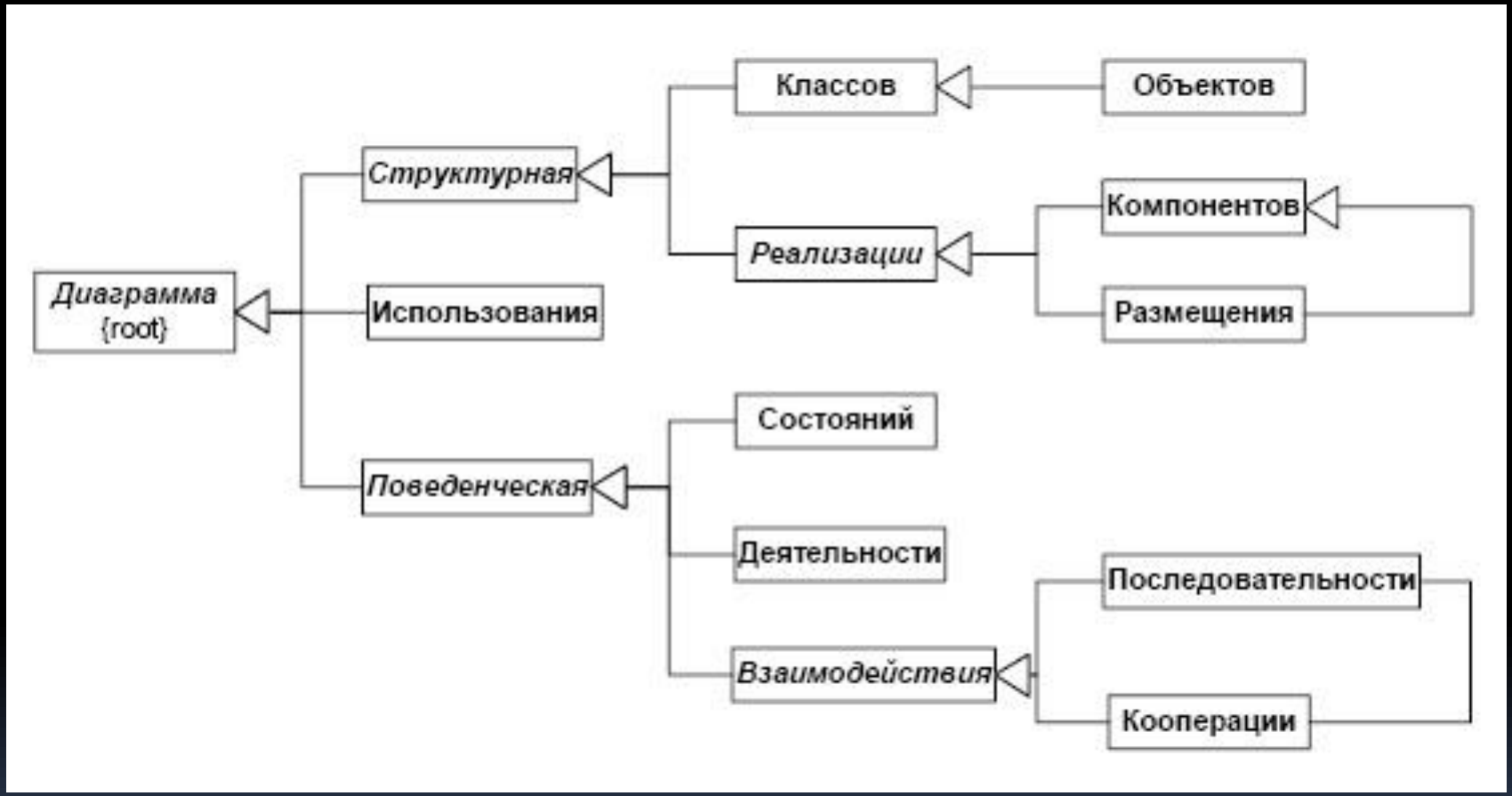
Анализ и проектирование на UML

Максим Валерьевич Хлопотов

Темы лекционных занятий

1. Введение в UML
2. Моделирование использования
3. **Моделирование структуры**
4. Моделирование поведения
5. Дисциплина моделирования

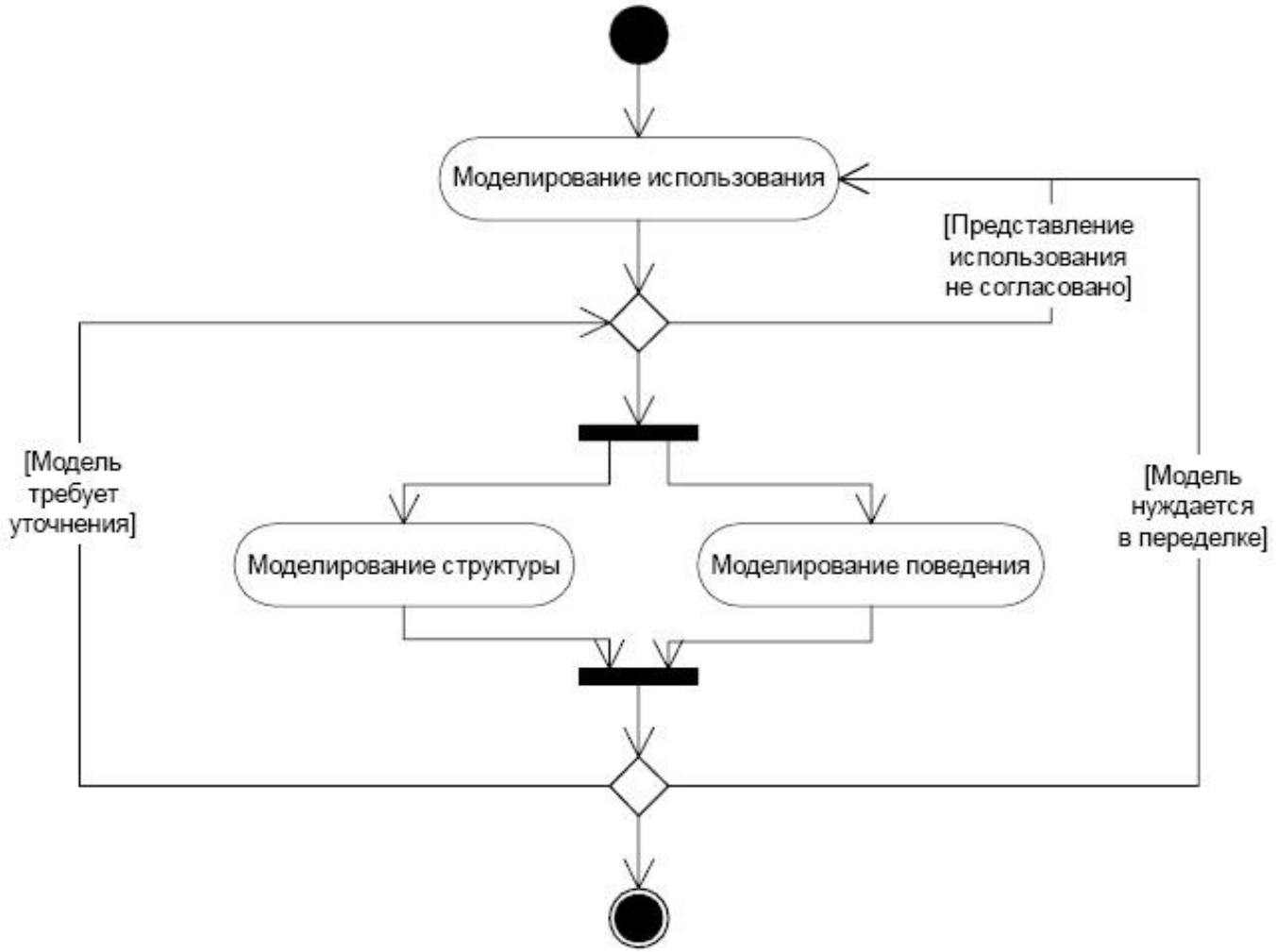
Иерархия диаграмм UML



Представления

Выделим три представления:
- представление использования (что делает система полезного?);
- представление структуры (из чего состоит система?);
- представление поведения (как работает система?).

Процесс моделирования



Процесс разработки

Начальная фаза — это краткий период формирования общего видения и рамок проекта

Фаза развития (elaboration) — это первая последовательность итераций, в течение которых решаются следующие задачи:

- Реализуются и тестируются базовые архитектурные элементы.
- Изучается и стабилизируется большая часть требований.
- Обосновываются и устраняются основные риски.

Процесс разработки

- На первой итерации *стадии развития* основное внимание уделяется фундаментальным вопросам ООА/П и построения объектной системы.
- Для построения программной системы необходимо рассмотреть и множество других вопросов, таких как разработка базы данных, исследование удобства использования и проектирование интерфейса пользователя.
- Однако эти вопросы выходят за рамки основ ООА/П и применения UML.

Процесс разработки

- Фаза развития включает несколько итераций фиксированной длительности (например, четыре недели), в течение которых реализуются наиболее рискованные, значимые и архитектурно важные части системы, а также идентифицируется и проясняется большая часть требований.
- Прецеденты упорядочиваются по приоритетности и сначала рассматривать самые важные. На каждом итерации нужно дорабатывать и уточнять основные требования. На начальных итерациях процесс изменения требований может происходить достаточно интенсивно. Однако в дальнейшем он стабилизируется.

Процесс разработки

- Примерный список артефактов, разработка которых начинается на этапе развития.

Модель предметной области

- Она представляет собой визуализацию понятий предметной области, напоминающую статическую модель сущностей предметной области

Модель проектирования

- Это набор диаграмм, описывающих логику проектного решения. Сюда относятся диаграммы программных классов, диаграммы взаимодействия объектов, диаграммы пакетов и т.д.

Прототипы интерфейса пользователя

Процесс разработки

Описание программной архитектуры

- Это документ, в котором рассмотрены основные архитектурные моменты и способы их реализации. В нем приводятся основные идеи проектного решения и обосновывается их целесообразность для данной системы. Сюда относится диаграмма размещения.

Модель данных

- Включает схему базы данных и стратегию отображения объектов в необъектное представление. Может быть показана на диаграмме классов

Процесс разработки

- После фазы развития начинается фаза конструирования.
- В начале фазы конструирования фактически уже имеется некоторый прототип, который на каждой последующей итерации дорабатывается.
- Фаза конструирования состоит из некоторого числа итераций фиксированной длительности, в течение которых основное внимание уделяется доработке системы, поскольку наиболее принципиальные вопросы уже решены на стадии развития.

Процесс разработки

- На этом этапе тоже некоторое внимание уделяется описанию прецедентов, однако значительно меньшее, чем на стадии развития.
- К этому моменту большая часть функциональных и нефункциональных требований должна быть постепенно стабилизирована. Это не означает необходимости замораживания требований и завершения исследований. Однако теперь степень модификации требований от итерации к итерации значительно снижается.

Моделирование структуры

- Моделируя структуру, мы описываем составные части системы и отношения между ними.
- UML является объектно-ориентированным языком моделирования, поэтому не удивительно, что основным видом составных частей, из которых состоит система, являются объекты.
- Однако, на UML можно моделировать и другие структуры.

Назначение структурного моделирования

Следующие структуры нужно моделировать на UML:

- структура связей между объектами во время выполнения программы;
- структура хранения данных;
- структура программного кода;
- структура компонентов в приложении;
- структура используемых вычислительных ресурсов;
- структура сложных объектов, состоящих из взаимодействующих частей
- структура артефактов в проекте.

Структурное моделирование

Структура связей между объектами во время выполнения программы

В парадигме ООП процесс выполнения программы состоит в том, что программные объекты взаимодействуют друг с другом, обмениваясь сообщениями. Наиболее распространенным типом сообщения является вызов метода объекта одного класса из метода объекта другого класса.

Для того чтобы вызвать метод объекта, нужно иметь доступ к этому объекту. Один объект "знает" другие объекты и, значит, может вызвать открытые методы, использовать и изменять значения открытых свойств и т.д.

В этом случае, мы говорим что объекты связаны. Для моделирования структуры связей в UML используются отношения ассоциации на диаграмме классов.

Структурное моделирование

Структура хранения данных

Программы обрабатывают данные, которые хранятся в памяти компьютера. В парадигме объектно-ориентированного программирования для хранения данных во время выполнения программы предназначены свойства объектов, которые моделируются в UML атрибутами классов.

Объекты, которые сохраняют (по меньшей мере) значения своих свойств даже после того, как завершился породивший их процесс, мы будем называть хранимыми.

В настоящее время самой распространенным способом хранения объектов является использование системы управления базами данных (СУБД). При этом хранимому классу соответствует таблица базы данных, а хранимый объект (точнее говоря, набор значений хранимых атрибутов) представляется записью в таблице.

Для моделирования структуры хранения данных в UML применяются ассоциации с указанием кратности полюсов.

Структурное моделирование

Структура программного кода

Программы существенно отличаются по величине: от сотен строк кода (и менее) до сотен миллионов строк (и более). Столь большие количественные различия не могут не проявляться и на качественном уровне.

Для маленьких программ структура кода практически не имеет значения, для больших — наоборот, имеет едва ли не решающее значение.

Поскольку UML не является языком программирования, модель не определяет структуру кода непосредственно, однако косвенным образом структура модели существенно влияет на структуру кода.

Структура классов и пакетов в модели UML фактически полностью моделирует структуру кода приложения.

Структурное моделирование

Структура сложных объектов, состоящих из взаимодействующих частей.

Для моделирования этой структуры применяется диаграмма внутренней структуры классификатора. В курсе не будем подробно рассматривать.

Часто этот тип диаграмм применяется для описания паттернов проектирования.

Структурное моделирование

Структура компонентов в приложении

Приложение, состоящее из одной исполнимой компоненты, имеет тривиальную структуру компонентов, моделировать которую нет нужды. Большинство современных приложений состоят из многих компонентов, даже если и не являются распределенными.

Компонентная структура предполагает описание двух аспектов: во-первых, как классы распределены по компонентам, во-вторых, как (через какие интерфейсы) компоненты взаимодействуют друг с другом.

Оба эти аспекта моделируются диаграммами компонентов UML.

Структурное моделирование

Структура используемых вычислительных ресурсов

Многокомпонентное приложение, как правило, бывает распределенным, т. е. различные компоненты выполняются на разных компьютерах.

Диаграммы размещения и компонентов позволяют включить в модель описание и этой структуры.

Диаграммы классов

- Диаграмма классов является основным средством моделирования структуры UML.
- Диаграммы классов наиболее информационно насыщены по сравнению с другими типами канонических диаграмм UML.
- На диаграммах классов в качестве сущностей применяются прежде всего классы, как в своей наиболее общей форме, так и в форме многочисленных стереотипов и частных случаев: интерфейсы, типы данных, процессы и др.
- Кроме того, в диаграмме классов могут использоваться (как и везде) пакеты и примечания.

Ассоциация на диаграмме классов

- Отношение ассоциации является самым важным на диаграмме классов.
- В общем случае ассоциация, которая обозначается сплошной линией, соединяющей классы, означает, что экземпляры одного класса связаны с экземплярами другого класса. Поскольку экземпляров может быть много, и каждый может быть связан с несколькими, ясно, что ассоциация является дескриптором, который описывает множество связанных объектов.
- В UML ассоциация является классификатором, экземпляры которого называются связями.

Список стандартных ассоциаций

- A является транзакцией, которая связана с другой транзакцией B
- A является элементом транзакции B
- A является товаром или услугой для транзакции B
- A является ролью, связанной с транзакцией B
- A является физической или логической частью B
- A физически или логически содержится в B
- A является описанием B
- A известен/зарегистрирован/записан/включен в B
- A является членом B
- A является организационной единицей B
- A использует, управляет или владеет B
- A следует за B

Ассоциации для POS-системы

- CashPayment-Sale (Платеж наличными-Продажа)
- SalesLineItem-Sale (Элемент продажи-Продажа)
- Item-SalesLineItem (Элемент-Элемент продажи)
- Customer- Payment (Покупатель-Платеж)
- Drawer-Register (Устройство печати торговых чеков-Реестр)
- Register-Store (Реестр-Магазин)
- Item-Shelf (Товар-Полка)
- ProductDescription-item (Описание товара-Товар)
- Sale-Register (Продажа-Реестр)
- Cashier-Store (Кассир-Магазин)
- Department-Store (Отдел-Магазин)
- Cashier-Register (Кассир-Реестр)
- SalesLineItem-SalesLineItem (Наименование товара-Следующее наименование товара)

Таблица соответствия

A является транзакцией, которая связана с другой транзакцией B	CashPayment-Sale (Платеж наличными-Продажа)
A является элементом транзакции	SalesLineItem-Sale (Элемент продажи-Продажа)
A является товаром или услугой для транзакции B	Item-SalesLineItem (Элемент-Элемент продажи)
A является ролью, связанной с транзакцией B	Customer- Payment (Покупатель-Платеж)
A является физической или логической частью B	Drawer-Register (Устройство печати торговых чеков-Реестр)
A физически или логически содержится в B	Register-Store (Реестр-Магазин)
	Item-Shelf (Товар-Полка)
A является описанием B	ProductDescription-Item (Описание товара-Товар)
A известен/зарегистрирован/записан/включен в B	Sale-Register (Продажа-Реестр)
A является членом B	Cashier-Store (Кассир-Магазин)
A является организационной единицей B	Department-Store (Отдел-Магазин)
A использует, управляет или владеет B	Cashier-Register (Кассир-Реестр)
A следует за B	SalesLineItem-SalesLineItem (Наименование товара-Следующее наименование товара)

Интерфейс

- В UML имеется несколько частных случаев классификаторов, которые, подобно классам, предназначены для моделирования структуры, но обладают рядом специфических особенностей. Наиболее важным из них являются интерфейсы.
- Интерфейс — это именованный набор абстрактных операций.
- Другими словами, интерфейс — это абстрактный класс, в котором нет атрибутов и все операции абстрактны. Поскольку интерфейс — это абстрактный класс, он не может иметь непосредственных экземпляров.
- Роль — это интерфейс, который предоставляет классификатор в данной ассоциации.

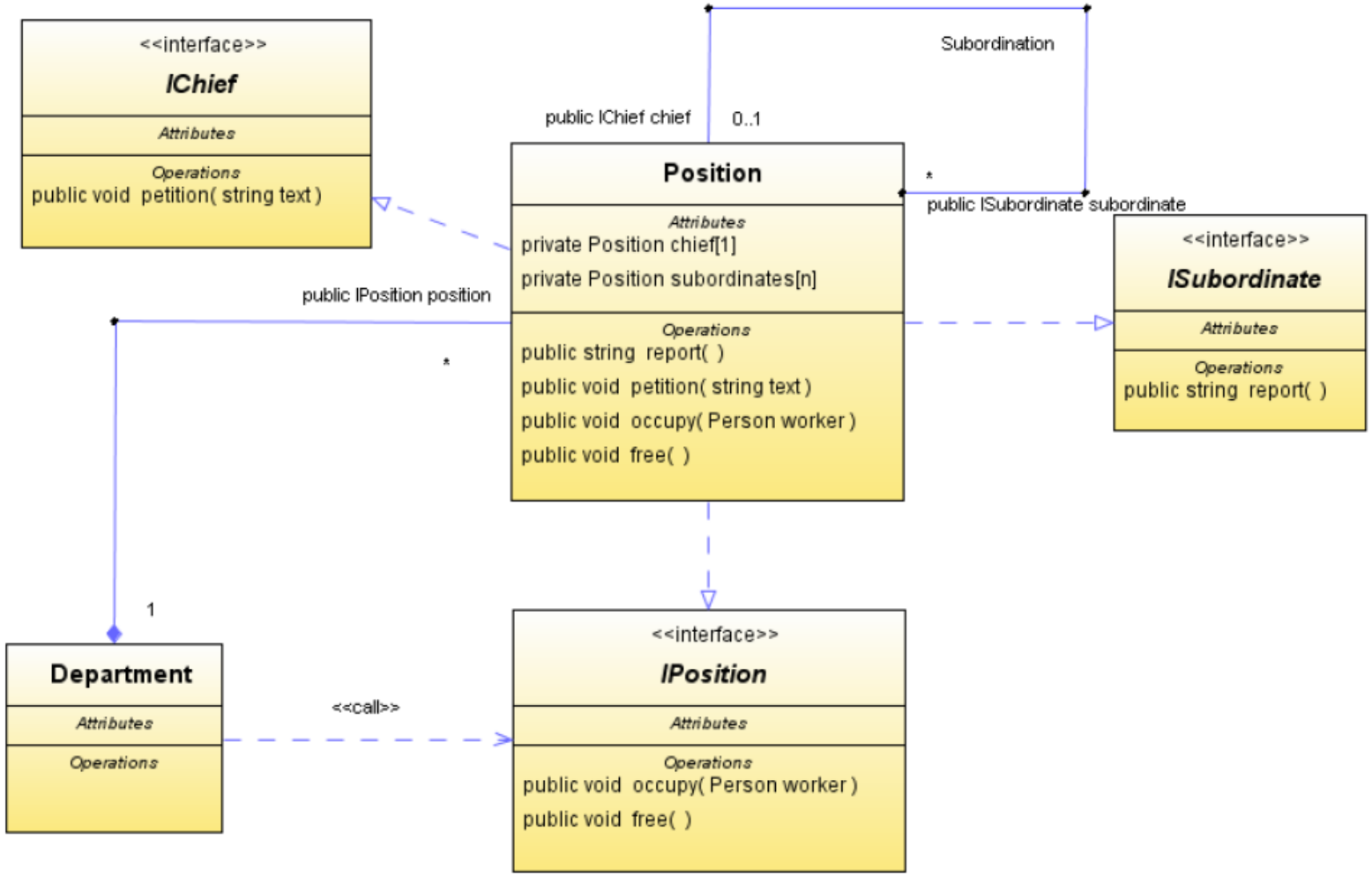
Интерфейс

- Между интерфейсами и другими классификаторами, в частности классами, на диаграмме классов применяются два отношения:
- классификатор (в частности, класс) использует интерфейс — это показывается с помощью зависимости со стереотипом «call»;
- классификатор (в частности, класс) реализует интерфейс — это показывается с помощью отношения реализации.

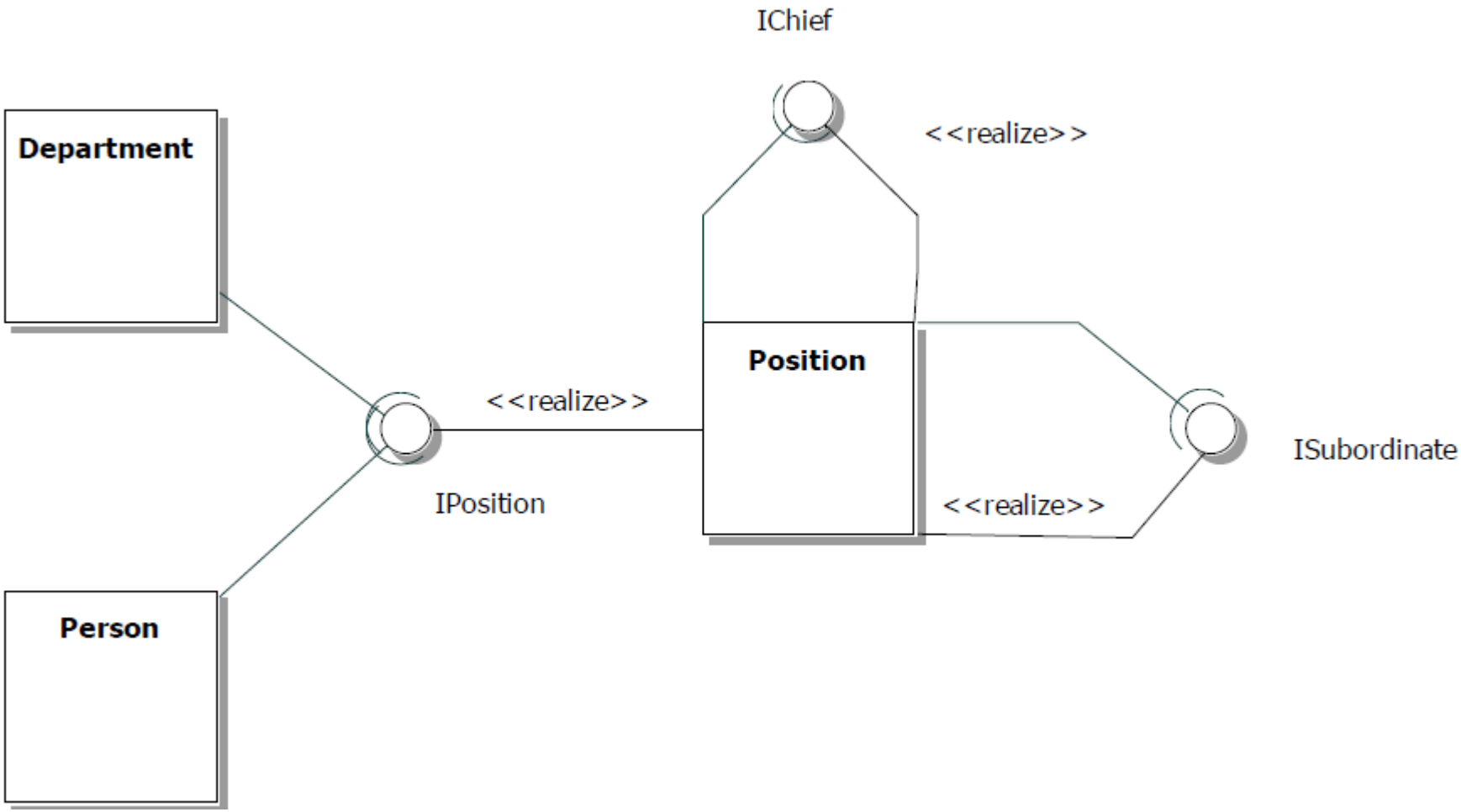
Интерфейс

- Допустим, что класс Department для реализации операций связанных с движением кадров, использует операции класса Position, позволяющие занимать и освобождать должность — другие операции класса Position классу Department не нужны. Для этого можно определить соответствующий интерфейс IPosition и связать его отношениями с данными классами.

Интерфейс


<<interface>>
IChief
Attributes
Operations
public void petition(string text)
Subordination
public IChief chief
0..1
Position
Attributes
private Position chief[1]
private Position subordinates[n]
Operations
public string report( )
public void petition(string text)
public void occupy(Person worker)
public void free( )
*
public ISubordinate subordinate
<<interface>>
ISubordinate
Attributes
Operations
public string report( )
public IPosition position
*
1
Department
Attributes
Operations
<<call>>
<<interface>>
IPosition
Attributes
Operations
public void occupy(Person worker)
public void free( )

Интерфейс


IChief
<<realize>>
Department
Position
<<realize>>
ISubordinate
IPosition
<<realize>>
Person

Типы данных

- *Тип данных* — это совокупность множества значений (может быть очень большого или даже потенциально бесконечного) и конечного множества операций, применимых к данным значениям.
- UML не является сильно типизированным языком: например, в модели можно указывать типы атрибутов классов и параметров операций, но это не обязательно.

Типы данных

Для каких элементов модели можно указать тип?

В UML типизированы могут быть:

- атрибуты классов, в том числе классов ассоциаций;
- параметры операций, в том числе тип возвращаемого значения;
- роли полюсов ассоциаций;
- параметры шаблонов (см. ниже).

Типы данных

В модели UML можно использовать три вида типов данных:

- Примитивные типы, которые считаются предопределенными в UML
- Типы данных, которые определены в языке программирования, поддерживаемым инструментом
- Типы данных, которые определены в модели

Типы данных

- Типы данных, которые определены в модели. В стандарте UML предусмотрен конструктор типов данных: перечислимый тип, который определяется с помощью стереотипа «enumeration».
- Наряду со стандартным стереотипом «enumeration» многие инструменты допускают использование стереотипа «datatype», который означает построение типа данных с помощью не специфицированного конструктора типов.

Типы данных

- Рекомендации, определяющие необходимость использования в модели дополнительных типов данных.
- Тип данных, изначально рассматриваемый как число или строка, может быть представлен в виде нового типа данных в модели предметной области в следующих случаях.
- Если он составлен из отдельных частей. (Номер телефона, имя человека).
- Если с этим типом обычно ассоциируются операции, такие как синтаксический анализ и проверка. (Номер страхового полиса).
- Если он содержит другие атрибуты. (Для льготной цены могут устанавливаться сроки действия - начало и конец).
- Если этот тип используется для задания количества с единицами измерения. (Стоимость товара измеряется в некоторых единицах).
- Это абстракция одного или нескольких типов.

Диаграммы классов

- Диаграммы классов содержат множество деталей. Для практически значимых систем диаграммы классов в конечном итоге получаются довольно сложными. Пытаться прорисовать сложную диаграмму классов сразу "на всю глубину" нерационально — слишком велик риск "утонуть" в деталях.
- Удачная модель структуры сложной системы создается за несколько (может быть даже за несколько десятков) итераций, в которых моделирование структуры перемежается моделированием поведения.

Диаграммы классов

- Описывать структуру удобнее параллельно с описанием поведения. Каждая итерация должна быть небольшим уточнением как структуры, так и поведения.
- Не обязательно включать в модель все классы сразу. На первых итерациях достаточно идентифицировать очень небольшую (10%) долю всех классов системы.
- Не обязательно определять все свойства класса сразу. Начните с имени — операции и атрибуты постепенно выявятся в процессе моделирования поведения.
- Не обязательно показывать на диаграмме все свойства класса. В процессе работы диаграмма должна легко охватываться одним взглядом.
- Не обязательно определять все отношения между классами сразу. Пусть класс на диаграмме "висит в воздухе" — ничего с ним не случится.

Выводы

- Диаграммы классов моделируют структуру объектов и связей между ними.
- Классы выбираются на основе анализа предметной области, взаимного согласования элементов модели и общих теоретических соображений.
- Сущности на диаграммах классов связываются главным образом отношениями ***ассоциации*** (в том числе агрегирования и композиции) и ***обобщения***.
- Базовая нотация ассоциации позволяет указать, что объекты ассоциированных классов могут взаимодействовать во время выполнения. Для ассоциации в UML предусмотрено наибольшее количество различных дополнений.

Выводы

- В UML имеется несколько частных случаев классификаторов, которые, подобно классам, предназначены для моделирования структуры, но обладают рядом специфических особенностей. Наиболее важным из них являются интерфейсы и типы данных.
- Описывать структуру удобнее параллельно с описанием поведения. Каждая итерация должна быть небольшим уточнением как структуры, так и поведения.

Диаграммы реализации

- Диаграммы реализации — это обобщающее название для диаграмм компонентов и диаграмм размещения. Название объясняется тем, что данные диаграммы приобретают особую важность на позднейших фазах разработки — на фазах реализации и перехода. На ранних фазах разработки — анализа и проектирования — эти диаграммы либо вообще не используются, либо имеют самый общий, не детализированный вид.

Диаграммы реализации

- Диаграммы компонентов и размещения имеют больше общего, чем различий, поскольку предназначены для моделирования двух теснейшим образом связанных структур:
- структуры компонентов в приложении;
- структуры используемых вычислительных ресурсов.

Диаграммы компонентов

- Диаграмма компонентов предназначена для перечисления и указания взаимосвязей артефактов моделируемой системы.
- На диаграмме компонентов применяются следующие основные типы сущностей:
  - компоненты;
  - интерфейсы;
  - классы;
  - объекты.
- На диаграмме компонентов обычно используются отношения следующих типов:
  - зависимость;
  - ассоциация (главным образом в форме композиции);
  - реализация.

Компонент

- Абсолютно формальное определение компонента в UML дать невозможно.
- Компонент — это физически существующий и заменяемый артефакт системы.
- Прежде всего, это компонент в смысле сборочного программирования: особым образом оформленная часть программного кода, которая может работать в составе более сложной программы.
- Компоненты понимаются в UML в наиболее общем смысле: это не только исполнимые файлы с кодами программы, но и исходные тексты программ, веб-страницы, справочные файлы, сопроводительные документы, файлы с данными и вообще любые артефакты, которые тем или иным способом используются при работе приложения и входят в его состав.

Компонент

- При выделении компонентов применяются следующие неформальные критерии.
  - Компонент нетривиален. Это нечто более сложное и объемное, чем фрагмент кода или одиночный класс.
  - Компонент независим, но не самодостаточен. Он содержит все, что нужно для функционирования, но предназначен для работы во взаимодействии с другими компонентами.
  - Компонент однороден. Он выполняет несколько взаимосвязанных функций, которые могут быть естественным образом охарактеризованы как единое целое в контексте более сложной системы.
  - Компонент заменяем. Он поддерживает строго определенный набор интерфейсов и может быть без ущерба для функционирования системы заменен другим компонентом, поддерживающим те же интерфейсы.

Диаграмма компонентов

- В случае разработки "монолитного" настольного приложения диаграмма компонентов не нужна — она оказывается тривиальной и никакой полезной информации не содержит.
- Таким образом, диаграммы компонентов применяются только при моделировании многокомпонентных приложений.

Диаграмма компонентов

- Если приложение поставляется в виде "конструктора" (набора "кубиков" — компонентов) из которого при установке собирается конкретный уникальный экземпляр приложения, то диаграммы компонентов оказываются просто незаменимым средством.
- Многие современные приложения поставляются в виде большого (десятки и сотни) набора компонентов, из которых "на месте" собирается нужная пользователю, часто уникальная конфигурация.
- Рекомендуют использовать диаграммы компонентов для управления конфигурацией не только на фазе поставки и установки программного обеспечения, но и в процессе разработки: для отслеживания версий компонентов, вариантов сборки и т. п.

Зависимость компонентов

компонент с именем Control зависит от *импортируемого интерфейса* IDialog, который, в свою очередь, реализуется *компонентом* с именем DataBase.

При этом для второго *компонента* этот *интерфейс* является *экспортируемым*.

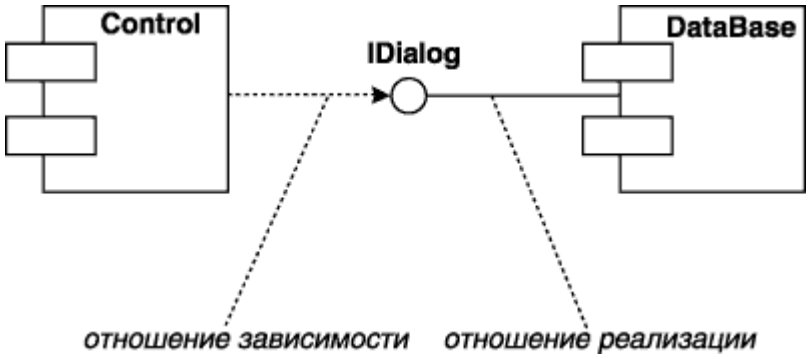


Диаграмма компонентов

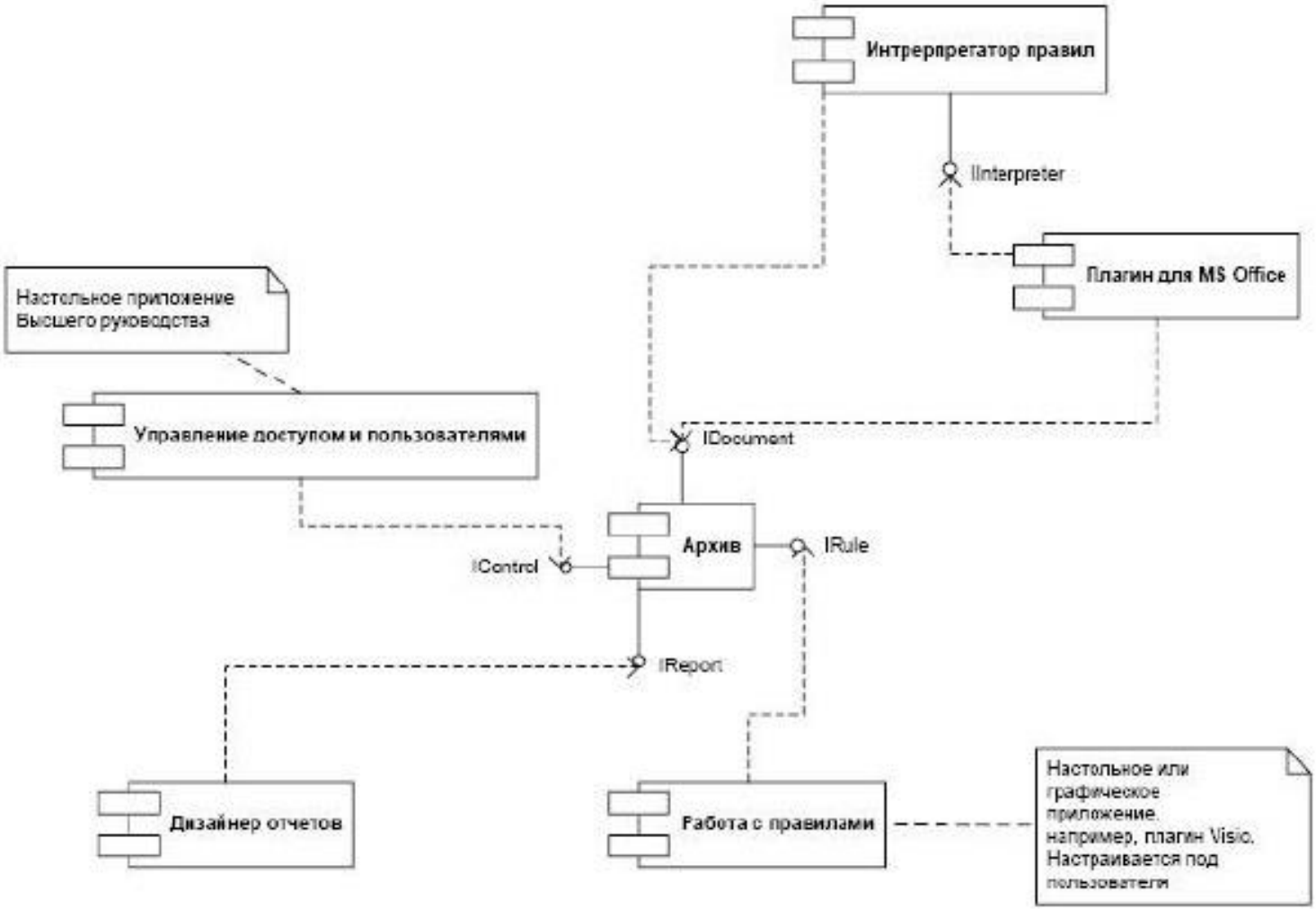


Диаграмма размещения

- Последним структурным аспектом, который необходимо обсудить, является описание размещения компонентов относительно участвующих в работе вычислительных ресурсов.
- В UML для этой цели предназначены диаграммы размещения. В UML 2.0 эти диаграммы переименованы в диаграммы развертывания.
- Если речь идет о настольном приложении, которое целиком хранится и выполняется на одном компьютере, то отдельная диаграмма размещения не нужна.
- При моделировании распределенных приложений значение диаграмм размещения резко возрастает.

Диаграмма размещения

- Диаграммы размещения наследуют возможности диаграмм компонентов.
- На диаграмме размещения, по сравнению с диаграммами компонентов, применяются дополнительный тип сущности — узел, имеющий два базовых стереотипа «device» и «execution environment».
- Узлы связываются между собой ассоциациями (чаще всего с именем).
- На узлах размещаются артефакты (физически реализованные компоненты).

Диаграмма размещения

- Узел — это физический вычислительный ресурс, участвующий в работе системы.
- Компоненты системы во время ее работы размещаются на узлах. В UML узел является классификатором, т. е. мы можем (и должны!) различать описание типа вычислительного ресурса (например, рабочая станция, последовательный порт) и описание экземпляра вычислительного устройства (например, устройство COM1 типа последовательный порт).
- Это различие моделируется согласно общему механизму UML: имя экземпляра узла подчеркивается, а имя типа узла — нет.

Диаграмма размещения

- На диаграмме узел представляется фигурой, изображающей прямоугольный параллелепипед.
- На примере (а) - имя типа узла, (б) – имя экземпляра узла .

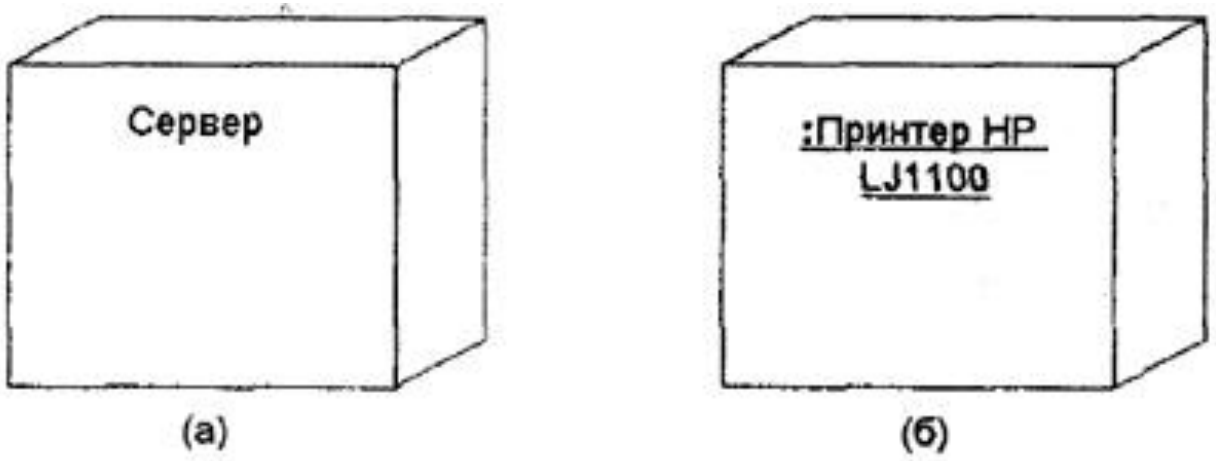


Диаграмма размещения

- Ассоциация между узлами означает то же, что и в других контекстах: возможность обмена сообщениями.
- Применительно к вычислительным сетям ассоциация означает наличие канала связи. Если нужно указать дополнительную информацию о свойствах канала, то это можно сделать используя общие механизмы: стереотипы, ограничения и именованные значения, приписанные ассоциации.

Диаграмма размещения

- Размещение компонента на узле, как правило, изображают, помещая фигуру компонента внутрь фигуры узла.

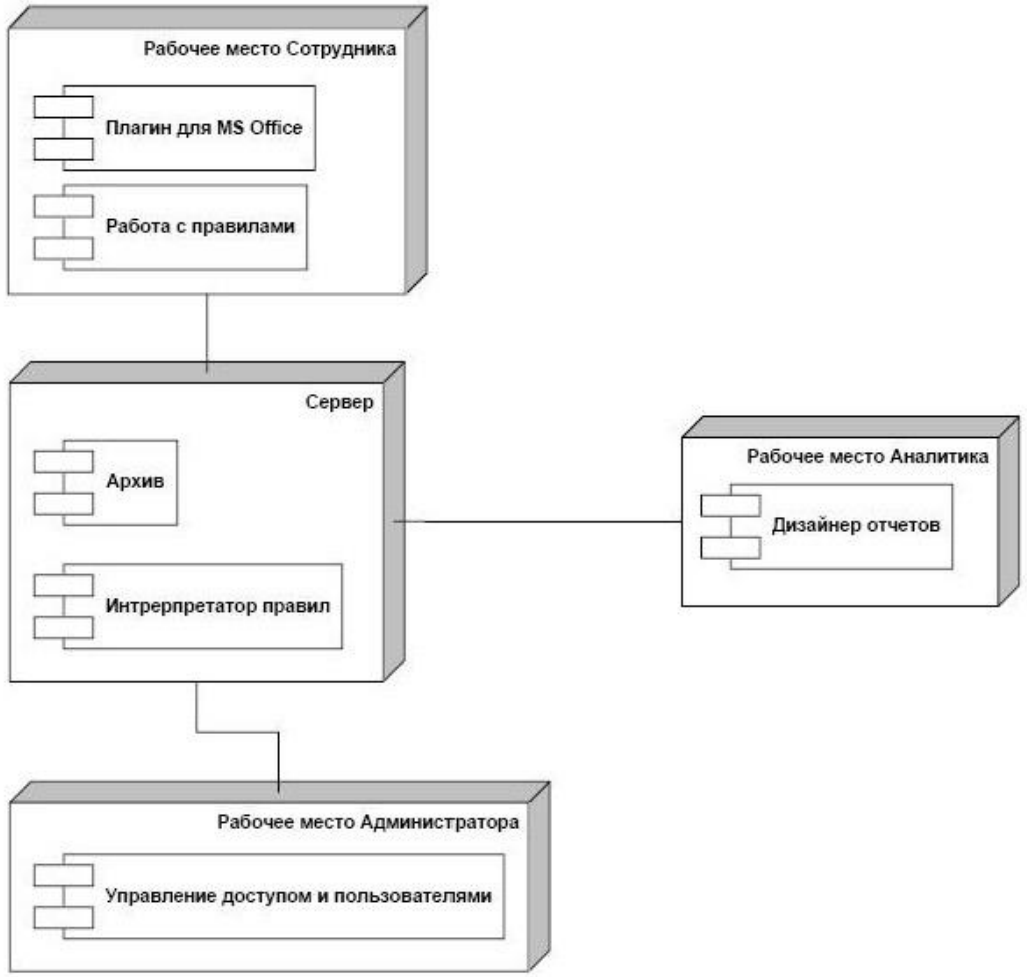
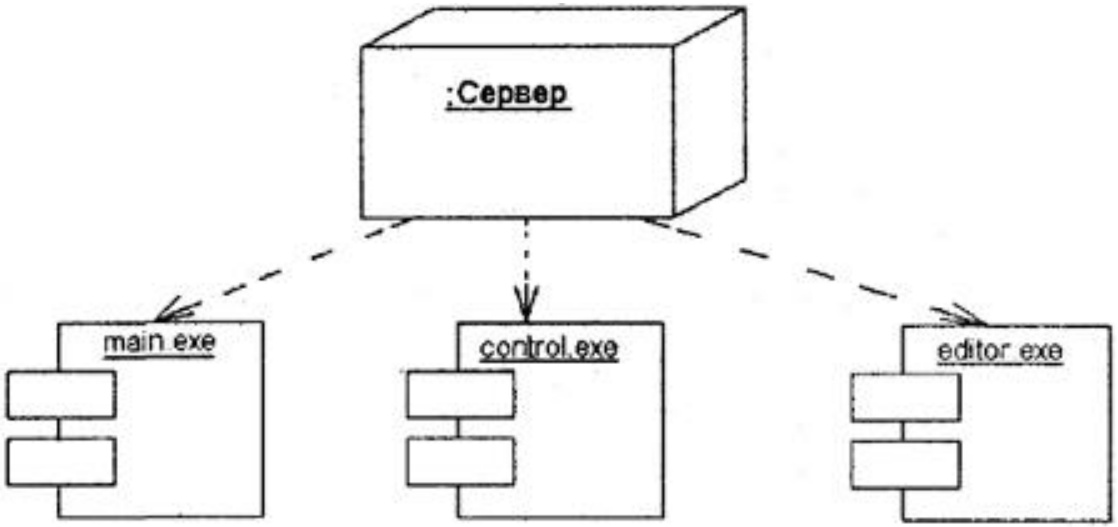
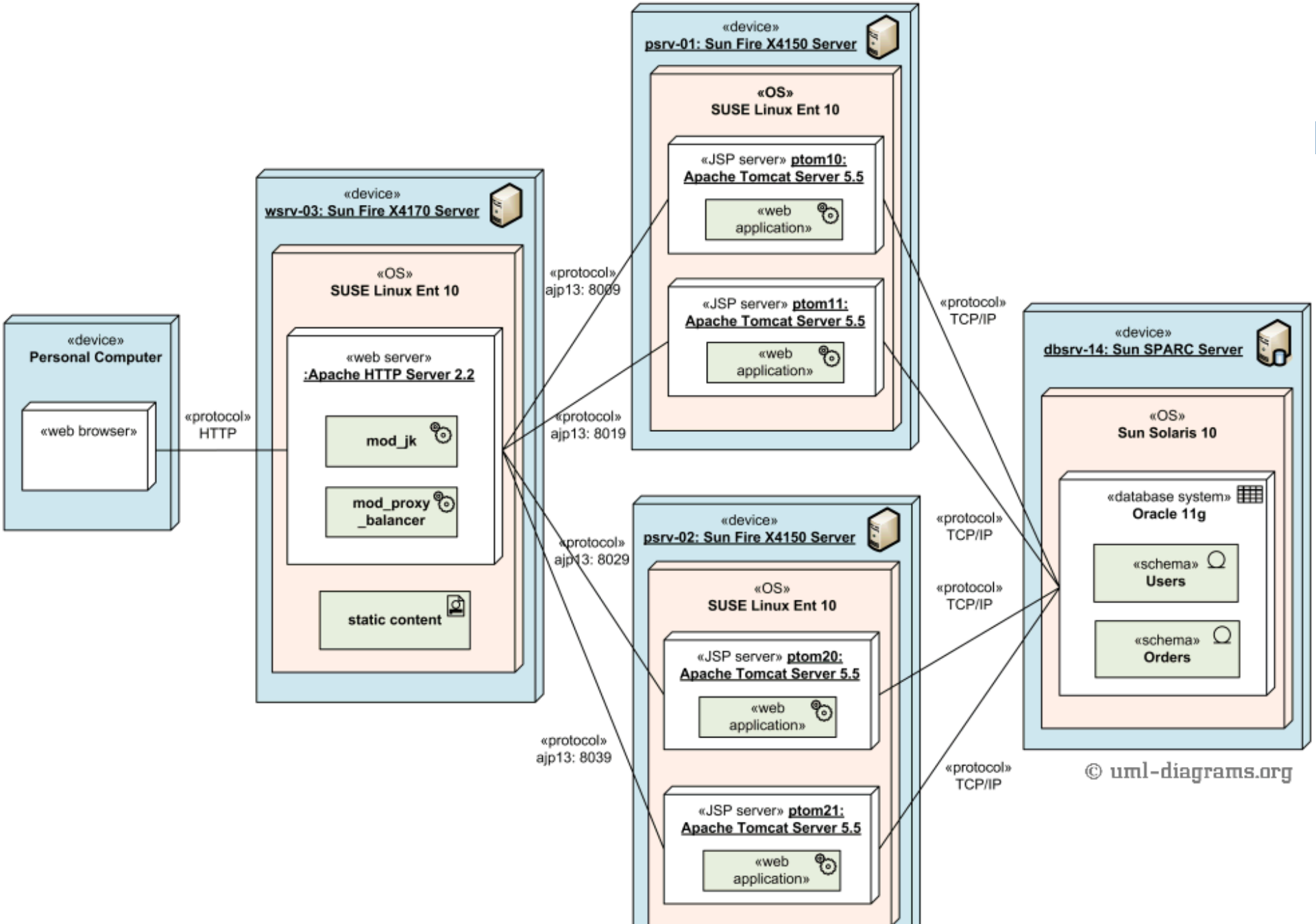


Диаграмма размещения

- Если это по каким-либо причинам неудобно, то отношение размещения можно передать отношением зависимости от узла к компоненту.




«device»
Personal Computer
«web browser»
«protocol»
HTTP
«device»
wsrv-03: Sun Fire X4170 Server
«OS»
SUSE Linux Ent 10
«web server»
:Apache HTTP Server 2.2
mod_jk
mod_proxy
_balancer
static content
«protocol»
ajp13: 8009
«protocol»
ajp13: 8019
«protocol»
ajp13: 8029
«protocol»
ajp13: 8039
«device»
psrv-01: Sun Fire X4150 Server
«OS»
SUSE Linux Ent 10
«JSP server» ptom10:
Apache Tomcat Server 5.5
«web
application»
«JSP server» ptom11:
Apache Tomcat Server 5.5
«web
application»
«device»
psrv-02: Sun Fire X4150 Server
«OS»
SUSE Linux Ent 10
«JSP server» ptom20:
Apache Tomcat Server 5.5
«web
application»
«JSP server» ptom21:
Apache Tomcat Server 5.5
«web
application»
«protocol»
TCP/IP
«protocol»
TCP/IP
«protocol»
TCP/IP
«protocol»
TCP/IP
«device»
dbsrv-14: Sun SPARC Server
«OS»
Sun Solaris 10
«database system»
Oracle 11g
«schema»
Users
«schema»
Orders
© uml-diagrams.org

Ошибки на диаграммах размещения

- Нужно помнить модель OSI
- Очень часто пытаются связать устройство со средой исполнения на другой устройстве или два устройства по протоколу HTTP.
- Не надо так.