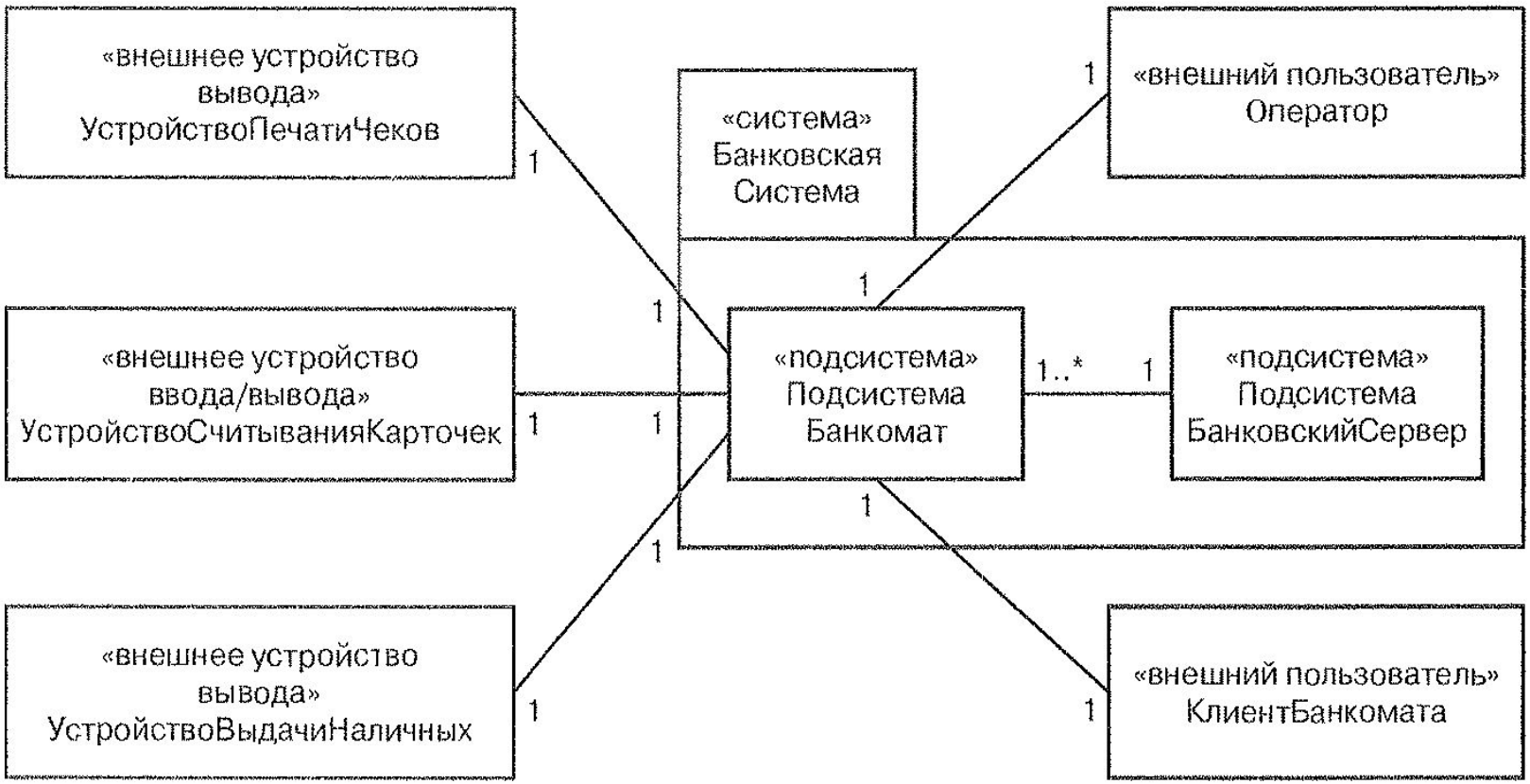
Рекомендации по выявлению подсистем

- В некоторых приложениях, например в системах клиент-сервер, подсистемы легко идентифицируются.
- Так, в банковской системе есть подсистема «Банкомат», размещенная в каждом банкомате, и центральная подсистема «Банковский Сервер».
- Это пример разбиения на подсистемы по территориальному признаку, когда географическая удаленность заложена в самом описании задачи.
- Территориальное распределение - очень веская причина для выделения подсистем.
- В других случаях не так очевидно, что следует считать подсистемой

Рекомендации по выявлению подсистем (2)

- Поскольку цель разбиения на подсистемы - объединить тесно связанные по функциональному признаку объекты, то начать лучше всего с вариантов использования.
- Объекты, участвующие в одном и том же ВИ (абстрактном или конкретном), - это кандидаты на включение в общую подсистему, если только они не являются территориально удаленными.
- Поэтому разбиение на подсистемы обычно производят после выявления объектов, принимающих участие в варианте использования, и изображения их на диаграммах взаимодействий.
- Напомним, что, хотя объект может участвовать в нескольких вариантах использования, он должен входить только в одну подсистему.
- Обычно объект относят к той подсистеме, с которой он связан наиболее тесно.

Пример программной архитектуры клиент-сервер: «банковская система»



Объединение диаграмм коммуникации

- Для перехода от анализа к проектированию и выявлению подсистем, надо сформировать начальный проект программы на основе полученных результатов - путем интегрирования различных частей модели анализа.
- При разработке модели анализа были составлены диаграммы взаимодействия для каждого варианта использования.
- Интегрированная диаграмма кооперации получается при объединении отдельных диаграмм для разных ВИ.
- Обычно порядковые номера сообщений на ней не указывают, чтобы не загромождать

- Интеграция, выполняемая на этом этапе, аналогична анализу стабильности (robustness analysis), применяемому в других методиках.
- Для выполнения такого анализа используется динамическая модель, так как она отражает коммуникационные интерфейсы, играющие огромную роль в распределенных приложениях и приложениях реального времени.
  - Часто варианты использования исполняются в определенном порядке.
  - Очередность объединения диаграмм кооперации должна соответствовать последовательности вариантов использования.

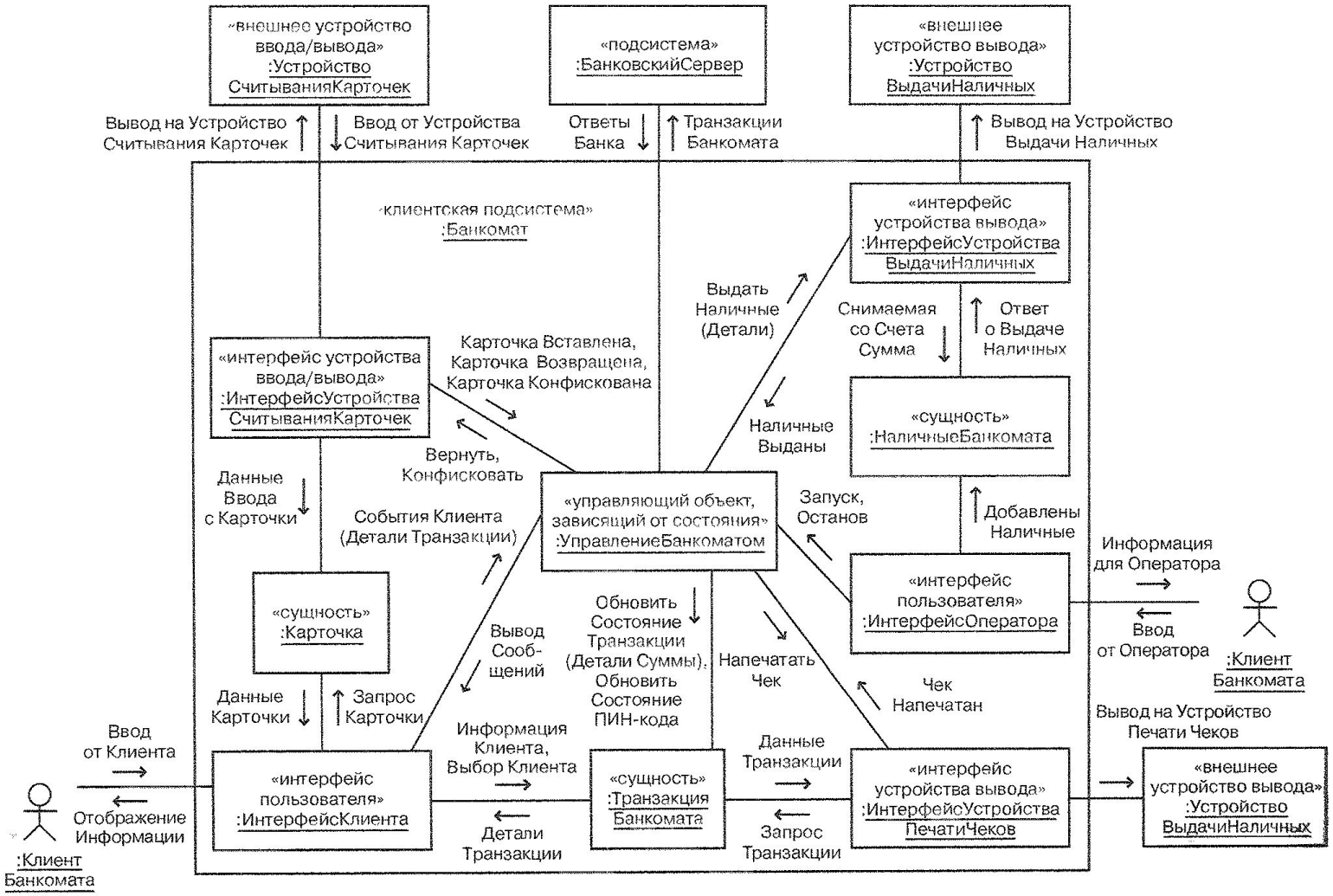
- Визуально интеграция (объединение) вариантов использования происходит следующим образом:
  - берется диаграмма кооперации для первого ВИ, и
  - на нее накладывается диаграмма кооперации для второго ВИ.
  - затем поверх них располагают третью диаграмму и т.д.
- В каждом случае в интегрированную диаграмму включаются новые объекты и сообщения, поэтому она постоянно усложняется.
- Некоторый объект или сообщение может присутствовать в нескольких диаграммах, но изображается он только один раз.

- Важно понимать, что на интегрированной диаграмме кооперации должны быть представлены все сообщения.
- Часто на диаграммах кооперации показывают только основную последовательность, а альтернативные опускают.
- На интегрированной диаграмме нужно отразить и сообщения, посылаемые при исполнении альтернативных последовательностей.
  - Ранее был рассмотрен пример диаграмм кооперации для банковской системы, поддерживающих несколько альтернативных последовательностей.
  - Эти дополнительные сообщения необходимо включить в интегрированную диаграмму, которая призвана дать полную картину всех коммуникаций.
  - Естественно, в таком случае информации оказывается гораздо больше.

- Один из способов уменьшить объем сведений состоит в агрегировании сообщений:
  - если один объект посылает другому несколько отдельных сообщений, то они показываются не по отдельности, а в составе одного агрегированного сообщения.
  - Например, все сообщения, приходящие объекту «Круиз Контроль», можно объединить в одно – «Сообщения Круиз-Контролю».
- Затем для определения состава полученного сообщения используется словарь сообщений.

- Таким образом, интегрированная диаграмма кооперации представляет собой результат объединения нескольких диаграмм кооперации, где изображены все объекты и их взаимодействия.
  - При этом номера сообщений не показываются.
- Пример интегрированной диаграммы кооперации для подсистемы «Банкомат» приведен на следующем слайде.

Пример консолидированной диаграммы кооперации: подсистема Банкомат



- Если система большая, то интегрированная диаграмма сильно разрастается, так что изображать все объекты на ней становится неудобно.
  - для упрощения, можно показывать объекты, а сообщения, которыми они обмениваются, помечать агрегированными именами (об этом уже говорилось выше).
- На предыдущем слайде примерами имен агрегированных сообщений являются
  - «транзакции Банкомата»,
  - «Ответы Банка» и
  - «События Клиента».

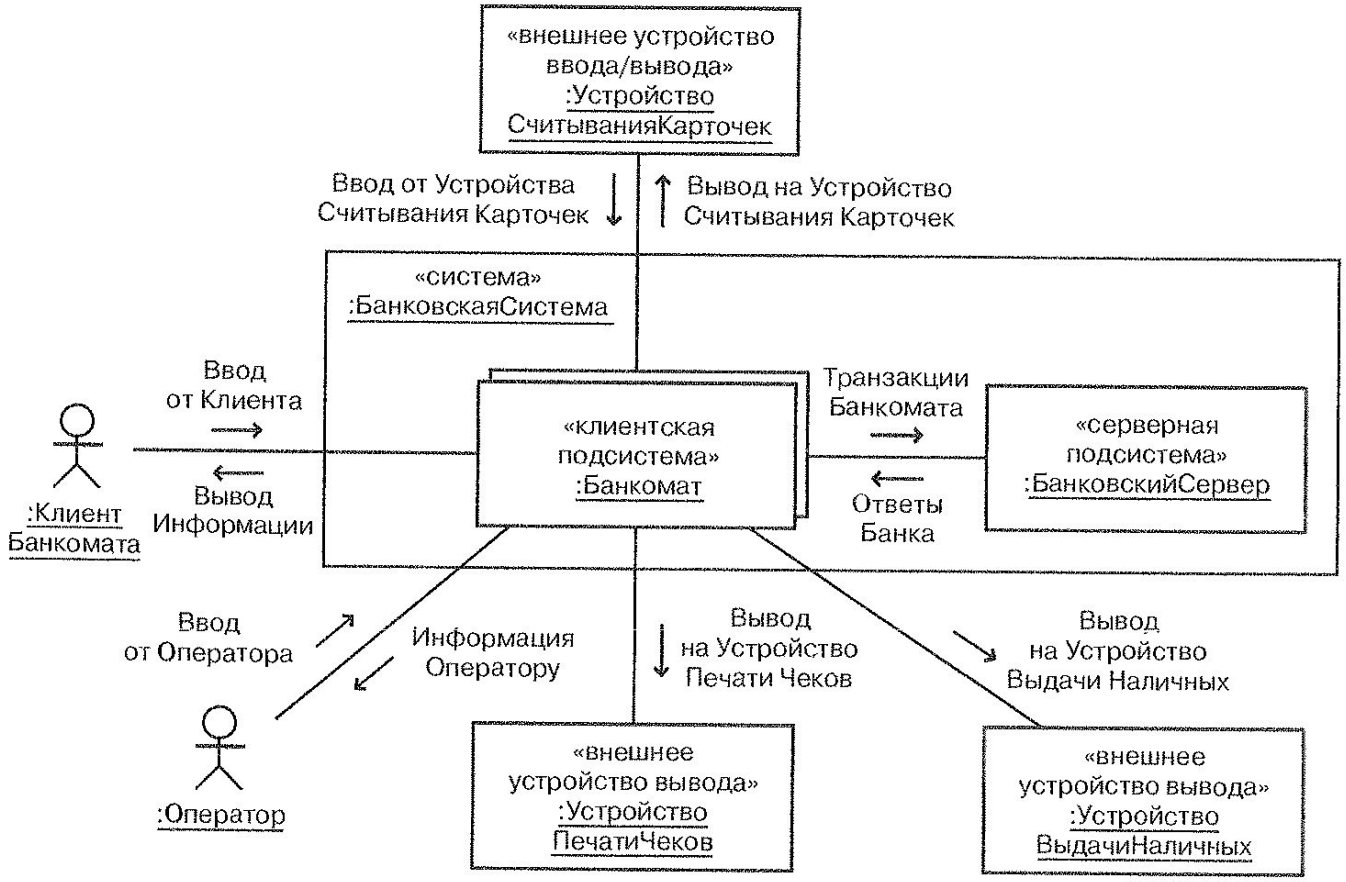
- Другой способ упрощения интегрированной ДК заключается в следующем:
  - создание интегрированной диаграммы кооперации для каждой подсистемы
  - создание высокоуровневую диаграмму кооперации подсистем, на которой будут показаны взаимодействия между подсистемами.

Изображение подсистем

- Подсистемы можно изображать на диаграммах кооперации или на диаграммах классов.
- Диаграмма классов подсистемы передает структурные отношения между подсистемами.
- На следующем сладе приведен пример диаграммы классов, описывающей отношения между «Банковским Сервером» и «Банкоматами» в банковской системе.
- Динамические взаимодействия между подсистемами можно изображать на диаграмме кооперации подсистем, которая представляет собой высокоуровневую консолидированную диаграмму кооперации.
- Структуру отдельной подсистемы лучше показать на ее собственной интегрированной диаграмме кооперации, отражающей все входящие в подсистему объекты и их взаимодействия.

Пример проектирования подсистем банковской системы

- Высокоуровневая диаграмма коммуникации банковской системы.



Рекомендации по выделению подсистем

- Для обеспечения высокой связанности внутри подсистем и низкой связанности между подсистемами, необходимо соблюдать следующие рекомендации:

1. композитные объекты должны включаться в одну подсистему;
2. агрегированная система вполне может представлять собой высокоуровневую подсистему;
3. территориально удаленные объекта должны принадлежать различным подсистемам;
4. клиенты и серверы должны принадлежать разным подсистемам;
5. объекты интерфейса пользователя находятся в отдельной подсистеме;
6. управляющий объект и управляемые им объекты,

1. Композиционные объекты должны включаться в одну подсистему

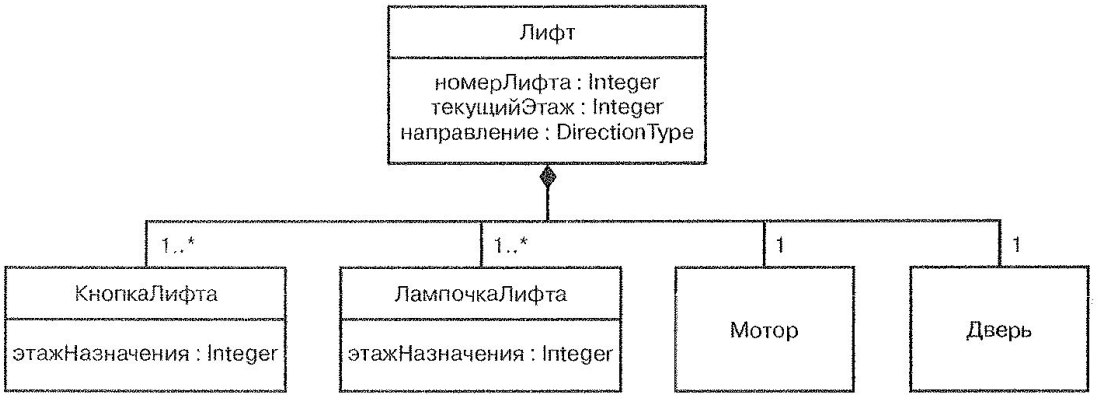
- Объекты, являющиеся частями одного и того же композиционного объекта, следует помещать в одну подсистему и должны отделяться от объектов, которые не являются частями того же самого композиционного объекта.
- Как раннее отмечалось, агрегирование, и композиция являются отношениями типа «часть-целое»,
  - композиция - это более сильная этого отношения.
  - в композиции составной объект (целое) и входящие в него объекты (части) создаются, живут и умирают вместе.
- Подсистема, которая содержит композицитный объект и все его части, сильнее внутренне связана (сцеплена), чем подсистема, содержащая агрегированный объект и составляющие его части.

- Подсистема обеспечивает сокрытие информации на более высоком уровне абстракции, нежели отдельный объект.
  - Программный объект можно использовать для моделирования реального объекта из предметной области.
- Композитный программный объект моделирует составной объект реального мира, который содержится в рассматриваемой предметной области.
- Композитный объект, как правило, включает в себя группу объектов, которые работают совместно и скоординированно.
  - Этим он напоминает сборную деталь в производстве.

- Часто приложению требуется несколько экземпляров композитного объекта (а значит, и каждого из составляющих его объектов).
- Отношение между композитным классом и его частями лучше всего показывать на статической модели, поскольку диаграмма классов позволяет показать кратность ассоциаций между целым и частями.

Пример композиционного класса

- Примером композиционного класса служит класс Лифт . Каждый «Лифт» содержит следующие объекты
  - «Кнопка Лифта»,
  - «Лампочка Лифта»,
  - «Мотор» и
  - «Дверь».
- Существует несколько экземпляров составного класса «Лифт» - по одному на каждый лифт.

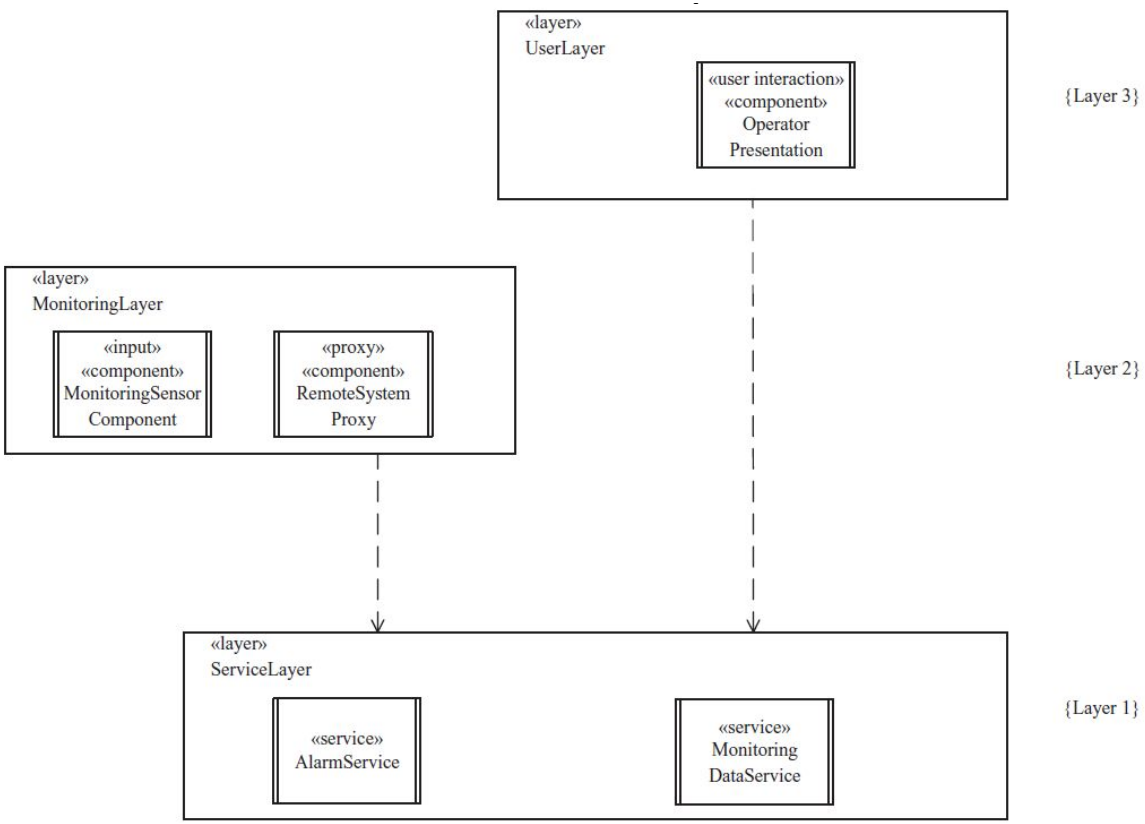


2. Агрегированная система как высокоуровневой подсистемой

- Агрегированная система вполне может быть высокоуровневой подсистемой, которая содержит композитные подсистемы.
- Агрегированная подсистема содержит объекты, сгруппированные по схожести выполняемых ими функций, которые могут пересекать территориальные границы.
- Эти агрегированные объекты группируются совместно так, как они функционально похожи и взаимодействуют друг с другом в одном и том же варианте использования (или нескольких ВИ).
- Агрегированные подсистемы могут использоваться в качестве более удобной высокоуровневой абстракции чем композиционные подсистемы, в особенности в тех случаях, когда имеется много компонент в очень распределенном приложении.
- В архитектуре ПО, которая связывает много организаций, может быть полезным показать каждую организацию в виде агрегированной подсистемы.

- Многоуровневая архитектура также м.б. структурирована таким образом, что каждый уровень проектируется в виде агрегированной подсистемы.
- Каждый уровень сам может состоять из набора территориально распределенных композиционных подсистем (спроектированных в виде компонент или сервисов).

- Например, архитектура ПО системы аварийной сигнализации (Emergency Monitoring System) имеет уровни, которые спроектированы в виде агрегированных подсистем.
- Каждый уровень содержит одну или несколько компо сервисов).



3. Объекта расположенные в разных местах должны принадлежать различным подсистемам

- Если два объекта физически находятся в разных местах, они должны принадлежать различным подсистемам.
- В распределенной среде общение между разнесенными подсистемами может происходить только путем обмена сообщениями.
- В системе управления лифтами, показанной на рис, 12.4, каждый экземпляр подсистемы Лифт реализуется в отдельном микропроцессоре, физически расположенном в конкретном лифте.
- Каждый экземпляр подсистемы Этаж также находится в отдельном микропроцессоре,

4. Клиенты и серверы должны принадлежать разным подсистемам

- Данную рекомендацию можно считать частным случаем разбиения по территориальному признаку, поскольку клиенты и серверы обычно находятся в разных местах.
- Например, в банковской системе есть много однотипных подсистем «Банкомат», размещенных в конкретных физических банкоматах на территории страны.
- Подсистема же «Банковский Сервер» находится в одном месте, возможно в Москве.

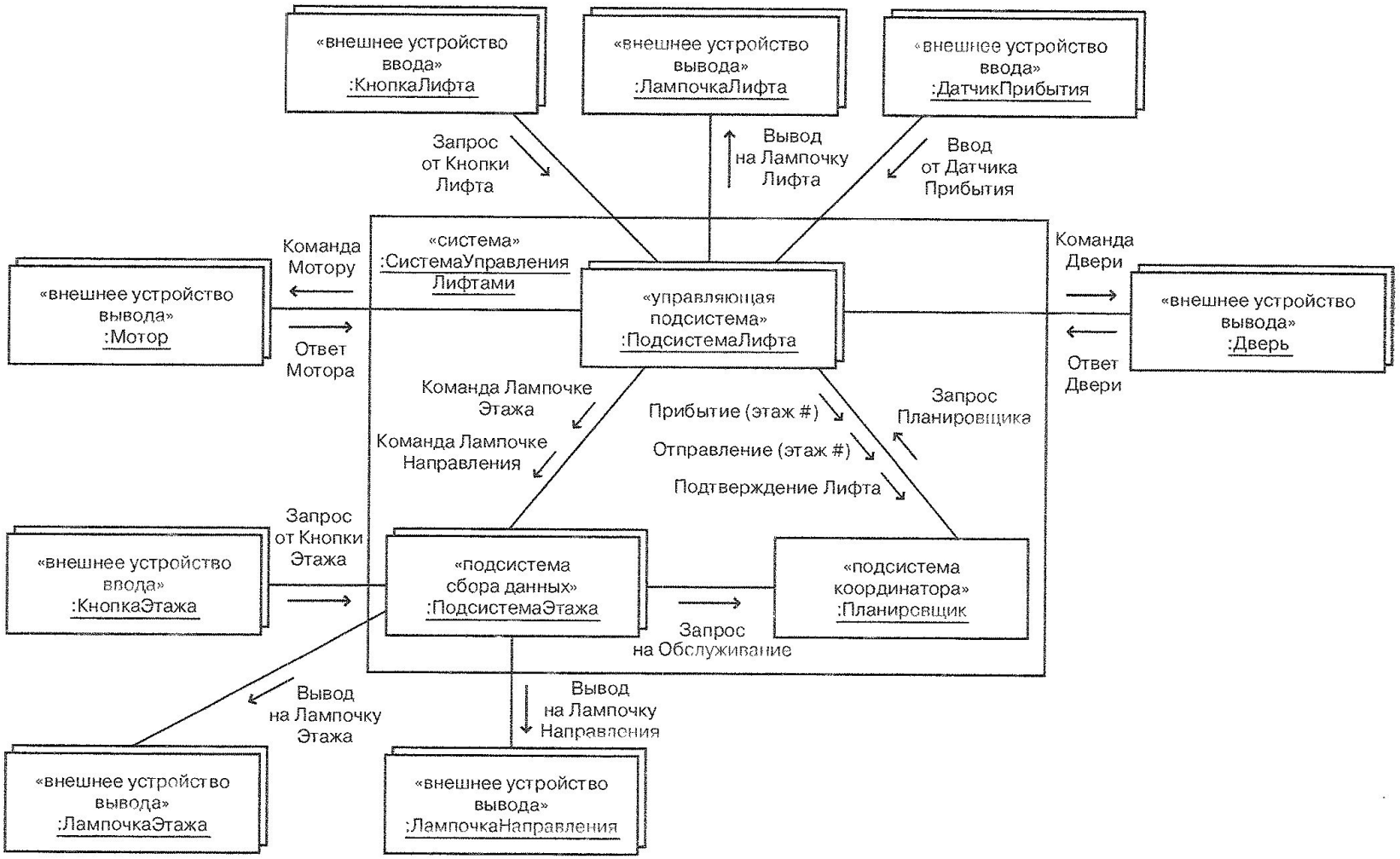
5. Объекты интерфейса пользователя должны находятся в отдельной подсистеме

- Обычно ПК пользователей являются частью крупной распределенной конфигурации, поэтому наиболее гибким будет решение, при котором объекты интерфейса пользователя находятся в отдельной подсистеме.
- Поскольку чаще всего такие объекты являются клиентами, данную рекомендацию можно рассматривать как частный случай рекомендаций, касающихся систем клиент-сервер.
- Объект интерфейса пользователя часто включает в себя более простые интерфейсные объекты.

Интерфейс с внешними объектами

- Подсистема имеет дело с частью множества актеров, показанных на модели прецедентов, и с частью множества внешних объектов реального мира, изображенных на диаграмме контекста.
- Внешние объекты должны иметь интерфейс только с одной подсистемой.
- Примеры для подсистем «Лифт» и «Этаж».

Пример распределенной архитектуры: система управления лифтами



6. Управляющий объект и управляемые им объекты, должны входить в одну подсистему

- Управляющий объект, а также те сущностные и интерфейсные объекты, которыми он манипулирует напрямую, должны входить в одну подсистему.
- Сущностный объект сильнее связан с теми объектами, которые его обновляют, чем с теми, которые извлекают из него данные.
- Поэтому в случае, когда есть выбор, сущностный объект следует помещать в одну систему с обновляющими его объектами

Виды подсистем

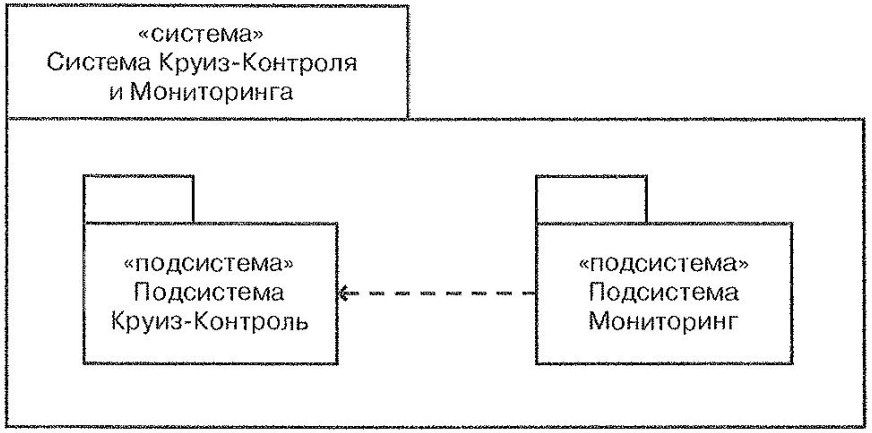
- Можно выделить следующие распространенные виды подсистем:
  1. подсистема управления;
  2. подсистема координации;
  3. подсистема сбора данных;
  4. подсистема анализа данных;
  5. серверная подсистема;
  6. подсистема интерфейса пользователя;
  7. подсистема ввода/вывода;
  8. системные службы.
- Такие виды подсистем, часто используются в
  - параллельных приложениях,
  - распределенных приложениях и
  - приложениях реального времени.
- Конкретный набор подсистем обычно определяется назначением приложения.

1. Подсистема управления

- ***Подсистема управления*** контролирует отдельной функцией всей системы.
- Функции подсистемы управления
  - получение входной информацию из внешней среды и
  - формирование выходной информацию для внешней среды (обычно без участия человека).
- Подсистема управления часто зависит от состояния, и тогда в ней есть хотя бы один зависящий от состояния управляющий объект.
- Иногда входные данных собираются другими подсистемами и используются подсистемой управления.
- Подсистема управления может предоставлять определенные данные и другим подсистемам.

Пример подсистемы управления: подсистема «Круиз-Контроль»

- Примером может служить подсистема «Круиз-Контроль» в системе круиз-контроля и мониторинга.
- Она получает входную информацию от ручки круиз-контроля, педали тормоза и двигателя.
- Выходная информация подается на дроссель.
- Решения о том, какое воздействие оказать на дроссель, зависят от состояния и принимаются без участия человека.



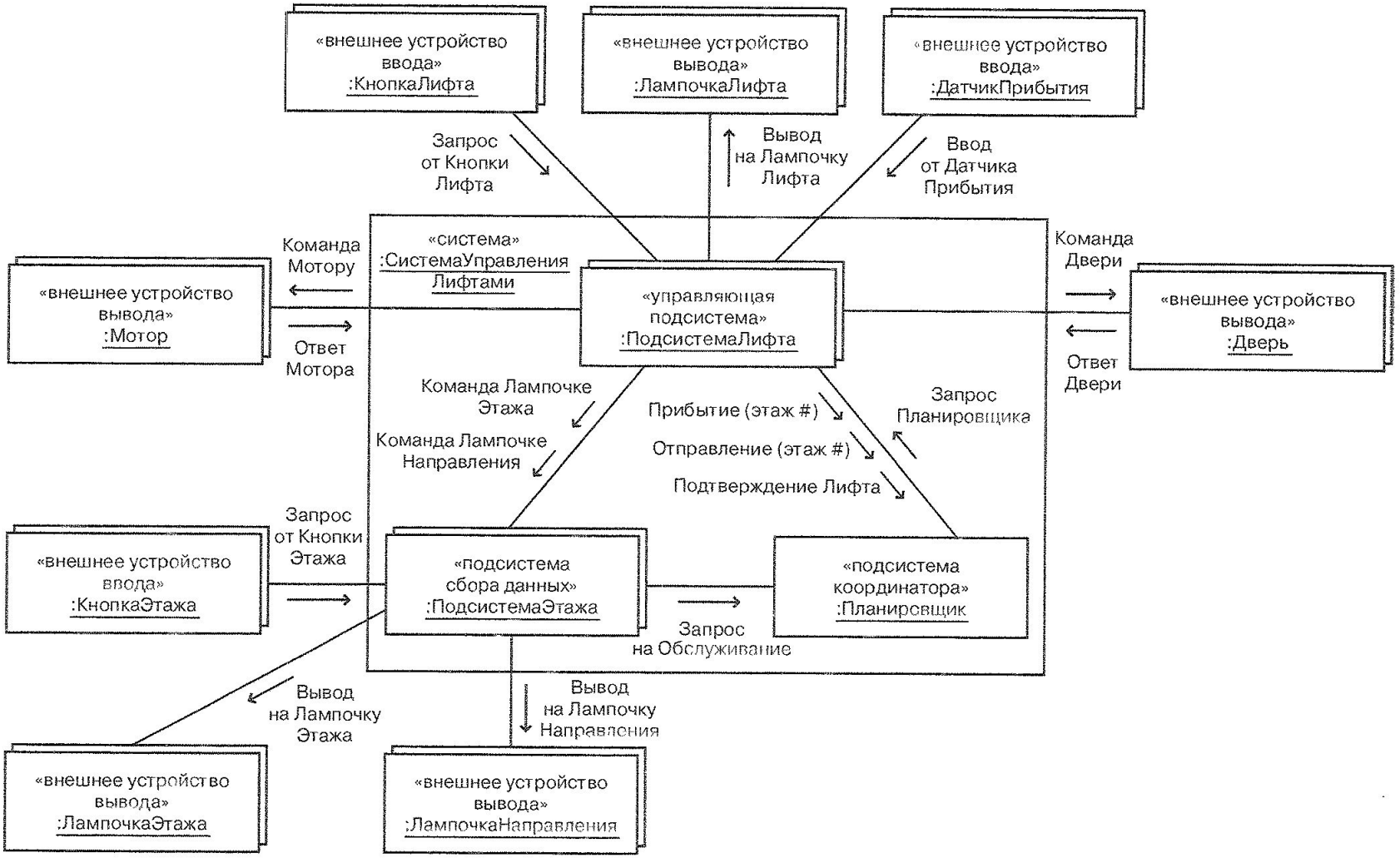
2. Подсистема координации

- Подсистема координации требуется если имеется несколько подсистем управления.
  - Если подсистемы управления абсолютно независимы, то координация не нужна.
- Функция подсистема координации состоит упорядочении действий подсистем управления.
- В некоторых случаях подсистемы управления самостоятельно координируют свою деятельность.
  - Такое возможно, если координация сравнительно проста, - например, между контроллерами рабочих станций в системе автоматизации производства.
- Если для координации необходимы значительные усилия, тогда лучше поручить это специальной подсистеме.
  - Например, подсистема координации может решить, какую часть работы должна выполнять каждая подсистема управления.

Пример: подсистема «Планировщик» в системе управления лифтами

- Здесь каждый запрос, сделанный пассажиром, должен быть обслужен лифтом, в котором он находится.
- Однако, если запрос поступает от человека, ожидающего на этаже, система вправе сама решить, какой из имеющихся лифтов обслужит запрос.
- Если лифт уже на пути к этажу и двигается в нужном направлении, то никаких специальных действий не требуется.
- В противном случае лифт нужно направить на этаж.
- Обычно решение принимается с учетом близости лифта к этажу и направления движения.
- Принятие решения возлагается на подсистему «Планировщик» (показана на следующем слайде).
- Когда подсистема «Планировщик» получает «Запрос на Обслуживание» от «Подсистемы Этажа», он решает, какому лифту передать запрос, после чего посылает «Запрос Планировщика» выбранной «Подсистеме Лифта».

Пример распределенной архитектуры: система управления лифтами

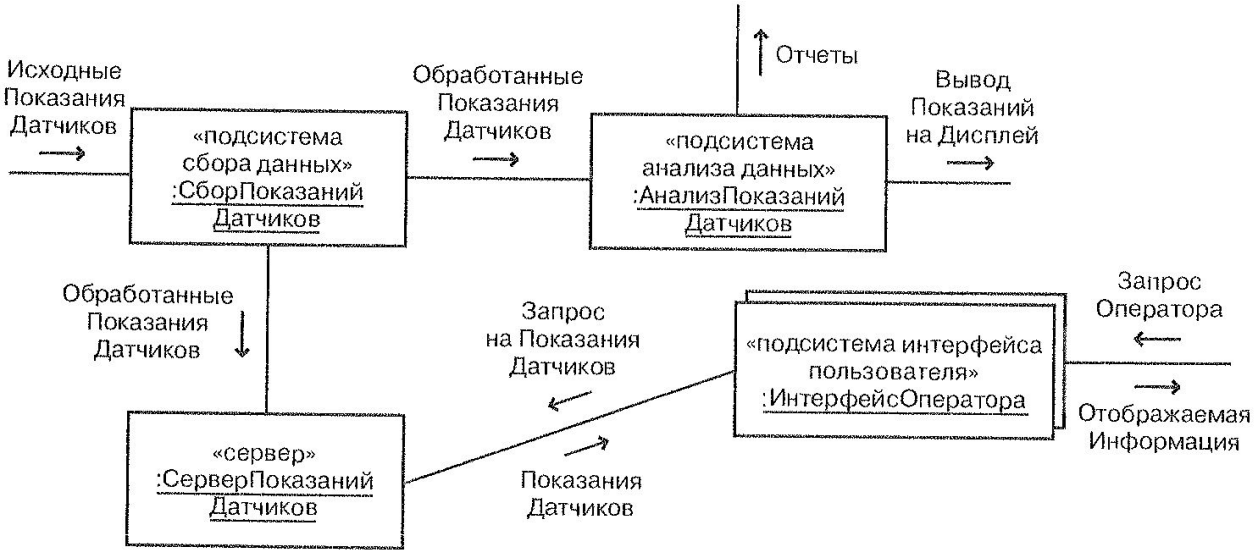


3. Подсистема сбора данных

- ***Подсистема сбора данных*** накапливает информацию, поступающую из внешней среды.
- Функции данной подсистемы:
  - может хранить сведения, прошедшие анализ и предварительную обработку.
  - может отвечать на запросы на получение исходных данных.
  - может частично обрабатывать собранную информацию:
    - Например, она способна передать усредненное значение нескольких показаний датчика, преобразовав его в инженерные единицы измерения.
  - может напрямую выводятся собранные данные во внешнюю среду.
- В разных реализациях подсистем сбора данных возможны также различные комбинации ее функций.

Пример подсистемы «Сбор Показаний Датчиков»

- Подсистемы «Сбор Показаний Датчиков» получает данные от различных цифровых и аналоговых датчиков в реальном времени.
- Частота опроса зависит от характеристик датчиков.
- Показания аналоговых датчиков преобразуются в инженерные единицы.
- Обработанные данные посылаются подсистемам-потребителям.
  - Например подсистеме «Анализ Данных».



4. Подсистема анализа данных

- Подсистема анализа данных предназначено для первичной обработки данных собранных другими подсистемами и предоставление их пользователям или посистемам.
- Функция подсистемы анализа данных:
  - первичная обработка (классификация) входных данных;
  - предоставление отчета;
  - визуальное отображение данных.
- В принципе сбором и анализом данных может заниматься одна и та же подсистема, как обстоит дело в случае подсистемы «Мониторинг» (система круиз-контроля).
- В других случаях поиск сведений осуществляется в реальном масштабе времени, а анализ производится позже.

Пример подсистема анализа данных

- Примером подсистемы анализа данных является подсистема «Анализ Показаний Датчиков» в системе охранной сигнализации):
  - получает данные от подсистемы «Сбор Показаний датчиков»,
  - анализирует текущие и прошлые показания,
  - выполняет статистический анализ (например, вычисление средних и стандартных отклонений),
  - выдает отчеты о трендах и генерирует сигналы

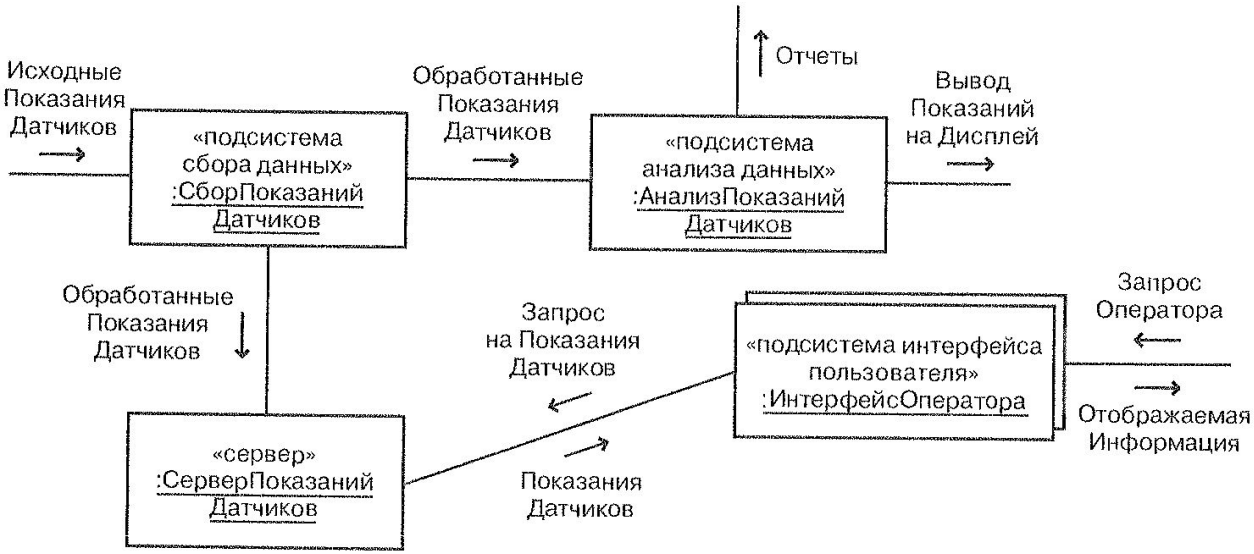
5. Серверная подсистема

- Серверная подсистема предназначена предоставлять сервисы другим подсистемам.
- Она отвечает на запросы клиентских подсистем, но сама не инициирует никаких запросов.
- Сервером может считаться любой объект, обслуживающий запросы клиентов.
- В простейшем случае объект-сервер может состоять всего из одного сущностного объекта.
- Более сложные серверы включают два и более объектов, в том числе
  - сущностные объекты;
  - координаторы, которые определяют, какой объект должен обслужить запрос клиента;
  - объекты бизнес-логики.
- Серверные подсистемы часто применяются в распределенных приложениях.

- Часто сервер предоставляет сервисы, связанные с одним или несколькими хранилищами данных, либо доступ к базе данных или ее части.
- Сервер может быть также ассоциирован с устройством ввода/вывода или набором таких устройств.
  - Например файл-серверы и серверы печати.

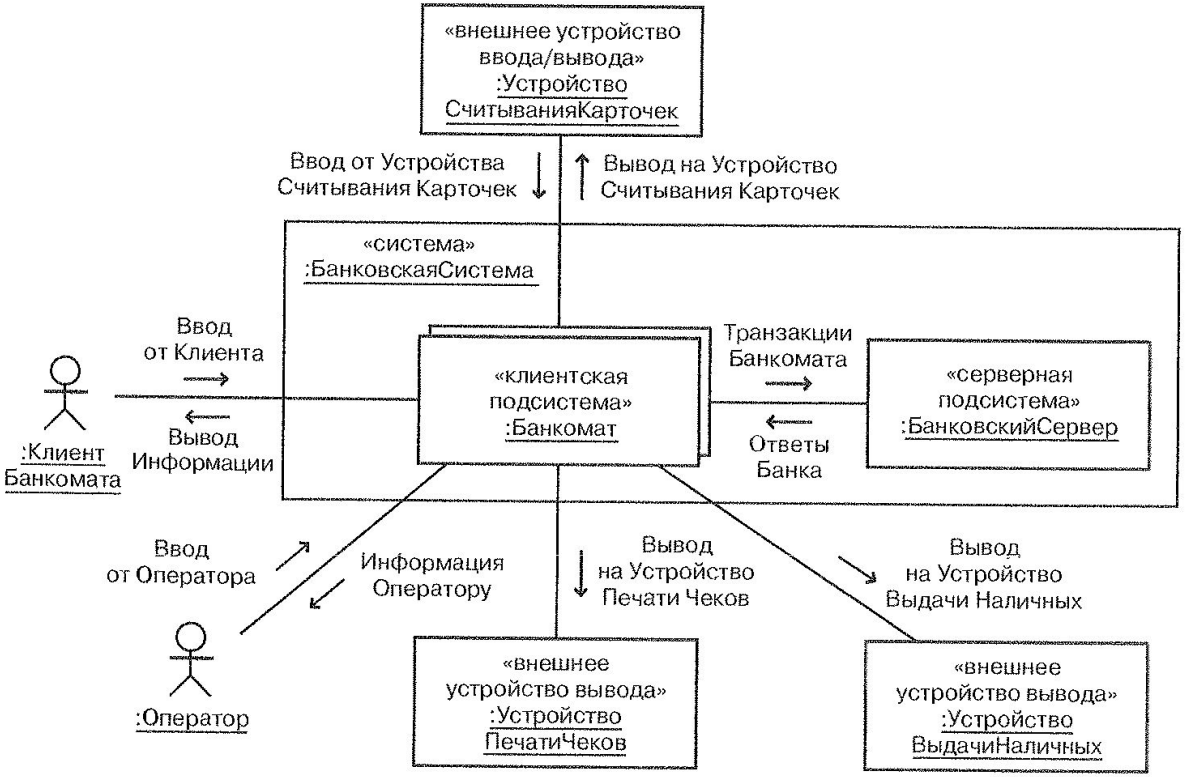
Пример сервера данных: «Сервер Показаний Датчиков»

- Функции «Сервера Показаний Датчиков»
  - хранит текущие и прошлые показания датчиков.
  - получает исходные данные от подсистемы «Сбор Показаний Датчиков».
  - предоставляются по запросам показания датчиков другим подсистемам,
    - Например: подсистеме «Интерфейс Оператора», которая отображает информацию.



Пример сервера данных: «Банковский Сервер»

- Подсистема «Банковский Сервер», которая обслуживает запросы от «Банкоматов».

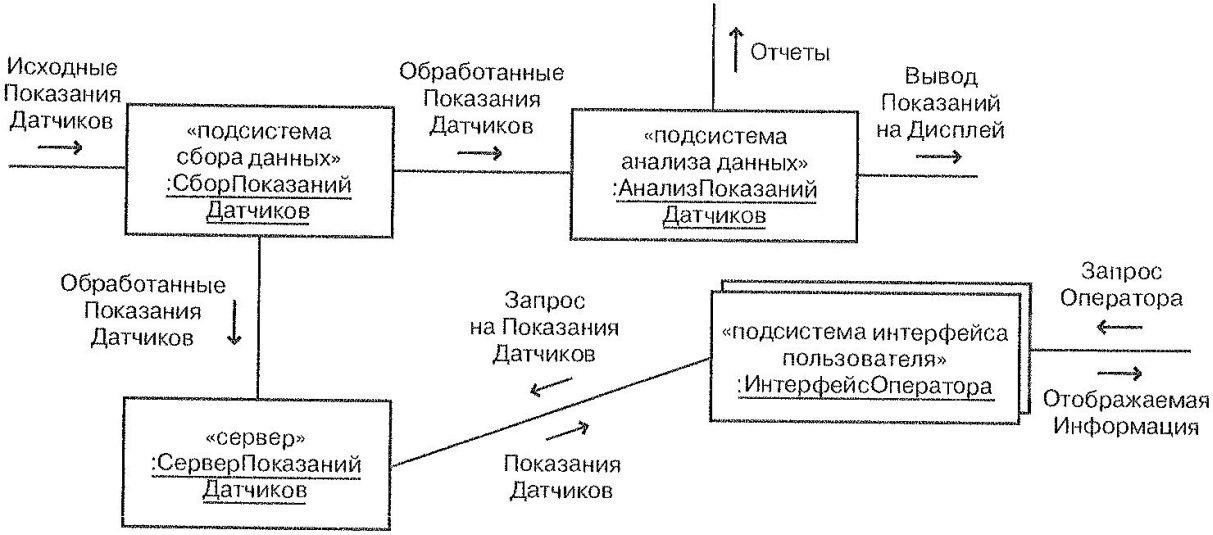


6. Подсистема интерфейса пользователя

- Подсистема интерфейса пользователя выступает в роли клиента, предоставляя пользователю доступ к сервисам одного или нескольких серверов.
- Таких подсистем может быть несколько, по одной для каждой категории пользователей.
- Обычно подсистемой интерфейса пользователя является объект, составленный из нескольких более простых интерфейсных объектов.

Пример подсистемы интерфейса пользователя

- Примером интерфейсной подсистемы является «Интерфейс Оператора», существующей в нескольких экземплярах.
- В системе автоматизации производства есть
  - подсистема интерфейса пользователя для дежурных операторов («Интерфейс Оператора») и
  - подсистема для менеджеров («Интерфейс Начальника Производства»).
- В подсистеме «Интерфейс Оператора» имеется объект для отображения тревожных сигналов и объект для индикации состояния рабочей станции.



7. Подсистема ввода/вывода

- В некоторых системах полезно сгруппировать все классы интерфейсов устройств в одну подсистему ввода/вывода, поскольку для разработки таких классов нужны специальные знания.
- Создавать и сопровождать подсистему ввода/вывода способны лишь немногие программисты обладающие специальными знаниями по работе в внешними устройствами .

8. Системные службы

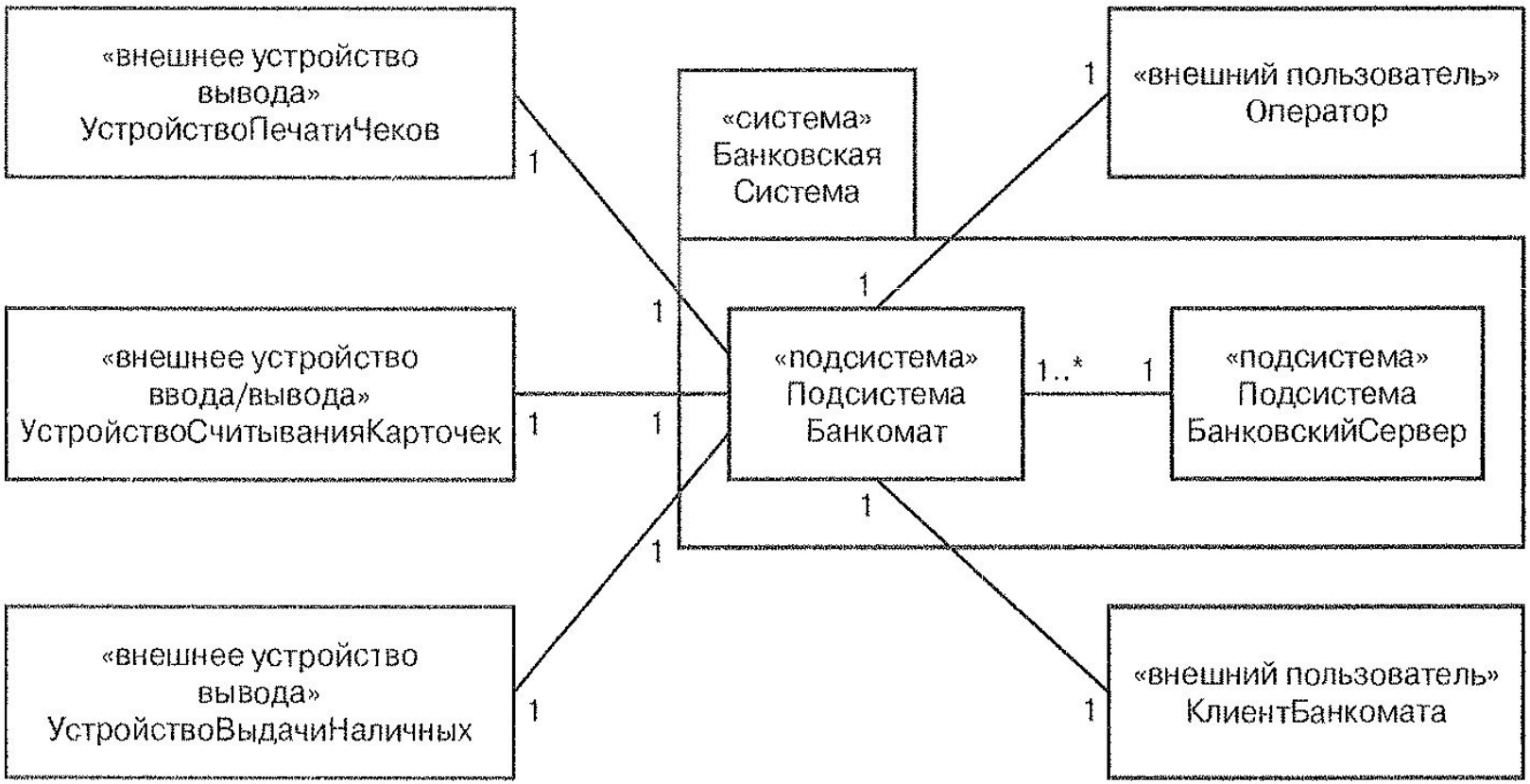
- Некоторые сервисы не связаны с конкретной предметной областью, а предоставляются самой системой.
  - Например управление файлами и сетью.
- Хотя такие подсистемы обычно не являются частью приложения, о них нужно помнить.
- Системные службы реализуются на уровне системы и используются прикладными подсистемами.
  - Примерами таких служб могут служить сервисы промежуточного слоя для организации взаимодействия между подсистемами.

Примеры разбиения ПО на подсистемы

Пример 1: разбиение банковской системы на подсистемы

- Декомпозиция банковской системы на клиент и сервер описана показана на следующем слайде.
- Банковская система - это пример трехуровневого приложения клиент-сервер.
- Поскольку третий уровень предоставляется системой управления базами данных, он не является частью приложения и отсутствует на диаграммах прикладного уровня.

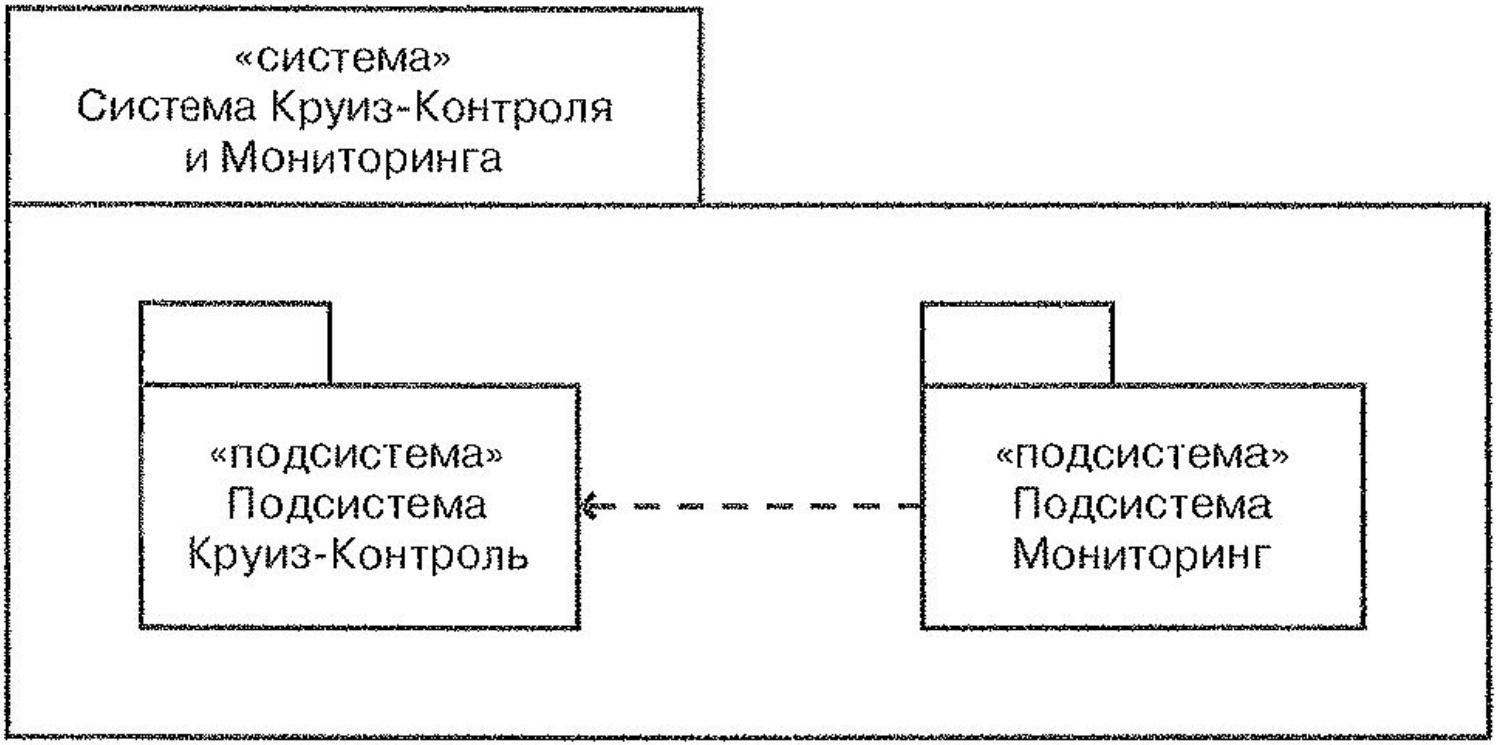
Пример программной архитектуры клиент-сервер: банковская система



Пример 2: разбиение системы круиз-контроля и мониторинга на подсистемы

- Анализ системы круиз-контроля и мониторинга показывает, что есть две относительно независимые и слабо связанные между собой подсистемы:
  - «Подсистема Круиз-Контроля» и
  - «Подсистема Мониторинга» (см. рис. 12.8).
- Первая является управляющей, а вторая обеспечивает сбор и анализ данных.
- Дальнейшее рассмотрение дает возможность определить интерфейс между этими подсистемами.
- Единственный элемент данных, необходимый обеим подсистемам, - это «Полный Пробег», который вычисляется «Подсистемой Круиз-

Система круиз-контроля и мониторинга: основные подсистемы



Пример 3: разбиение системы управления лифтами на подсистемы

- В качестве другого примера опишем систему управления лифтами.
- Изучая задачу, несложно обнаружить, что лифты и двери - это различные составные объекты.
- Число дверей обычно отличается от числа лифтов, но любая дверь состоит из одинакового набора объектов.
- То же самое можно сказать о каждом лифте.
- Таким образом, система управления лифтами распадается на
  - «Подсистему Этажа» и
  - «Подсистему Лифта»,
- причем экземпляров первой столько же, сколько этажей, а экземпляров второй столько же, сколько лифтов.
- При наличии нескольких лифтов необходимо координировать их работу.
- В частности, когда поступает запрос с этажа, нужно определить, какой лифт его обслужит.
- Поэтому требуется координирующая подсистема - Планировщик, которая направляет запрос конкретному лифту.

Пример распределенной архитектуры: система управления лифтами