Курс: «Математические модели комплексов программ»

Модуль: **«Архитектура программных систем»**

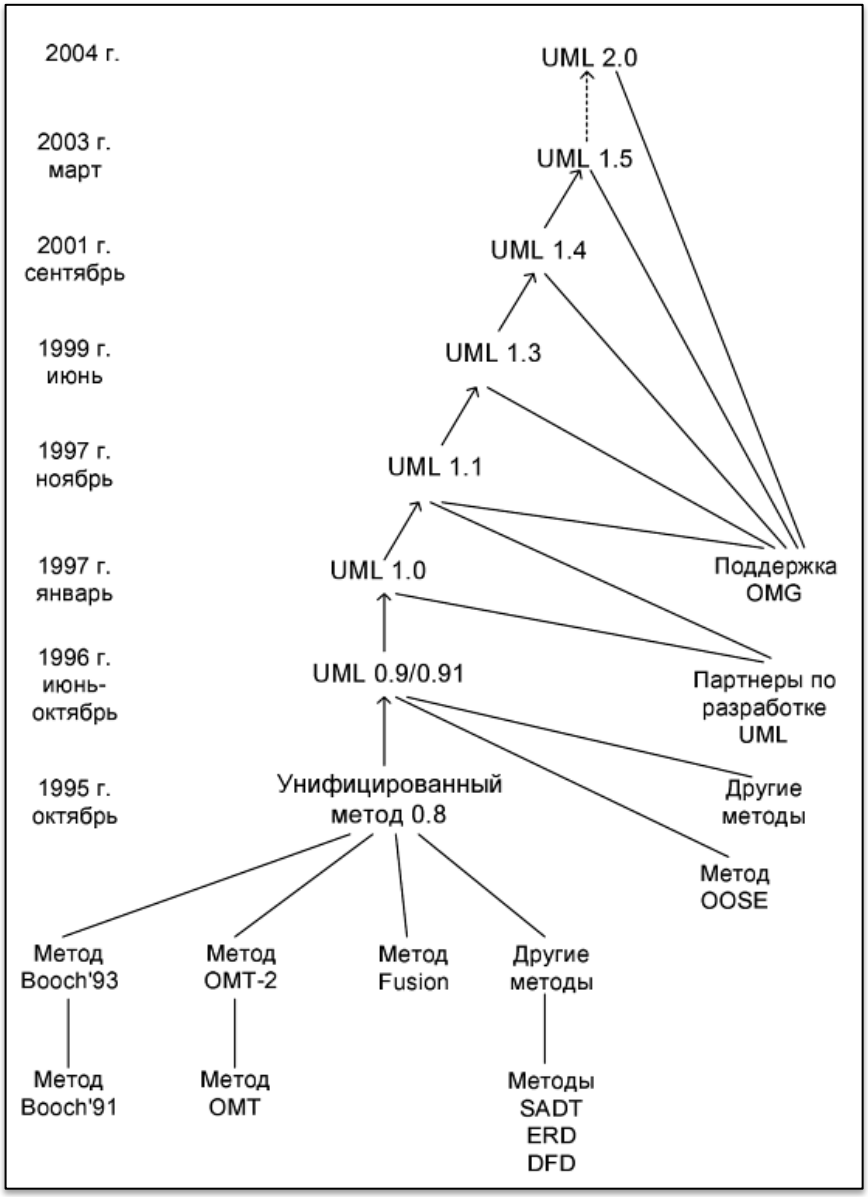
Лекция 3 (Князьков К.В.)

UML

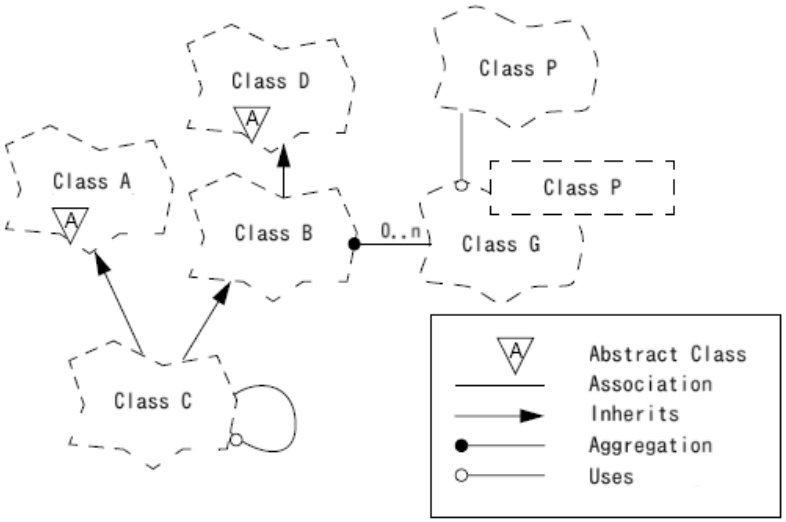
- Язык графического описания для объектного моделирования в области разработки программного обеспечения
- UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем
- Открытый стандарт, поддерживаемый консорциумом OMG (Object Management Group)
- Текущая версия: UML 2.4.1

История стандарта UML

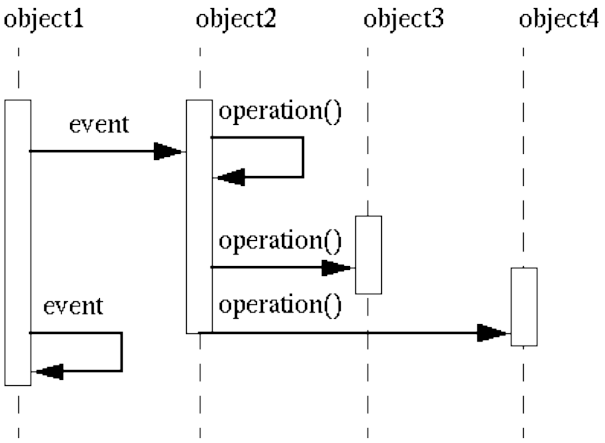
- Между 1989 и 1994 существовало более 50 различных методов объектно-ориентированного моделирования
- В 1994 Гради Буч и Джеймс Рамбо, работая вместе в Rational Software (в 2003 поглощена IBM), сделали первый шаг к унификации стандартов и объединили «**Метод Буча**» с «**Object Modeling Technique**» Рамбо, получив Unified Method 0.8.
- Осенью 1995 Айвар Якобсон присоединился к работе над унифицированным стандартом с методом «**Object-Oriented Software Engineering**»
- Промышленным тандартом UML стал под эгидой Object Management Group (OMG) в 1997 году.



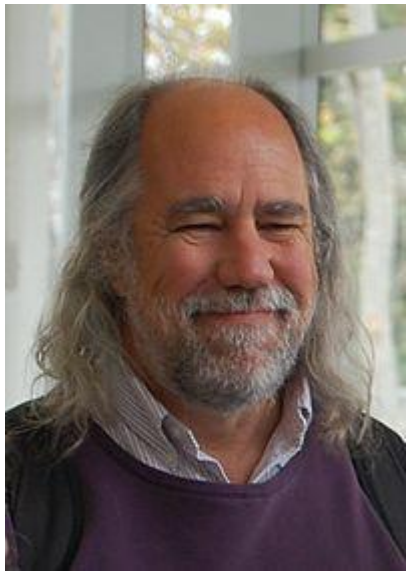
История UML → BOOCH



Class diagram



Interaction diagram

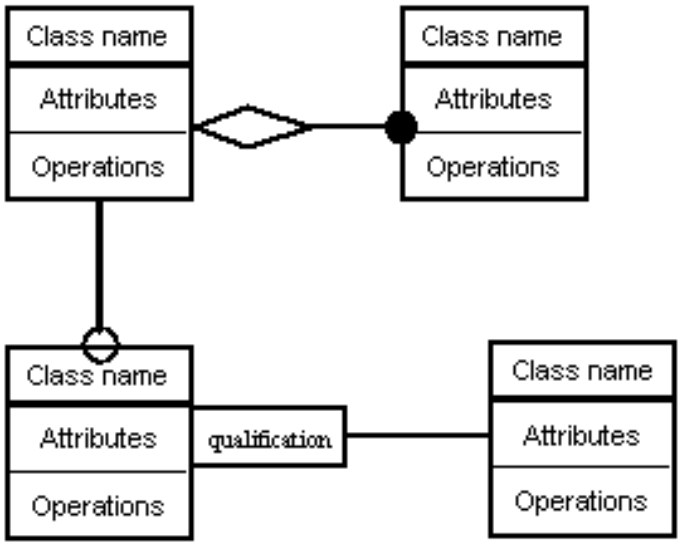


Гради Буч

Источники += Книга **«Объектно-ориентированный анализ и проектирование с примерами приложений»**

В открытом доступе выложена здесь: http://www.helloworld.ru/texts/comp/other/oop/index.htm

История UML → OMT

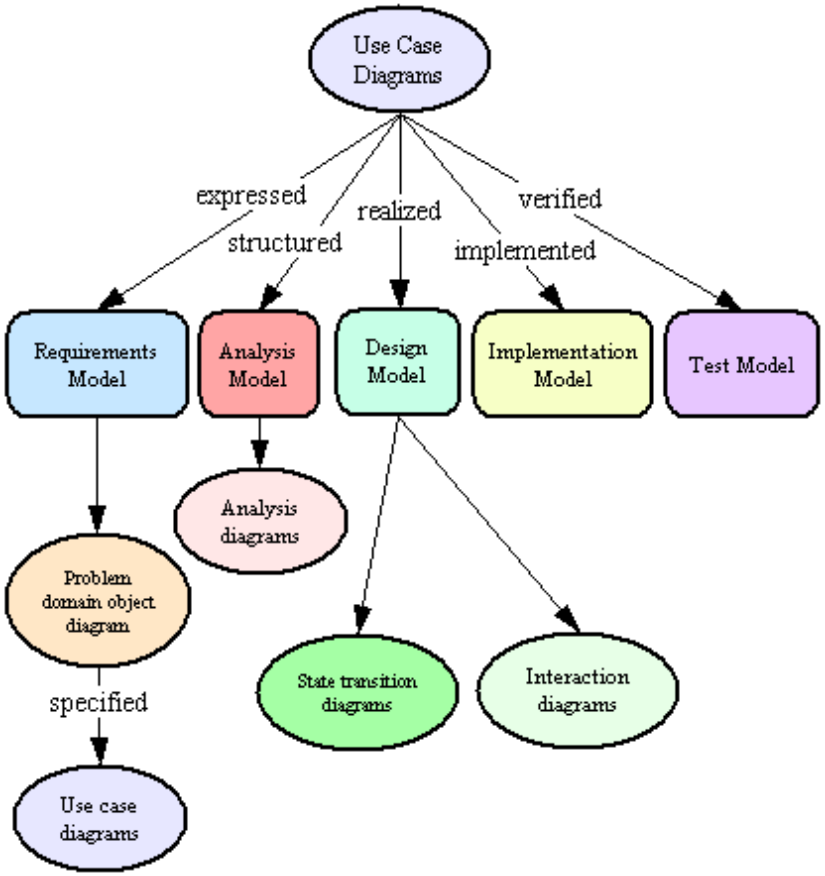


OMT Object Diagram

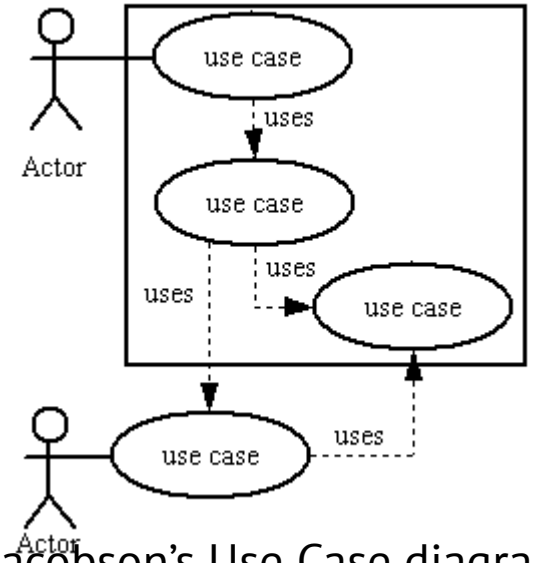
James Rumbaugh

Object-modeling technique (OMT) – подход к объектному моделированию в области моделирования ПО. Позволяет описать статическую объектную модель системы. Подход разработан в 1991.

История UML → OOSE



Object-Oriented Software Engineering

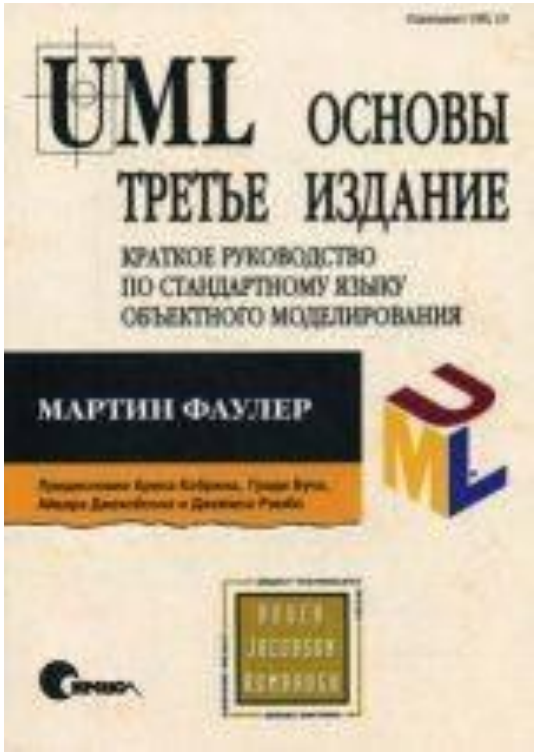


Jacobson's Use Case diagram

Ivar Jacobson

OOSE – объектный язык моделирования и соответствующая методология. Первая объектно-ориентированная методология, в которой использовались **прецеденты** (варианты использования). Разработана в 1992.

Мартин Фаулер (англ. Martin Fowler) — автор ряда книг и статей по архитектуре ПО, объектно-ориентированному анализу и разработке, языку UML, рефакторингу, экстремальному программированию, предметно-ориентированным языкам программирования.



MDA (Model Driven Architecture)

Разработка, управляемая моделями, (англ. **model-driven development**) — это стиль разработки программного обеспечения, когда модели становятся основными артефактами разработки, из которых генерируется код и другие артефакты

Model Driven Architecture (MDA) — создаваемая консорциумом OMG разновидность концепции "Разработка управляемая моделями": модельно-ориентированного подхода к разработке программного обеспечения. Его суть состоит в построении абстрактной метамодели управления и обмена метаданными (моделями) и задании способов ее трансформации в поддерживаемые технологии программирования (Java, CORBA, XML и др.).

Основные шаги разработки:

1. Сначала разрабатывается модель предметной области проектируемого приложения, полностью независимая от имплементирующей технологии. Она называется **Platform Independent Model** (PIM).
2. Затем PIM автоматически трансформируется специальным инструментом в платформо-зависимую модель (**Platform Specifical Model**, PSM).
3. PSM переводится в исходный код на соответствующем языке программирования.

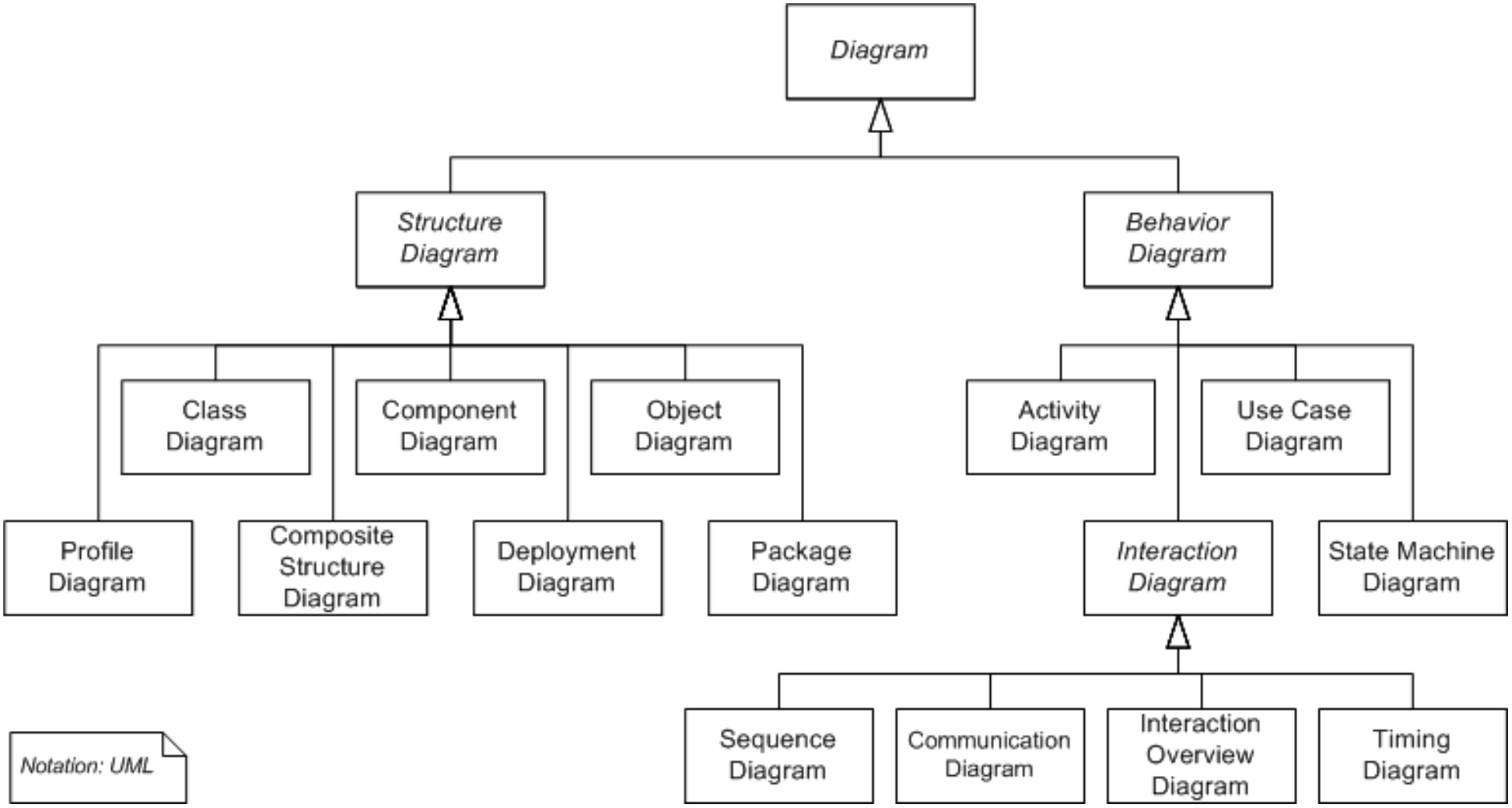
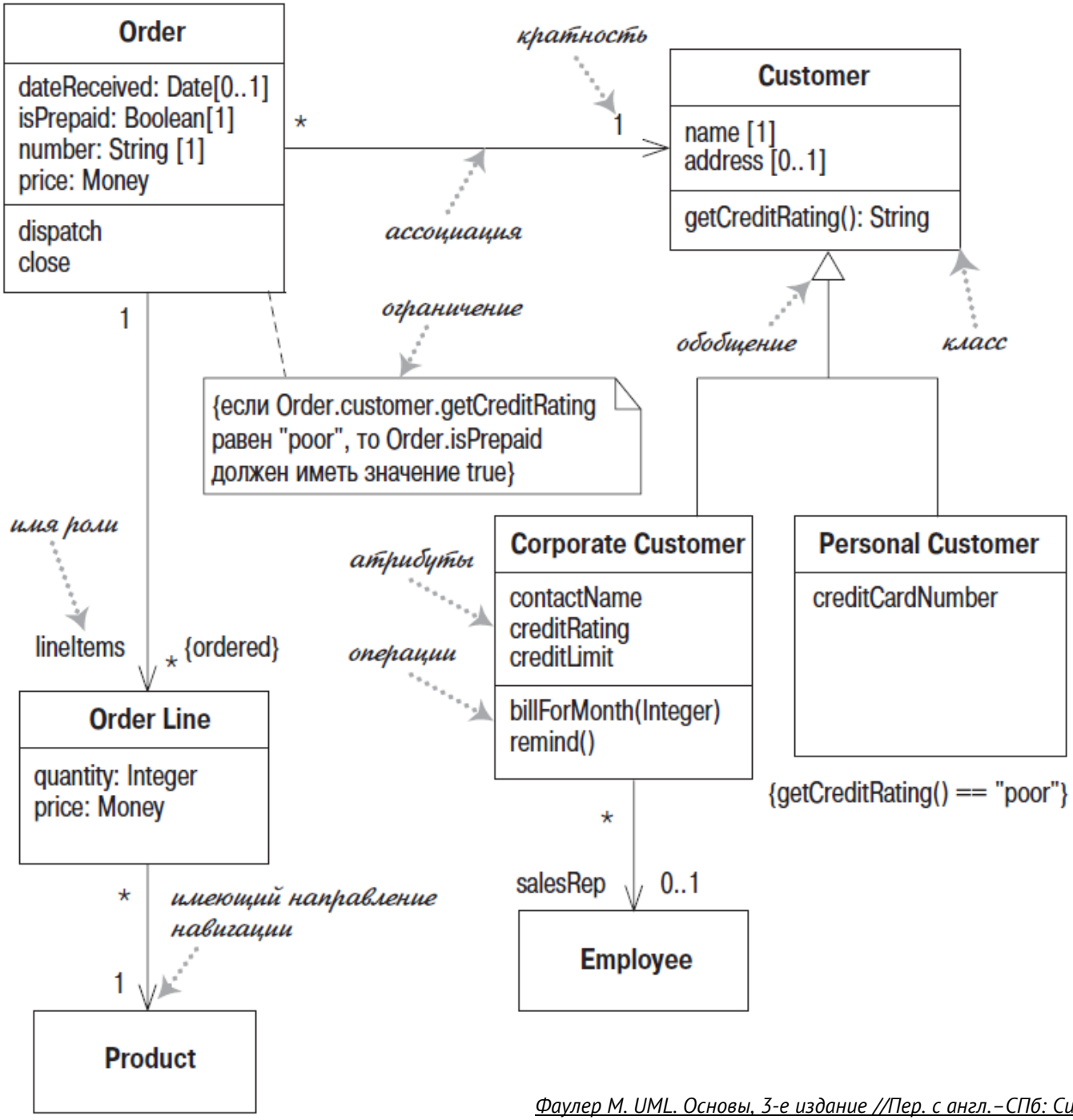


Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Component Diagram
Object Diagram
Activity Diagram
Use Case Diagram
Profile Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Interaction Diagram
State Machine Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram
Notation: UML

Диаграммы классов (class diagrams)

Точки зрения на построение диаграмм классов в зависимости от целей их применения:

- Концептуальная точка зрения — диаграмма классов описывает модель предметной области, в ней присутствуют только классы прикладных объектов;
- Точка зрения спецификации — диаграмма классов применяется при проектировании информационных систем;
- Точка зрения реализации — диаграмма классов содержит классы, используемые непосредственно в программном коде (при использовании объектно-ориентированных языков программирования).



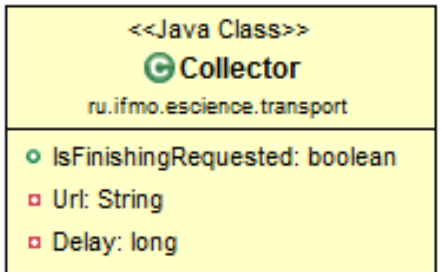
Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

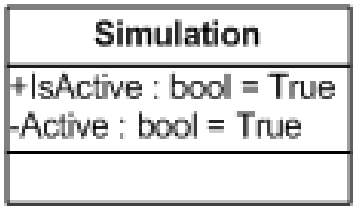
СВОЙСТВО = АТРИБУТ | АССОЦИАЦИЯ

Атрибуты

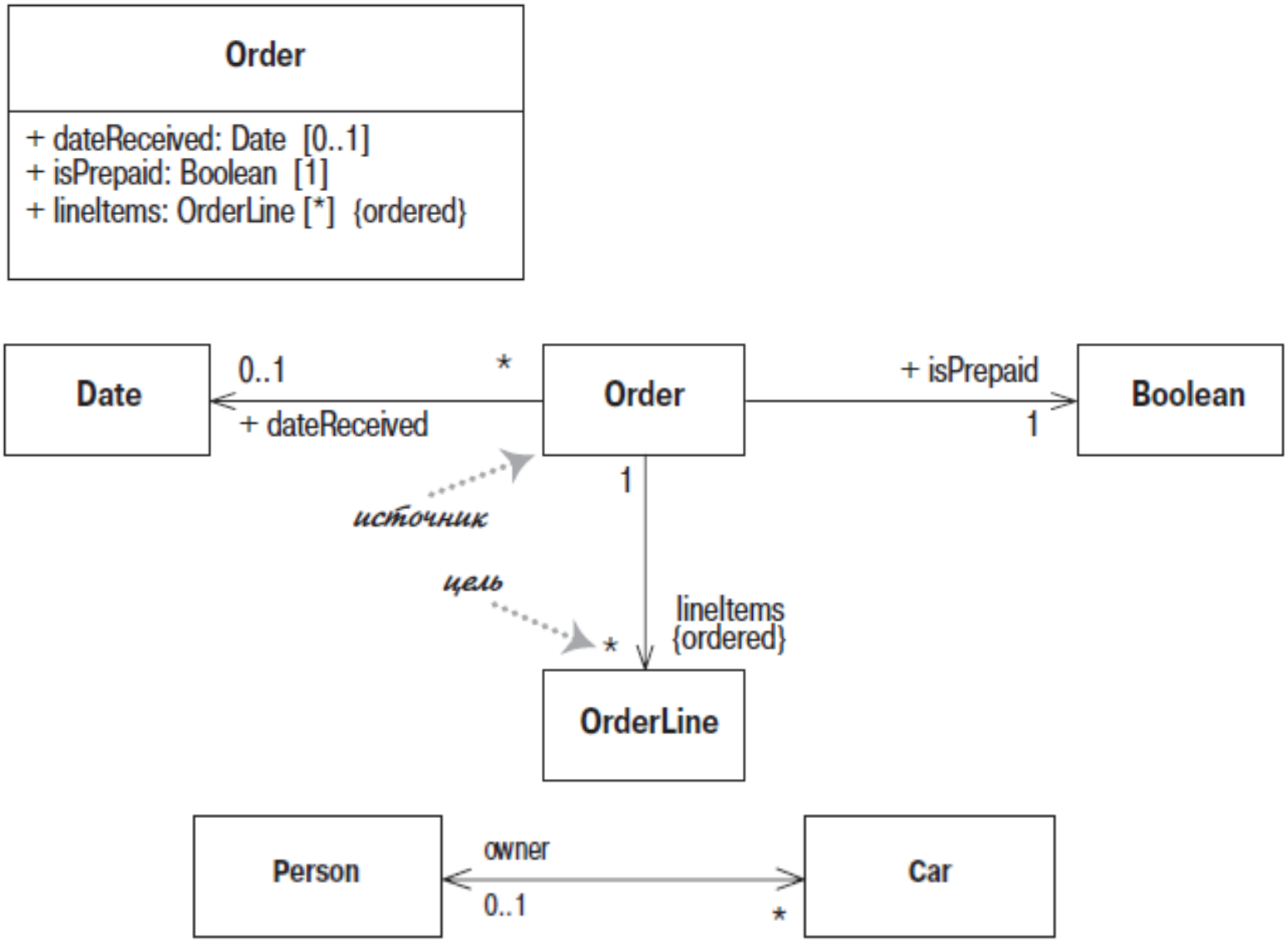
<видимость> <имя>: <тип> <кратность> = <значение по умолчанию> {<строка свойств>}

- IsActive: Bool [1] = True {readOnly}



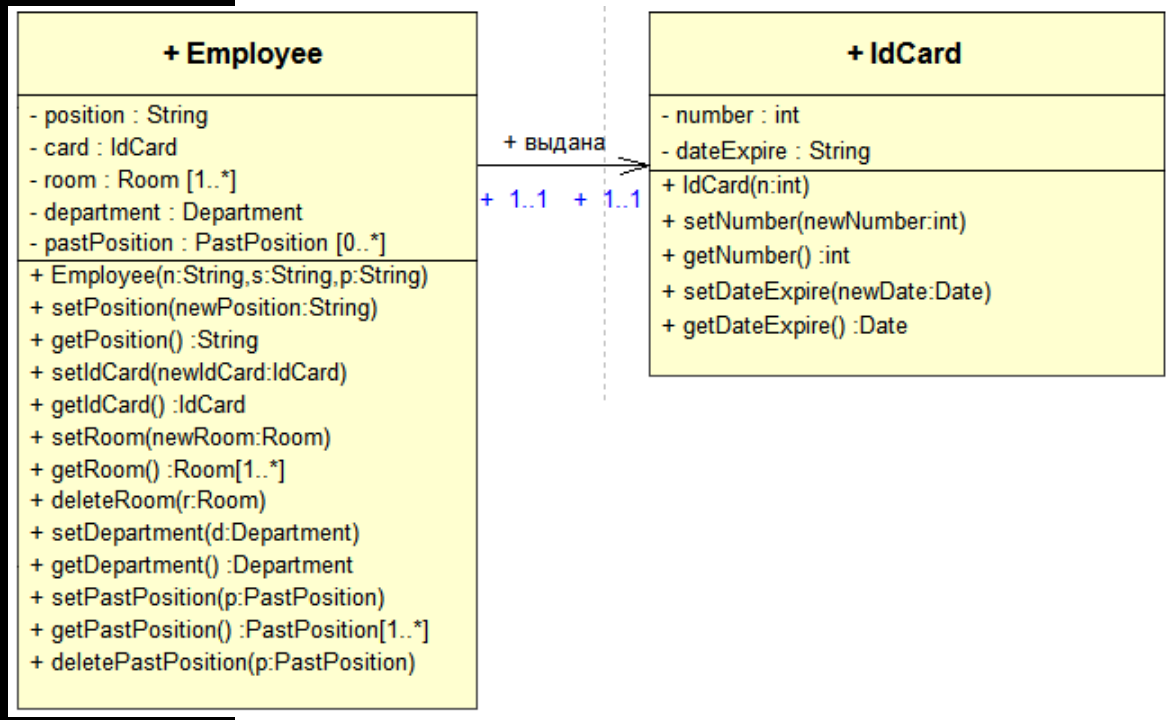


Ассоциации



Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

```java
public class Employee extends Man{
    private String position;
    private IdCard iCard;
    public Employee(String n, String s, String p){
        name = n;
        surname = s;
        position = p;
    }
    public void setPosition(String newPosition){
        position = newPosition;
    }
    public String getPosition(){
        return position;
    }
    public void setIdCard(IdCard c){
        iCard = c;
    }
    public IdCard getIdCard(){
        return iCard;
    }
}
public class IdCard{
    private Date dateExpire;
    private int number;
    public IdCard(int n){
        number = n;
    }
    public void setNumber(int newNumber){
        number = newNumber;
    }
    public int getNumber(){
        return number;
    }
    public void setDateExpire(Date newDateExpire){
        dateExpire = newDateExpire;
    }
    public Date getDateExpire(){
        return dateExpire;
    }
}
```



http://habrahabr.ru/post/150041/

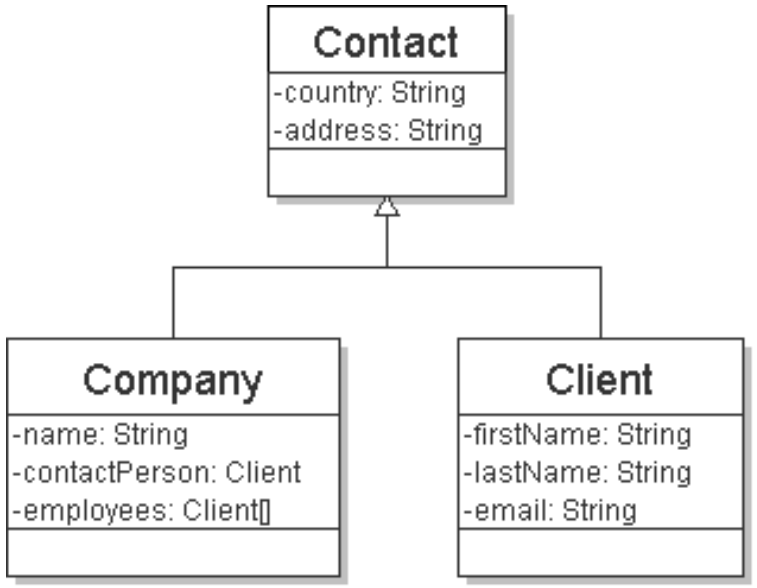
Операции

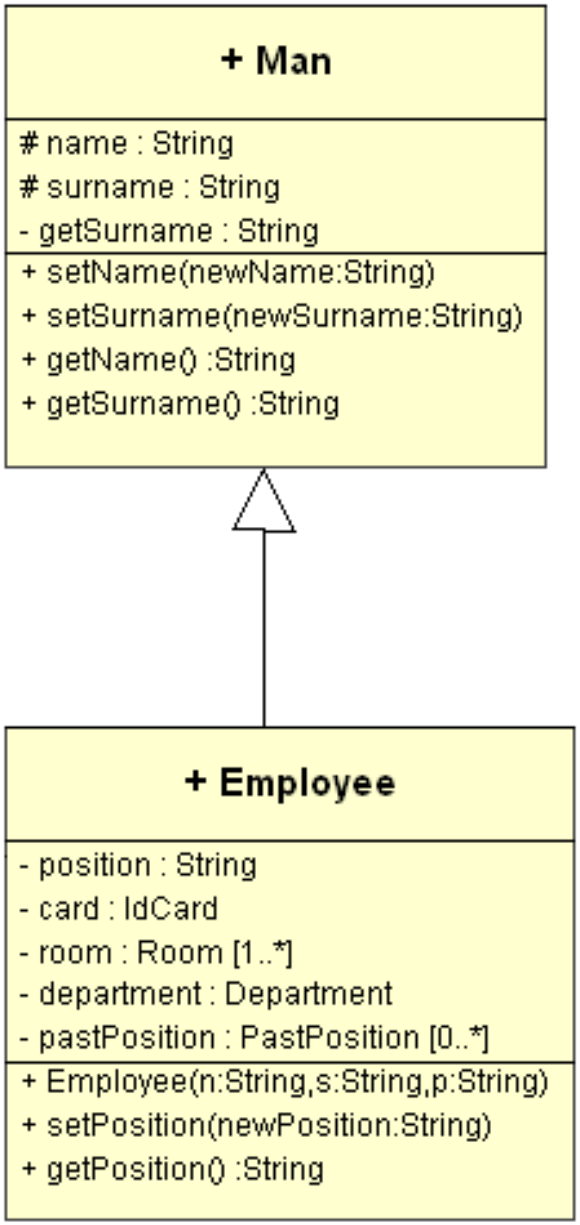
<видимость> <имя> (<список параметров>) : <возвращаемый тип> {<строка свойств>}

+ GetBalanceOn(date:Date) : Money

Обобщение (Generalization)

Обобщение (Generalization) показывает, что один из двух связанных классов (*подтип*) является частной формой другого (*надтипа*), который называется **обобщением** первого.

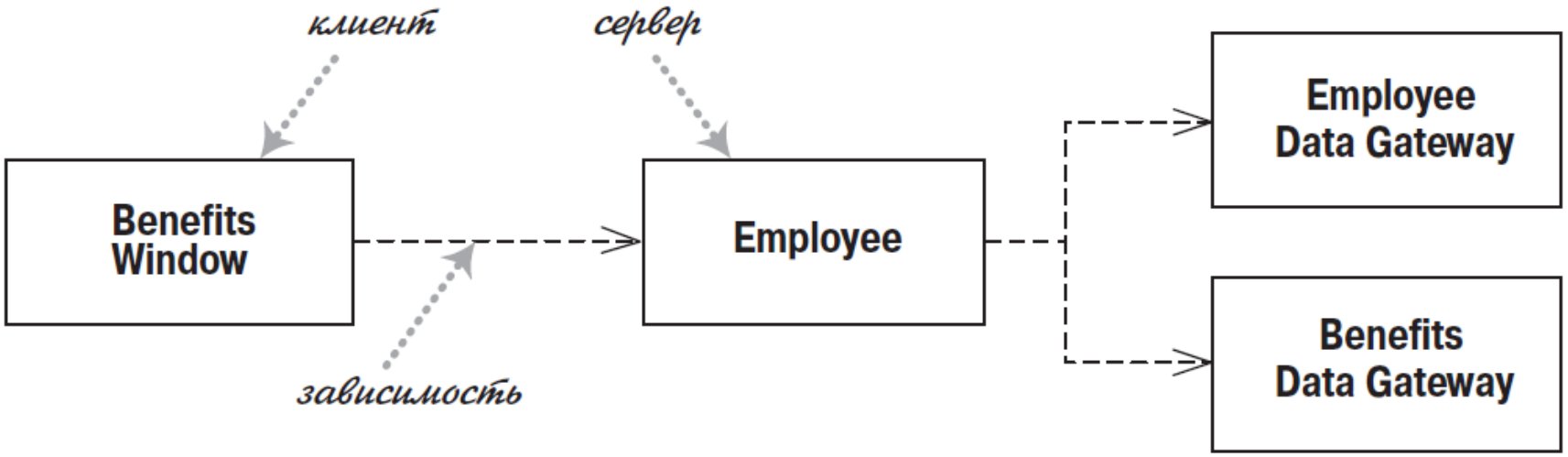





```java
public class Man{
protected String name;
    protected String surname;
    public void setName(String newName){
        name = newName;
    }
    public String getName(){
        return name;
    }
    public void setSurname(String newSurname){
        name = newSurname;
    }
    public String getSurname(){
        return surname;
    }
}
// наследуем класс Man
public class Employee extends Man{
    private String position;
    // создаем и конструктор
    public Employee(String n, String s, String p){
        name = n;
        surname = s;
        position = p;
    }
    public void setPosition(String newProfession){
        position = newProfession;
    }
    public String getPosition(){
        return position;
    }
}
```

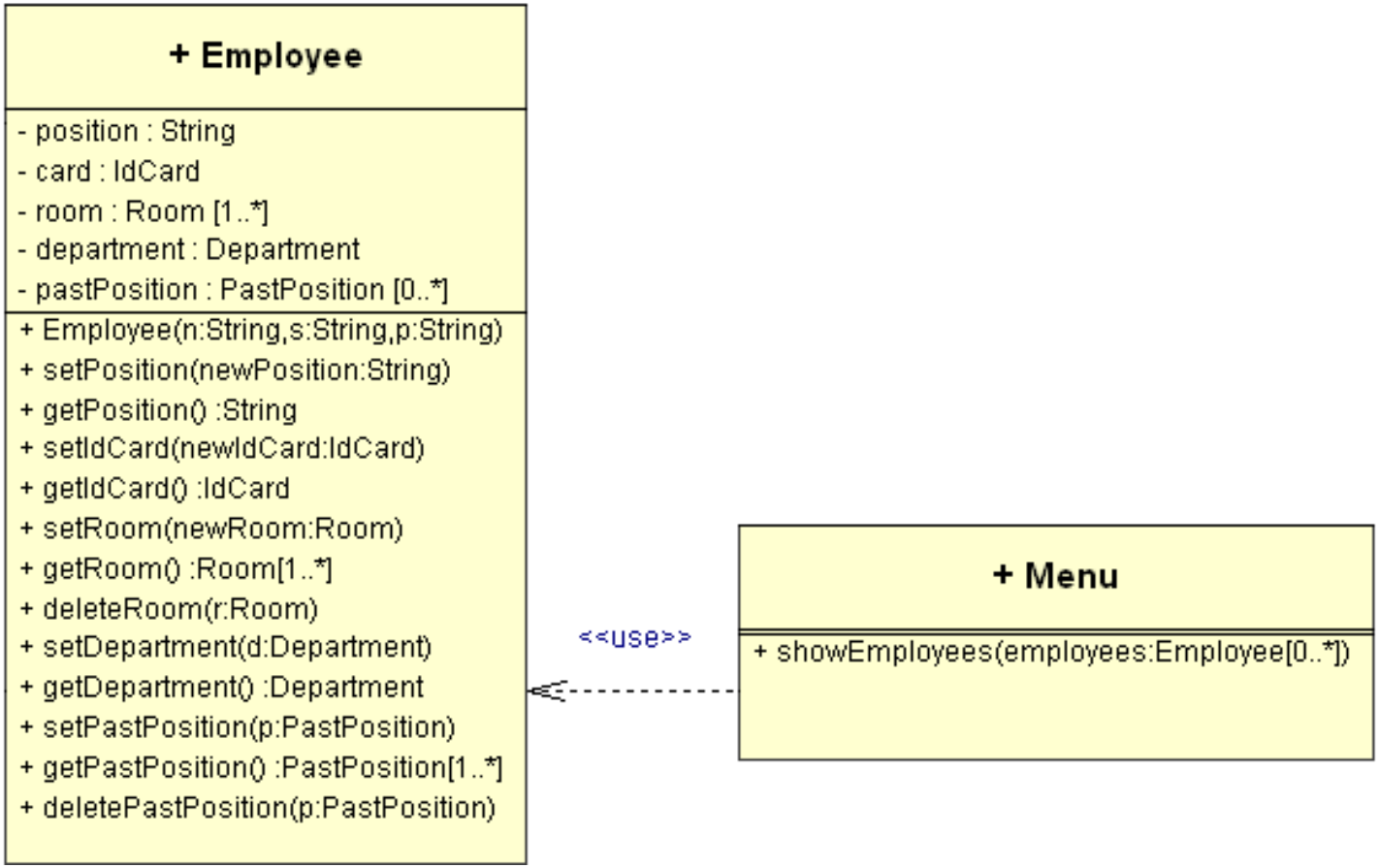
http://habrahabr.ru/post/150041/

Зависимость



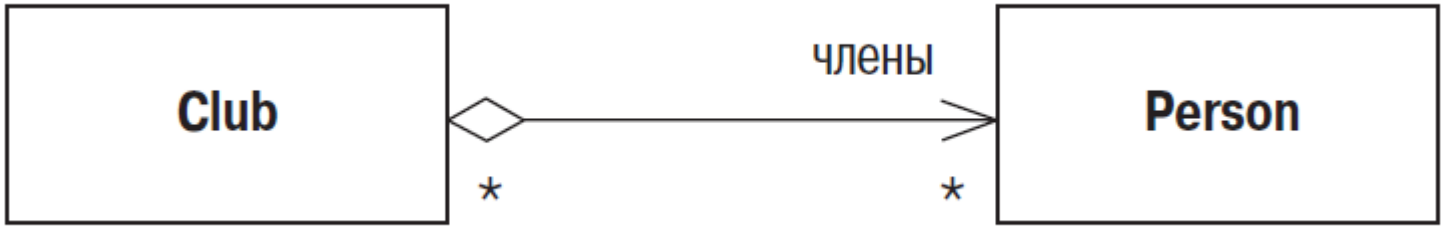
- call (вызывать)
- create (создавать)
- derive (производить)
- instantiate (создать экземпляр)
- permit (разрешать)
- realize (реализовывать)
- refine (уточнить)
- substitute (заместить)
- trace (проследить)
- use (использовать)

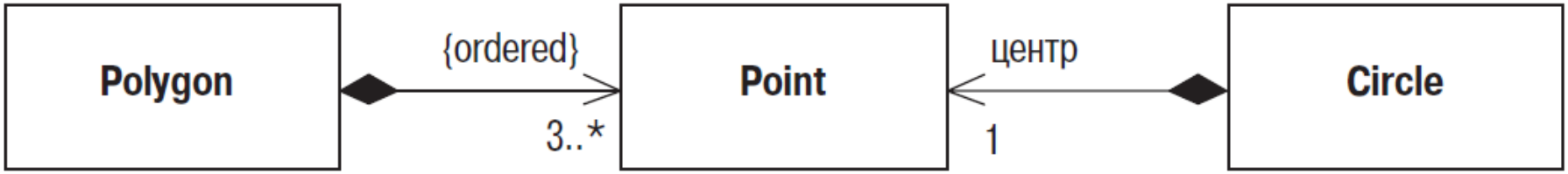
Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.



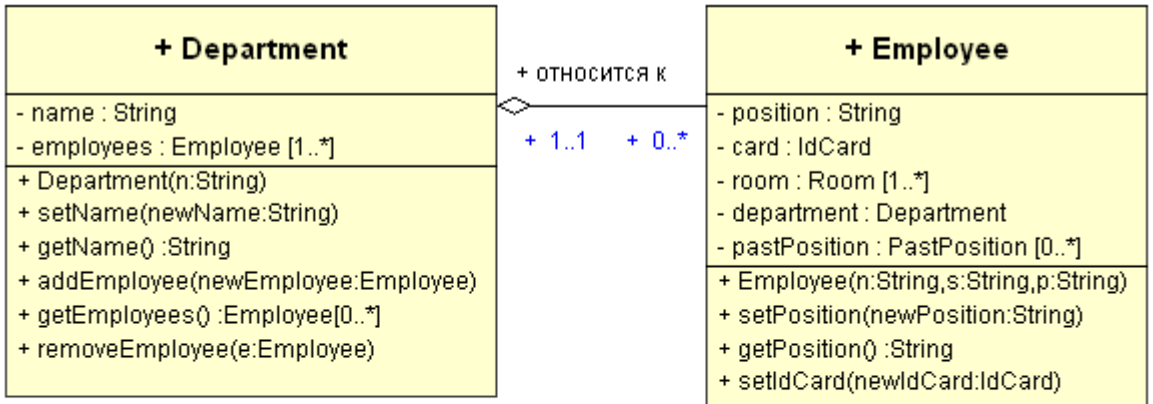
http://habrahabr.ru/post/150041/

Агрегация и композиция



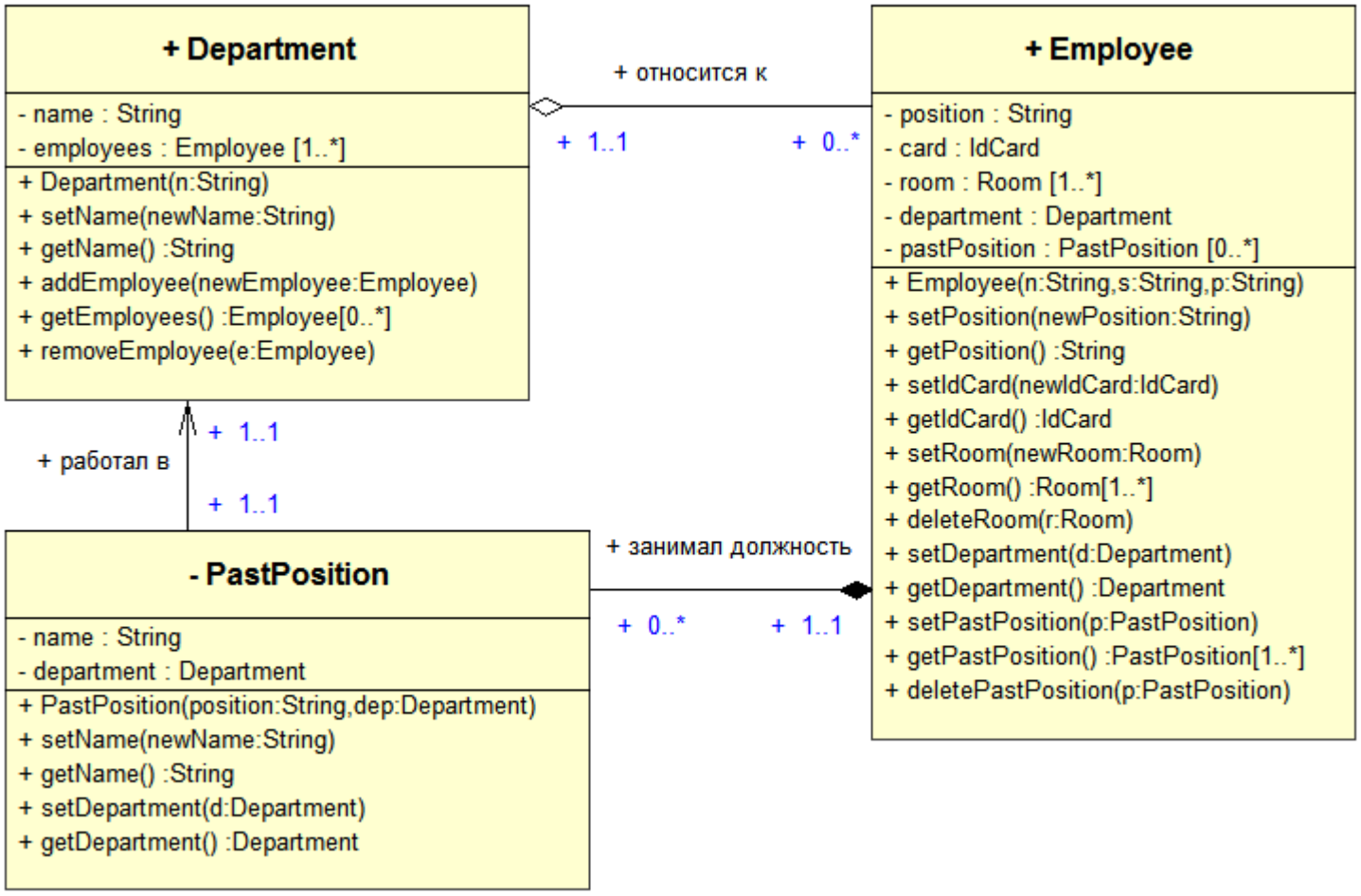


Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

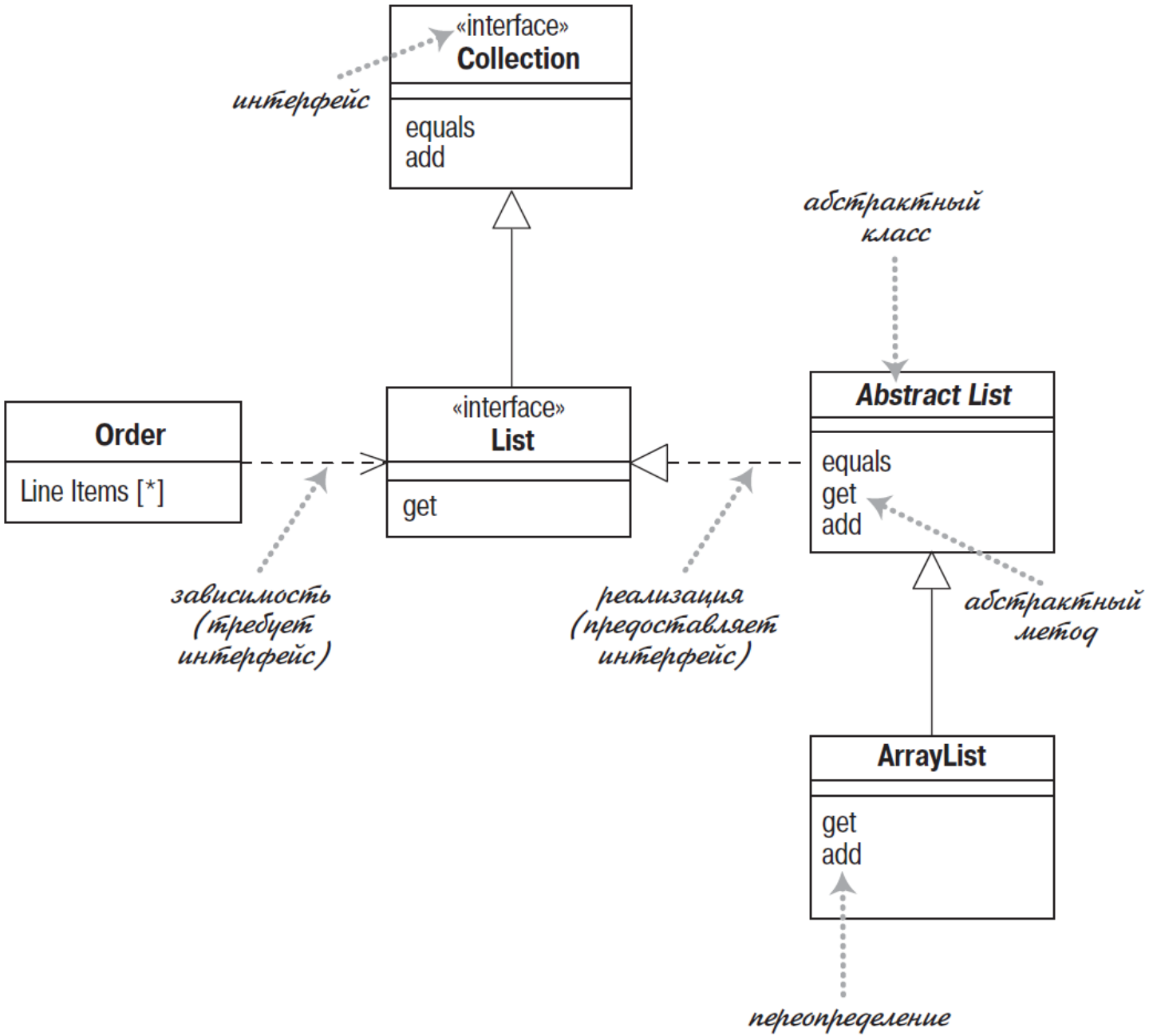



```
public class Department{
    private String name;
    private Set employees = new HashSet();
    public Department(String n){
        name = n;
    }
    public void setName(String newName){
        name = newName;
    }
    public String getName(){
        return name;
    }
    public void addEmployee(Employee newEmployee){
        employees.add(newEmployee);
        // связываем сотрудника с этим отделом
        newEmployee.setDepartment(this);
    }
    public Set getEmployees(){
        return employees;
    }
    public void removeEmployee(Employee e){
        employees.remove(e);
    }
}
```

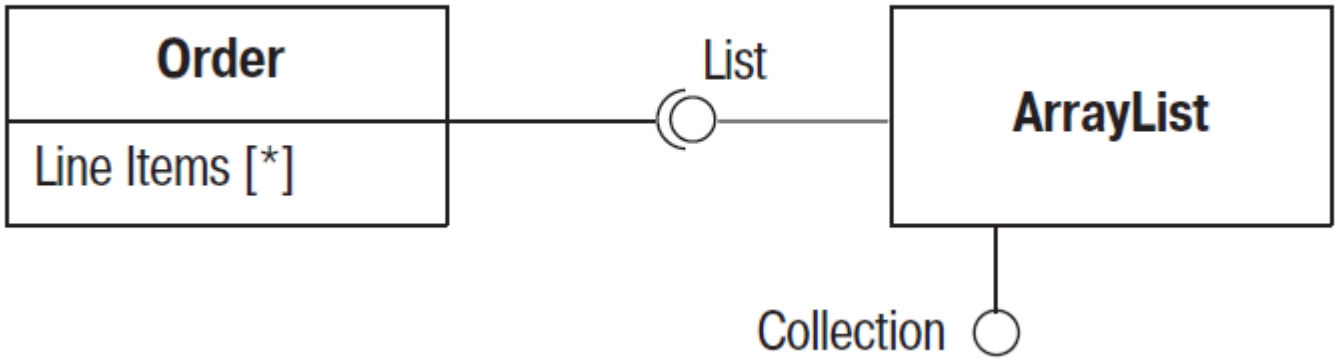
http://habrahabr.ru/post/150041/

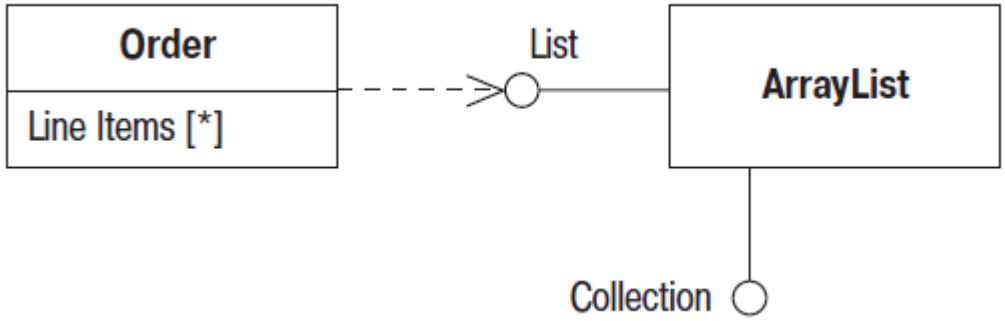


http://habrahabr.ru/post/150041/

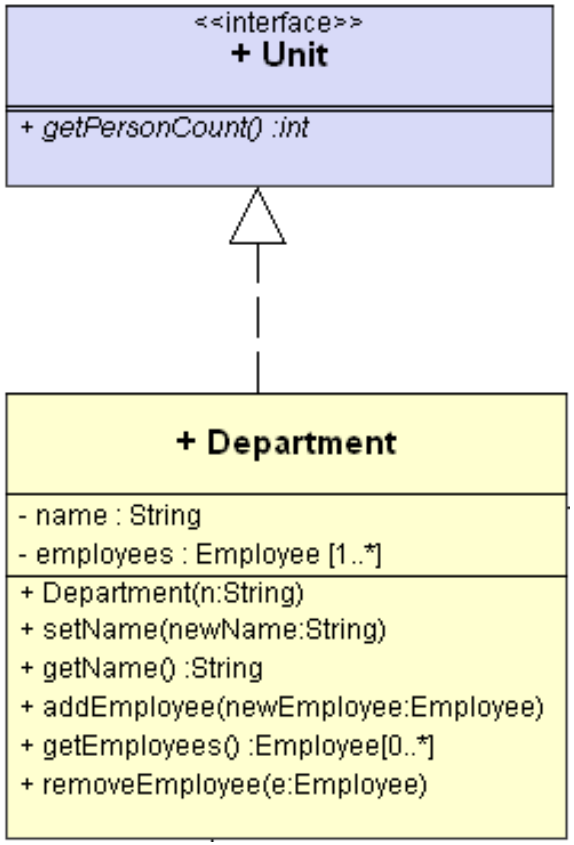


Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.





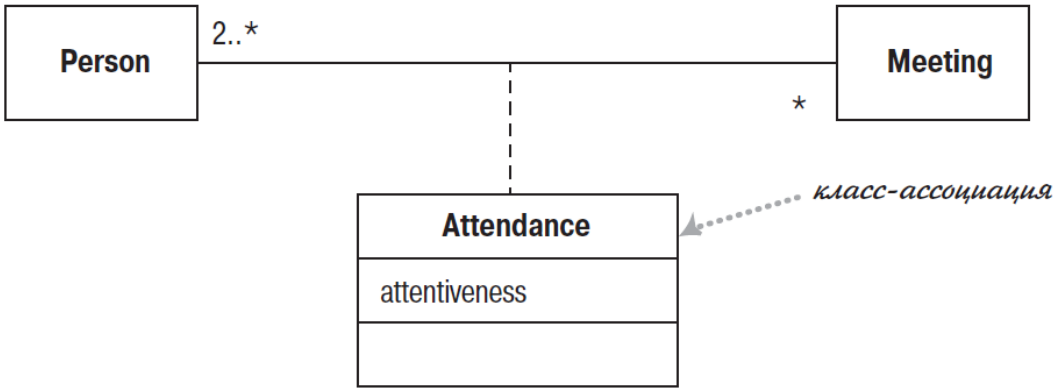
Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

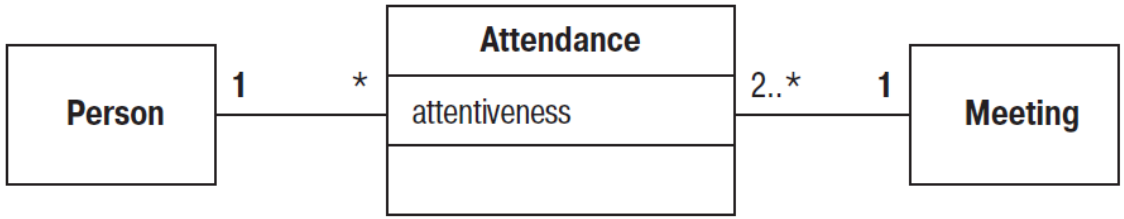



```
public class Department implements Unit{
    ...
    public int getPersonCount(){
        return getEmployees().size();
    }
```

http://habrahabr.ru/post/150041/

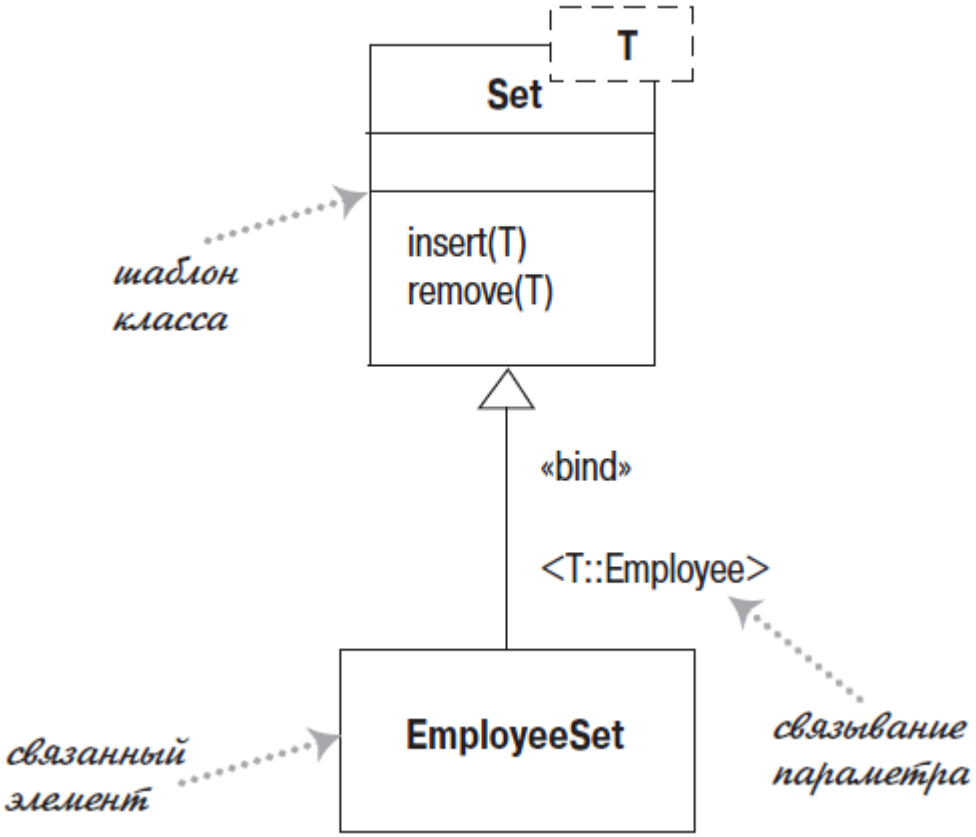
Класс-ассоциация





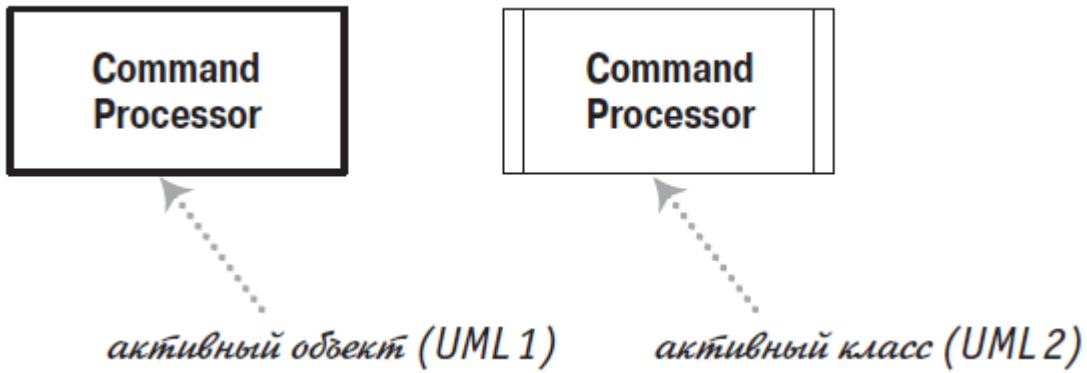
Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

Шаблоны классов

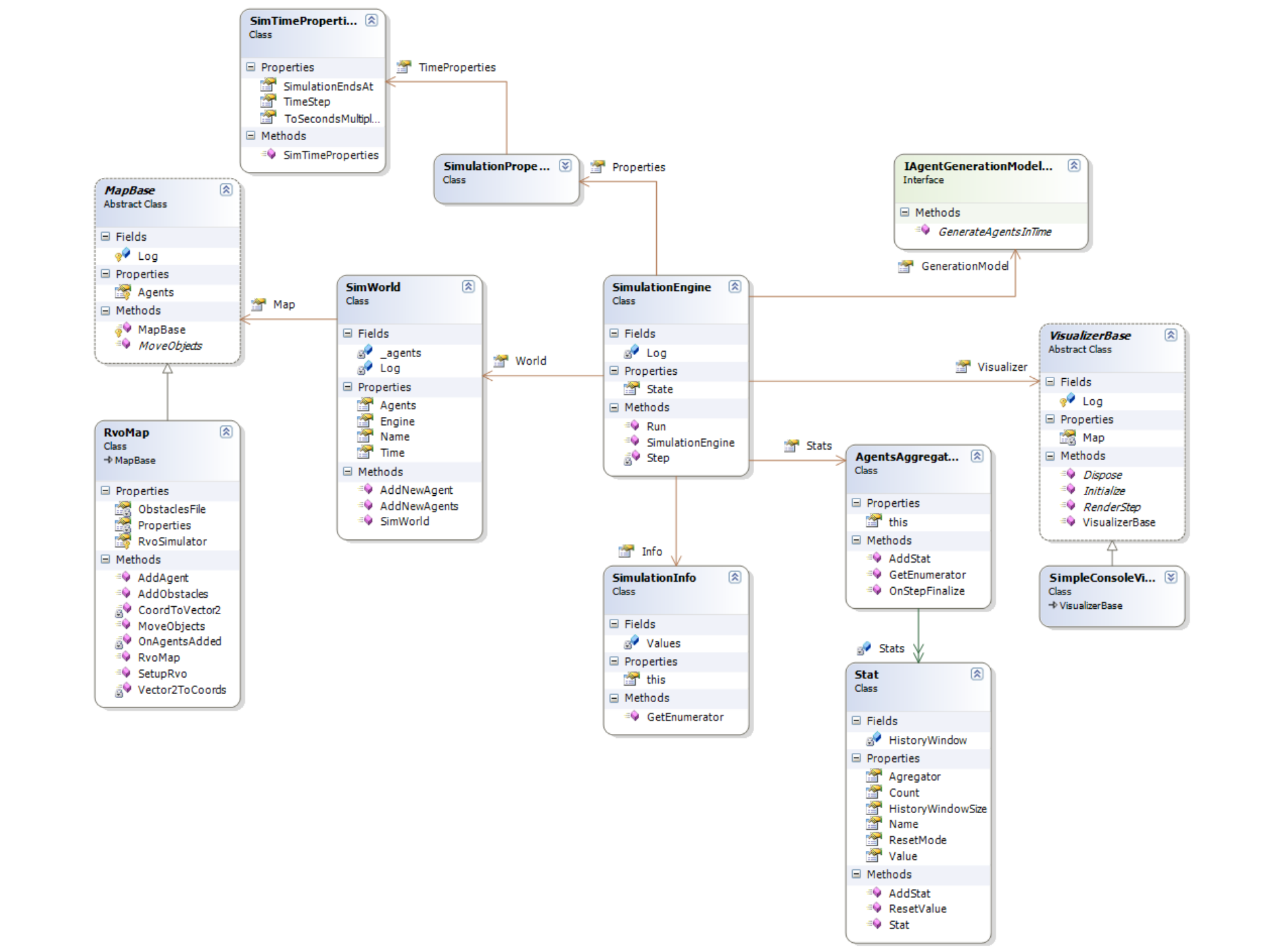


Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

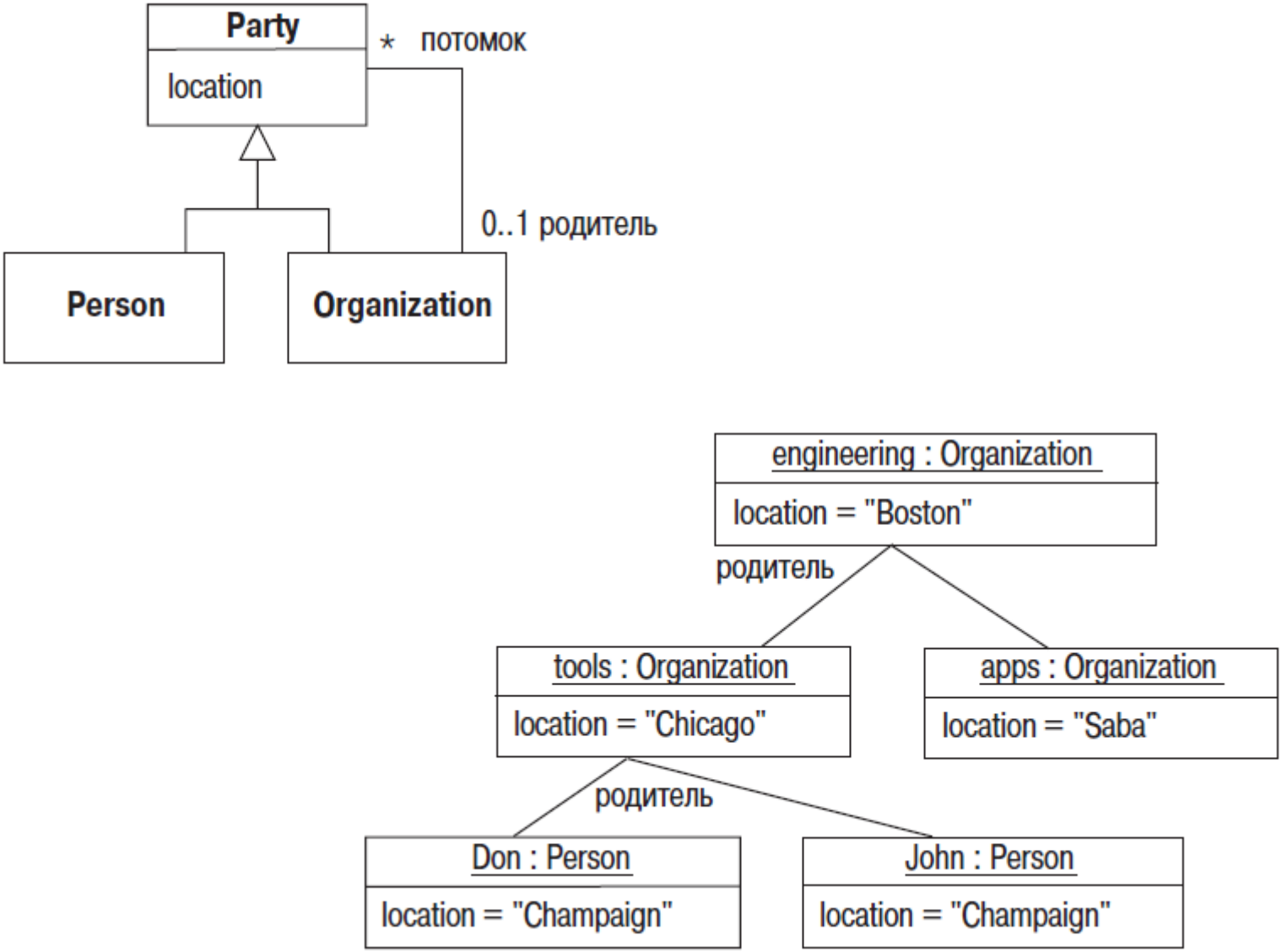
Активные классы



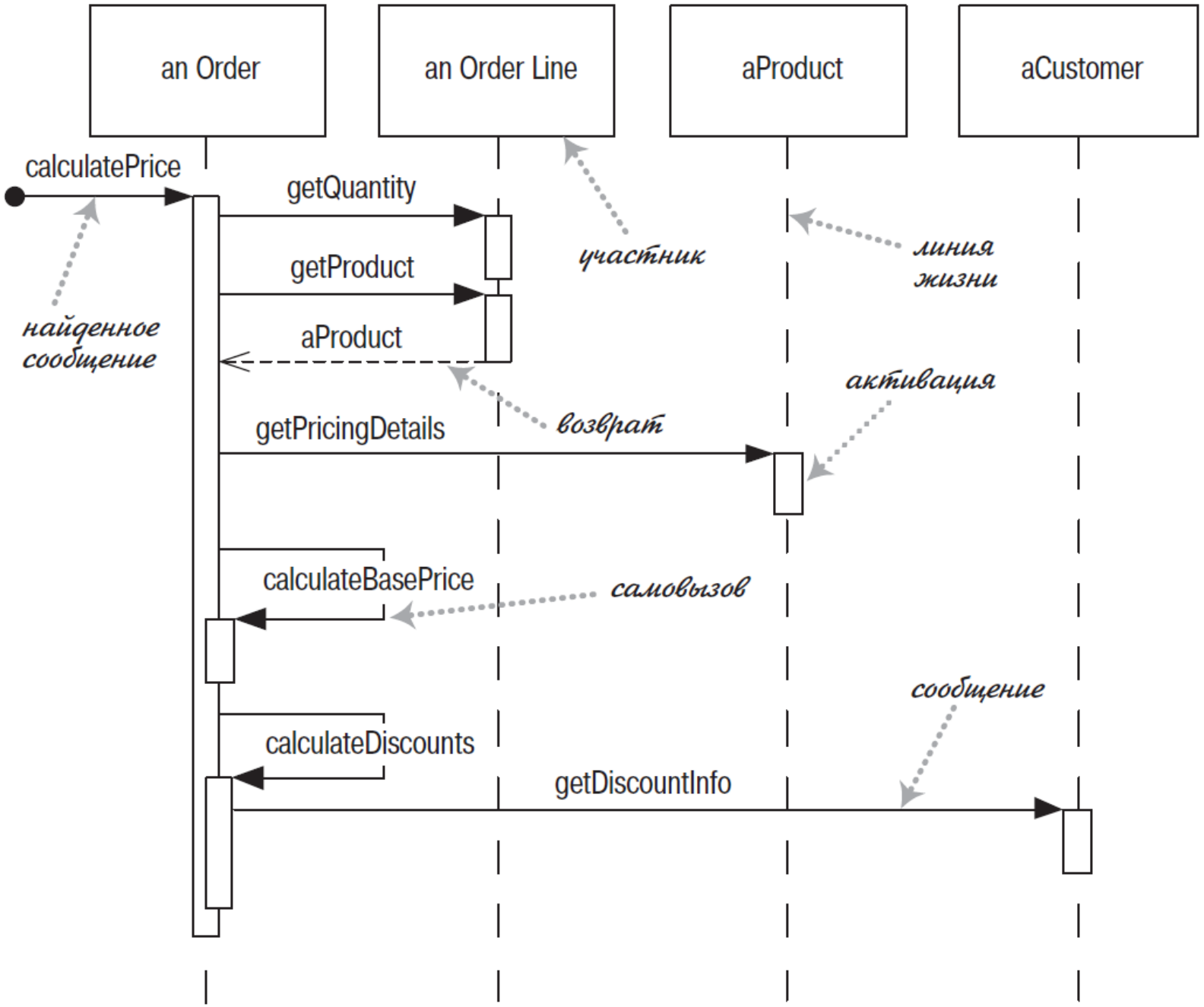
Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.


SimTimeProperti...
Class
Properties
SimulationEndsAt
TimeStep
ToSecondsMultipl...
Methods
SimTimeProperties
TimeProperties
SimulationPrope...
Class
Properties
IAgentGenerationModel...
Interface
Methods
GenerateAgentsInTime
GenerationModel
MapBase
Abstract Class
Fields
Log
Properties
Agents
Methods
MapBase
MoveObjects
Map
SimWorld
Class
Fields
_agents
Log
Properties
Agents
Engine
Name
Time
Methods
AddNewAgent
AddNewAgents
SimWorld
World
SimulationEngine
Class
Fields
Log
Properties
State
Methods
Run
SimulationEngine
Step
Visualizer
VisualizerBase
Abstract Class
Fields
Log
Properties
Map
Methods
Dispose
Initialize
RenderStep
VisualizerBase
RvoMap
Class
MapBase
Properties
ObstaclesFile
Properties
RvoSimulator
Methods
AddAgent
AddObstacles
CoordToVector2
MoveObjects
OnAgentsAdded
RvoMap
SetupRvo
Vector2ToCoords
Stats
AgentsAggregat...
Class
Properties
this
Methods
AddStat
GetEnumerator
OnStepFinalize
Info
SimulationInfo
Class
Fields
Values
Properties
this
Methods
GetEnumerator
SimpleConsoleVi...
Class
VisualizerBase
Stats
Stat
Class
Fields
HistoryWindow
Properties
Agregator
Count
HistoryWindowSize
Name
ResetMode
Value
Methods
AddStat
ResetValue
Stat

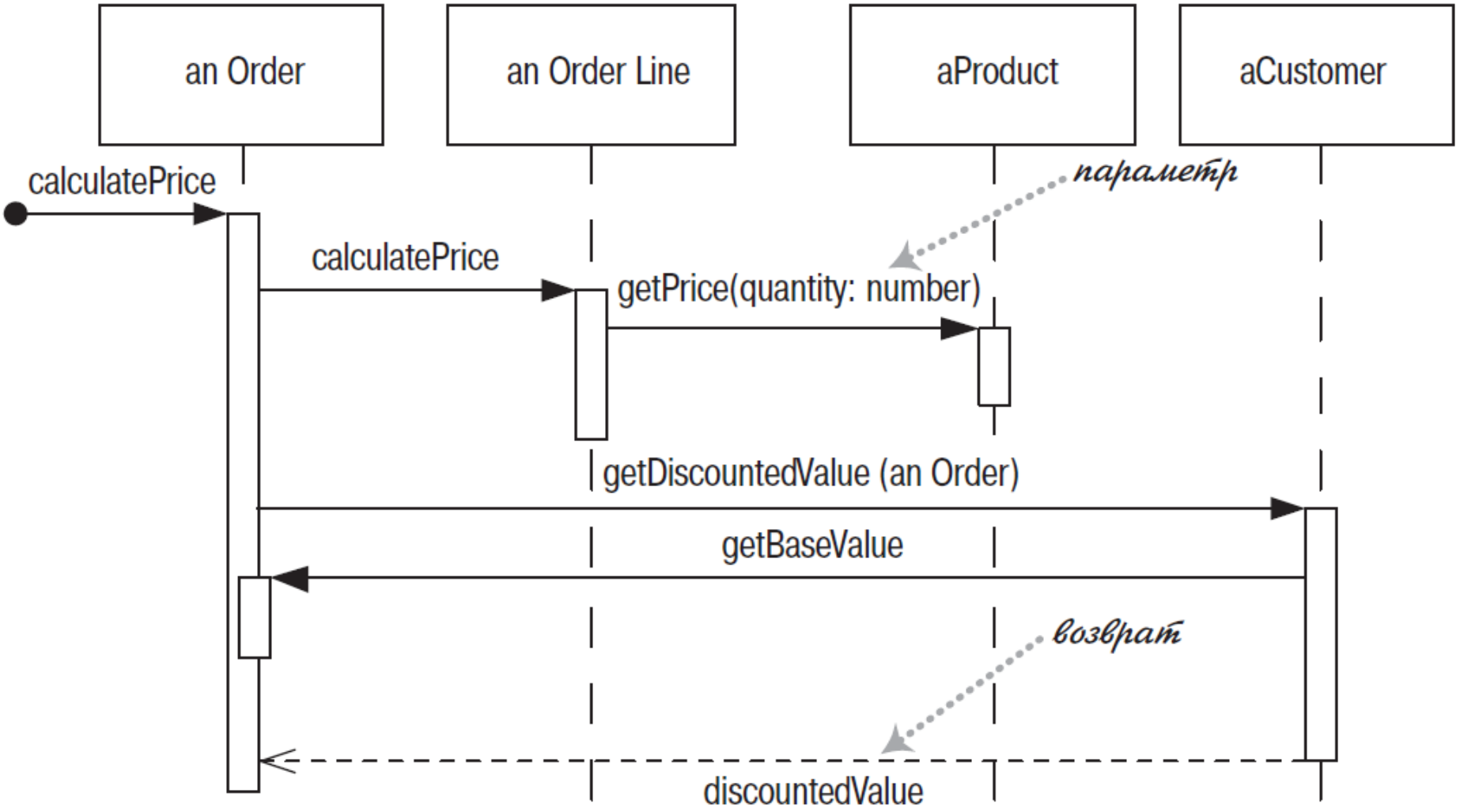
Диаграммы объектов
(object diagrams)


Party
location
* потомок
0..1 родитель
Person
Organization
engineering : Organization
location = "Boston"
родитель
tools : Organization
location = "Chicago"
apps : Organization
location = "Saba"
родитель
Don : Person
location = "Champaign"
John : Person
location = "Champaign"

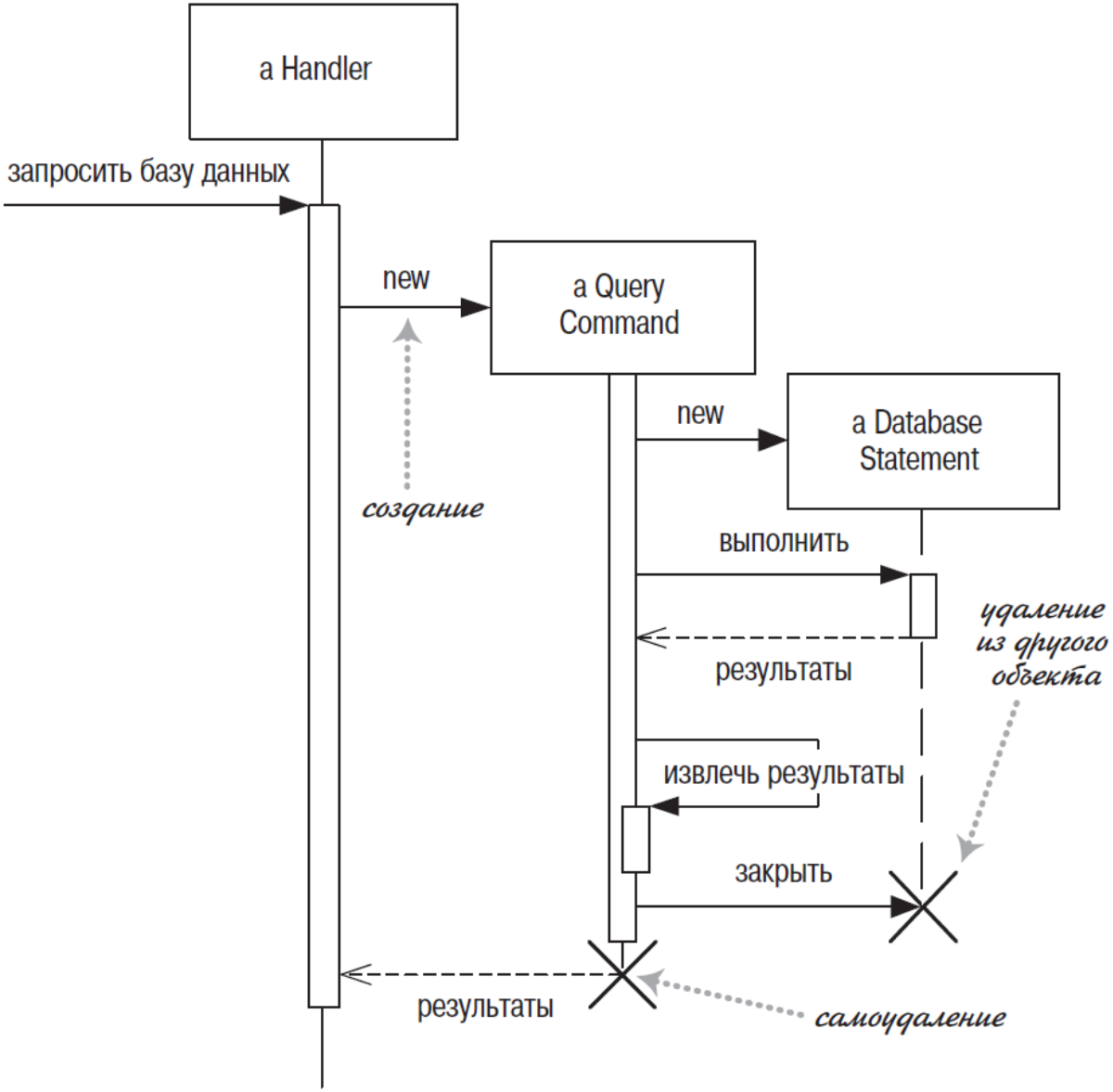
Диаграммы последовательности (Sequence diagrams)



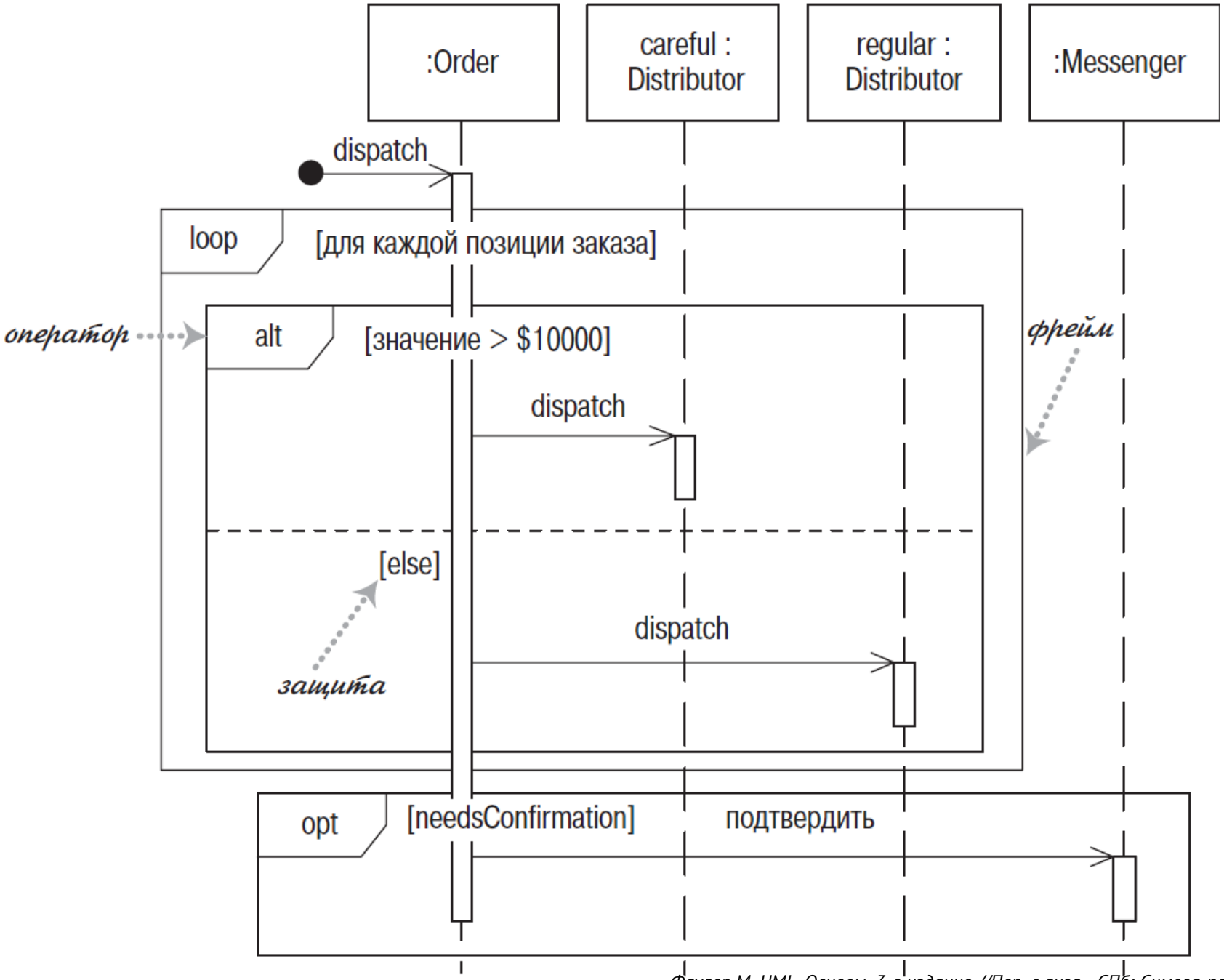
Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.



Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.



Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.



Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

Диаграммы прецедентов (use-case diagrams)

Понятие прецедента (варианта использования)

Сценарий использования, вариант использования, прецедент или же пользовательский сценарий (англ. Use Case) — в разработке программного обеспечения и системном проектировании это описание поведения системы, которым она отвечает на внешние запросы. Сценарий использования описывает, «кто» и «что» может сделать с рассматриваемой системой. Методика сценариев использования применяется для выявления требований к поведению системы, известных также как функциональные требования.

Вариант использования - описание множества возможных последовательностей действий (сценариев), приводящих к значимому для действующего лица результату.

Покупка товара

Целевой уровень: уровень моря

Главный успешный сценарий:

1. Покупатель просматривает каталог и выбирает товары для покупки.
2. Покупатель оценивает стоимость всех товаров.
3. Покупатель вводит информацию, необходимую для доставки товара (адрес, доставка на следующий день или в течение трех дней).
4. Система предоставляет полную информацию о цене товара и его доставке.
5. Покупатель вводит информацию о кредитной карточке.
6. Система осуществляет авторизацию счета покупателя.
7. Система подтверждает оплату товаров немедленно.
8. Система посылает подтверждение оплаты товаров по адресу электронной почты покупателя.

Расширения:

3а. Клиент является постоянным покупателем.

.1: Система предоставляет информацию о текущей покупке и ее цене, а также информацию о счете.

.2: Покупатель может согласиться или изменить значения по умолчанию, затем возвращаемся к шагу 6 главного успешного сценария.

6а. Система не подтверждает авторизацию счета.

.1: Пользователь может повторить ввод информации о кредитной карте или закончить сеанс.

Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

A. ХАРАКТЕРНАЯ ИНФОРМАЦИЯ
- — Цель в контексте
- — Область действия
- — Уровень
- — Предусловия
- — Условие успешного завершения
- — Условие неудачного завершения
- — Первичный действующий субъект
- — Условие начала действия

B. ОСНОВНОЙ СЦЕНАРИЙ ПРИ УСПЕШНОМ ЗАВЕРШЕНИИ
C. РАСШИРЕНИЯ
D. ВАРИАНТЫ
E. СОПУТСТВУЮЩАЯ ИНФОРМАЦИЯ
- — Приоритет
- — Рабочая характеристика
- — Частота
- — Превосходящий прецедент использования
- — Подчиненный прецедент использования
- — Канал связи с первичным действующим субъектом
- — Вторичные действующие субъекты
- — Канал связи со вторичными действующими субъектами

F. РАСПИСАНИЕ
G. ОТКРЫТЫЕ ПРОБЛЕМЫ

Шаблон сценария использования системы по А. Кокбэрну

Эндрю Х. Томас Дэвид. Программист-прагматик. Путь от подмастерья к мастеру //Питер Пресс. – 2007.

ПРЕЦЕДЕНТ ИСПОЛЬЗОВАНИЯ № 5: ПРИОБРЕТЕНИЕ ТОВАРА

A. ХАРАКТЕРНАЯ ИНФОРМАЦИЯ

- **Цель в контексте:** Покупатель напрямую направляет коммерческий запрос в нашу фирму и ожидает отгрузки товаров и выставления счета за указанные товары.
- **Область действия:** Фирма
- **Уровень:** Итоговая информация
- **Предусловия:** Нам известен покупатель, его адрес, и т.д.
- **Условие успешного завершения:** Покупатель получает товары, мы получаем оплату.
- **Условие неуспешного завершения:** Мы не производим отгрузку товаров, покупатель не производит оплату.
- **Первичный действующий субъект:** Покупатель, любой агент (или компьютер), действующий от имени заказчика
- **Условие начала действия:** Получение запроса на приобретение товара.

B. ОСНОВНОЙ СЦЕНАРИЙ С УСПЕШНЫМ ЗАВЕРШЕНИЕМ

1. Покупатель обращается в фирму с запросом на приобретение товара.
2. Фирма фиксирует имя покупателя, его адрес, требуемые товары, и т.д.
3. Фирма предоставляет покупателю информацию о товарах, ценах, сроках поставки, и т.д.
4. Покупатель подтверждает заказ.
5. Фирма компонует заказ, отправляет заказ покупателю.
6. Фирма высылает покупателю счет-фактуру.
7. Покупатель оплачивает счет-фактуру.

C. РАСШИРЕНИЯ

3а. Один из пунктов заказа отсутствует у данной фирмы: Заказ переоформляется.
4а. Покупатель производит оплату непосредственно кредитной картой: Прием оплаты кредитной картой (прецедент использования № 44).
7а. Покупатель возвращает товар: Оформление возвращенного товара (прецедент использования №105).

D. ВАРИАНТЫ

1. Покупатель может осуществить заказ по телефону, факсу, при помощи Интернет-формы (на странице), по другим сетям электронного обмена информацией.
7. Покупатель может оплатить заказ наличными, денежным переводом, чеком, или кредитной картой.

Эндрю Х. Томас Дэвид. Программист-прагматик. Путь от подмастерья к мастеру //Питер Пресс. – 2007.

D. ВАРИАНТЫ

1. Покупатель может осуществить заказ по телефону, факсу, при помощи Интернет-формы (на странице), по другим сетям электронного обмена информацией.
7. Покупатель может оплатить заказ наличными, денежным переводом, чеком, или кредитной картой.

E. СОПУТСТВУЮЩАЯ ИНФОРМАЦИЯ

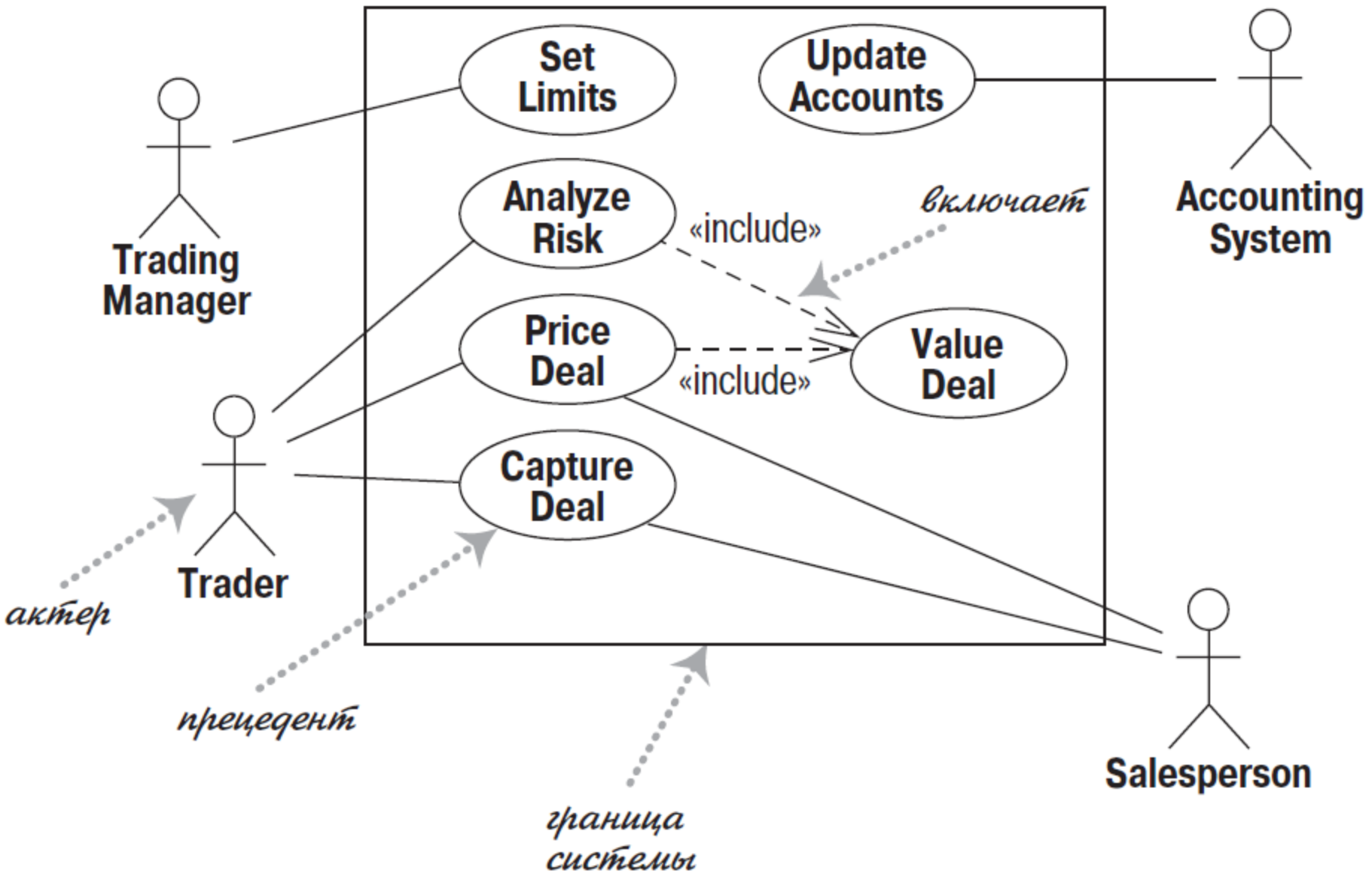
- **Приоритет:** Высший
- **Производительность:** 5 минут на оформление заказа, оплата в течение 45 дней
- **Частота:** 200 заказов в день
- **Превосходящий прецедент использования:** Управление взаимоотношением с заказчиком (прецедент использования № 2).
- **Подчиненные прецеденты использования:** Компоновка заказа (прецедент использования №15)
- Прием оплаты кредитной картой (прецедент использования № 44). Возврат товара покупателем (прецедент использования № 105).
- **Канал общения с первичным действующим субъектом:** по телефону, факсу или компьютерной сети.
- **Вторичные действующие субъекты:** компания — оператор платежной системы, банк, экспедиторская фирма.

F. РАСПИСАНИЕ

- **Должная дата:** Выпуск 1.0

G. ПРОБЛЕМЫ, ЯВЛЯЮЩИЕСЯ ОТКРЫТЫМИ

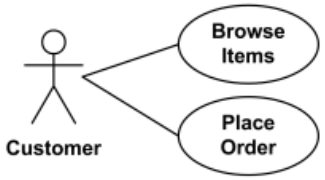
- Что происходит, если имеется лишь часть заказа?
- Что происходит, если кредитная карта похищена?

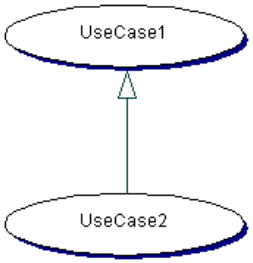


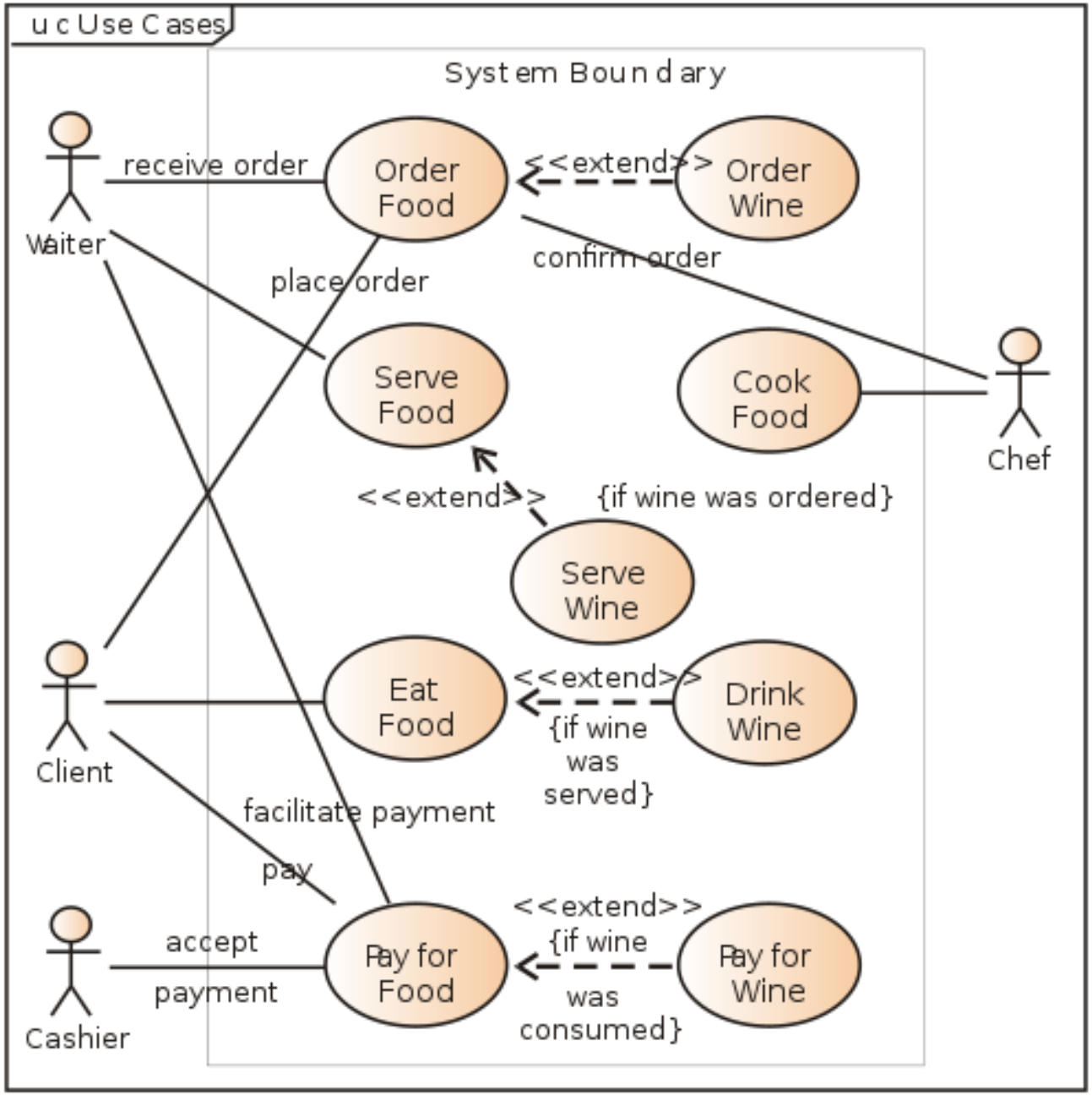
Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

Отношения

- **Ассоциация** означает, что действующее лицо тем или иным способом участвует в выполнении сценариев варианта использования (A – UC)
- **Обобщение** означает, что один объект наследует все свойства (в частности, участие в ассоциациях) другого объекта (A–A; UC– UC)
- **Отношение включения** показывает, что в каждый сценарий зависимого варианта использования в определенном месте вставляется в качестве подпоследовательности действий сценарий независимого варианта использования (UC – UC).
- **Отношение расширения** показывает, что некоторый сценарий независимого варианта использования может быть в определенном месте вставлен в качестве подпоследовательности действий в сценарий зависимого варианта использования (UC – UC).

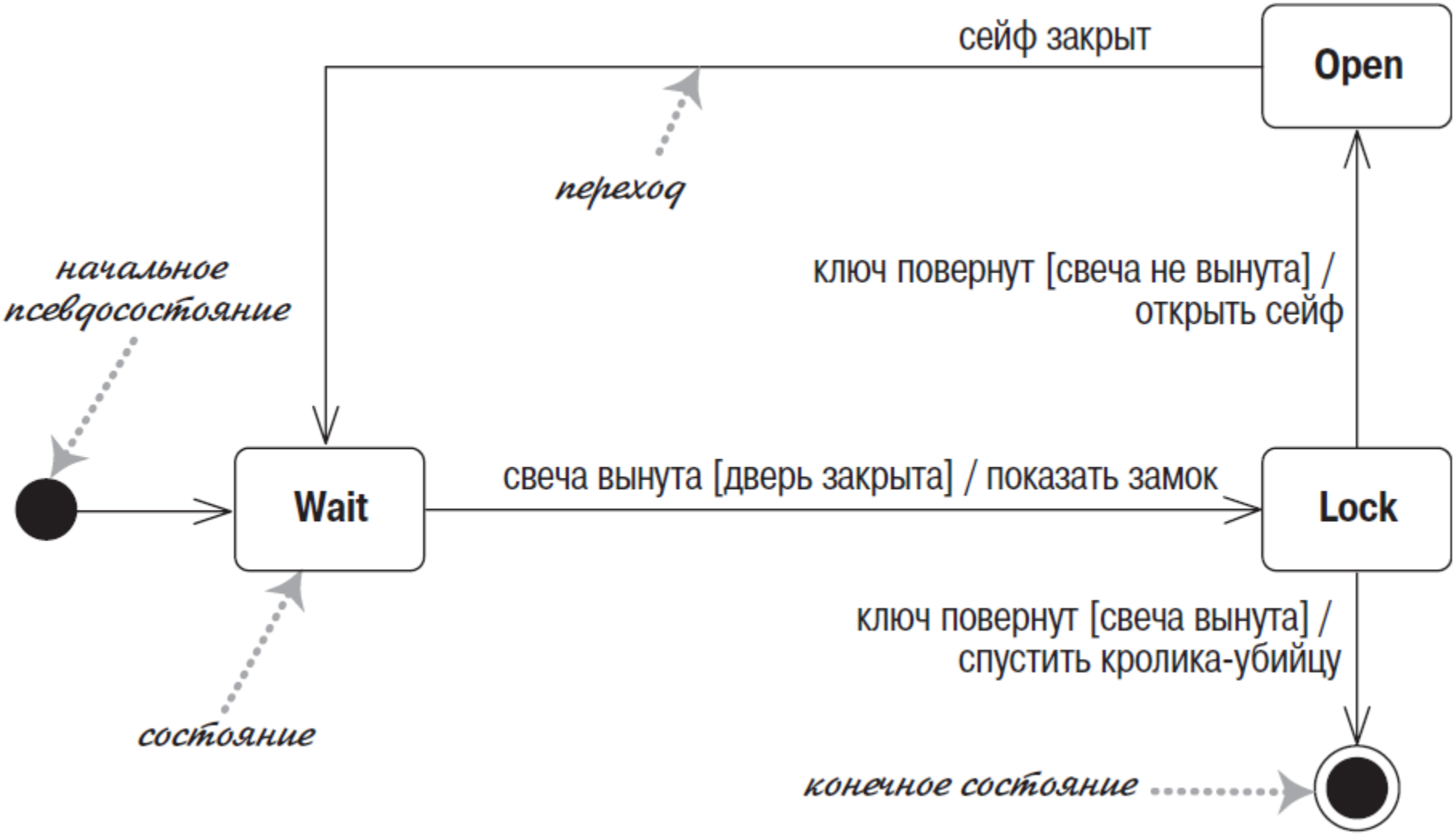





u c Use Cases
System Boundary
Waiter
receive order
Order Food
<<extend>>
Order Wine
confirm order
place order
Serve Food
Cook Food
Chef
<<extend>>
{if wine was ordered}
Serve Wine
Client
Eat Food
<<extend>>
Drink Wine
{if wine was served}
facilitate payment
pay
Cashier
accept payment
Pay for Food
<<extend>>
{if wine was consumed}
Pay for Wine

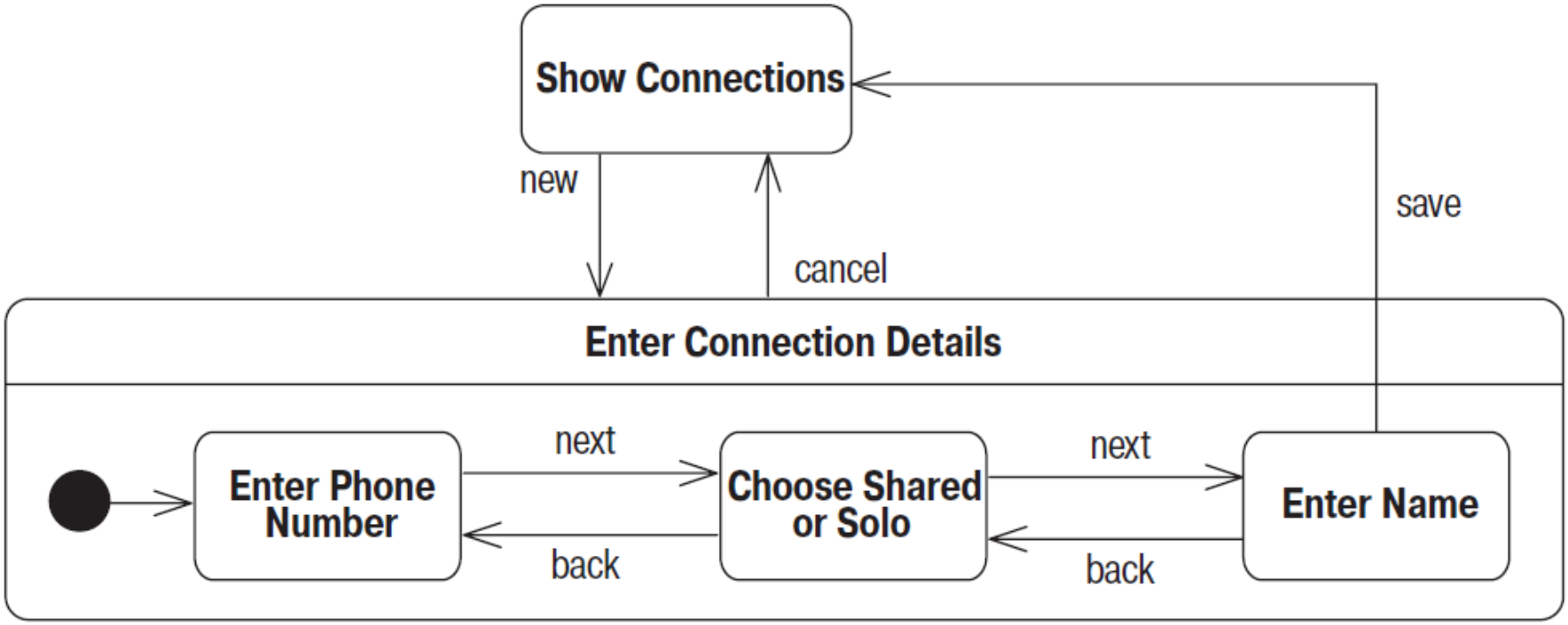
Диаграммы состояний
(State machine diagrams)

Пример диаграммы



Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

Суперсостояние

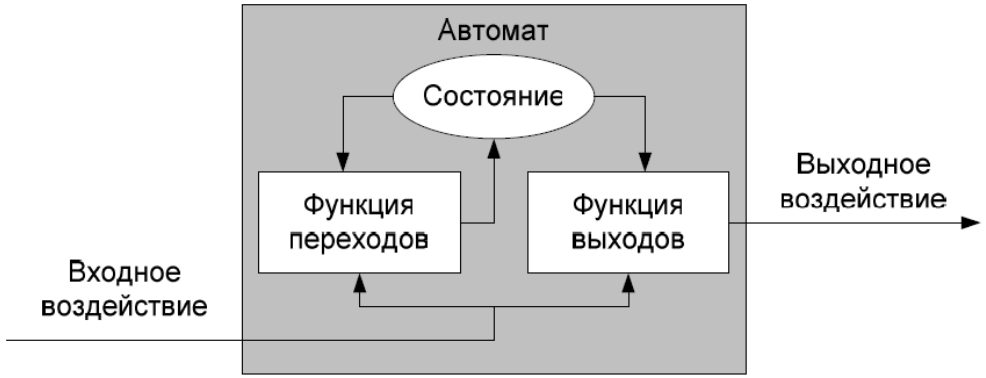


Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

Автоматное программирование

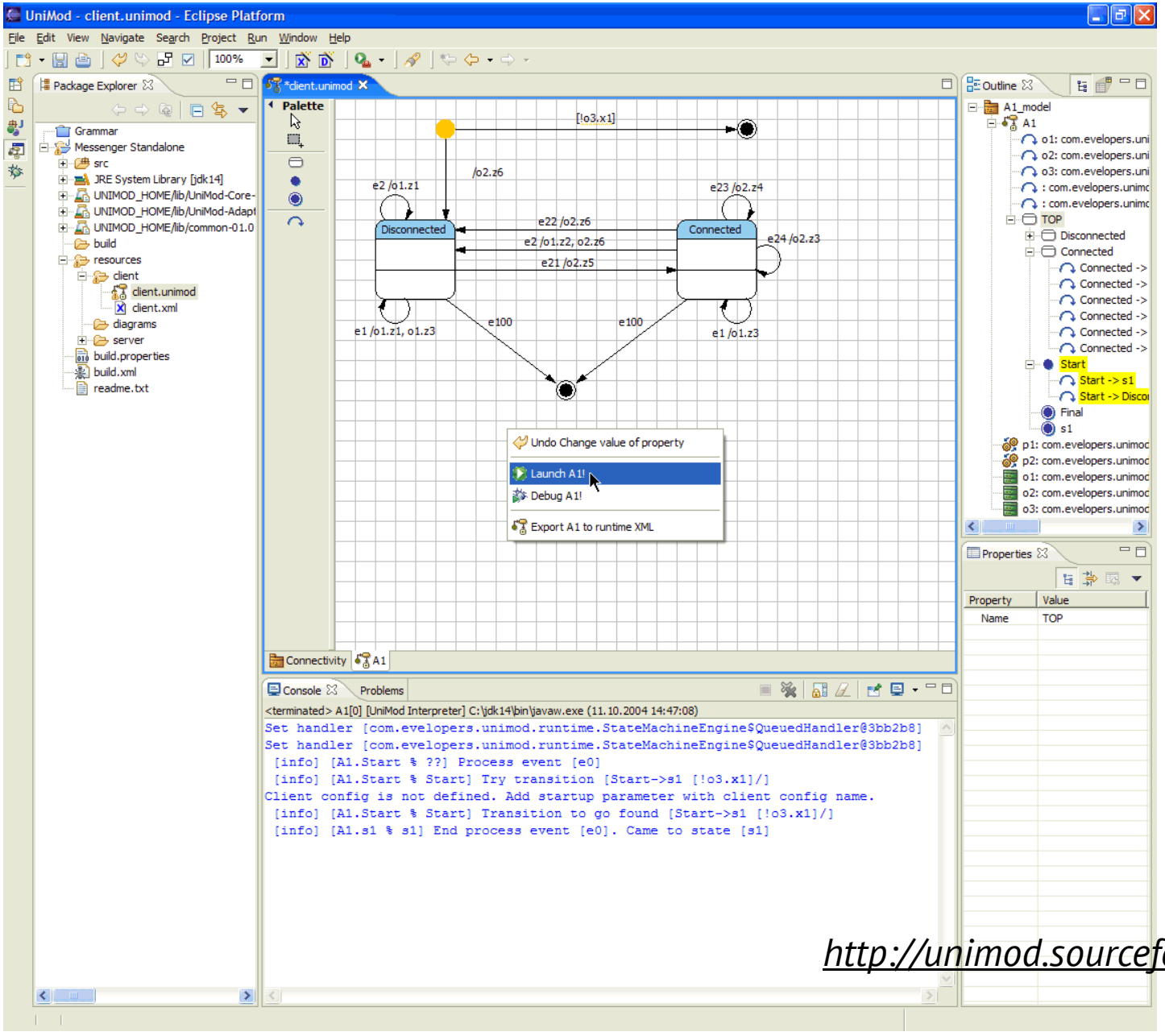
Автоматное программирование, иначе называемое «программирование от состояний» или «программирование с явным выделением состояний» – это метод разработки программного обеспечения, основанный на расширенной модели конечных автоматов и ориентированный на создание широкого класса приложений.

Поликарпова Н.И., Шалыто А.А. Автоматное программирование[2]. — СПб.: Питер, 2009. — 176 с. — ISBN 978-5-388-00692-9.



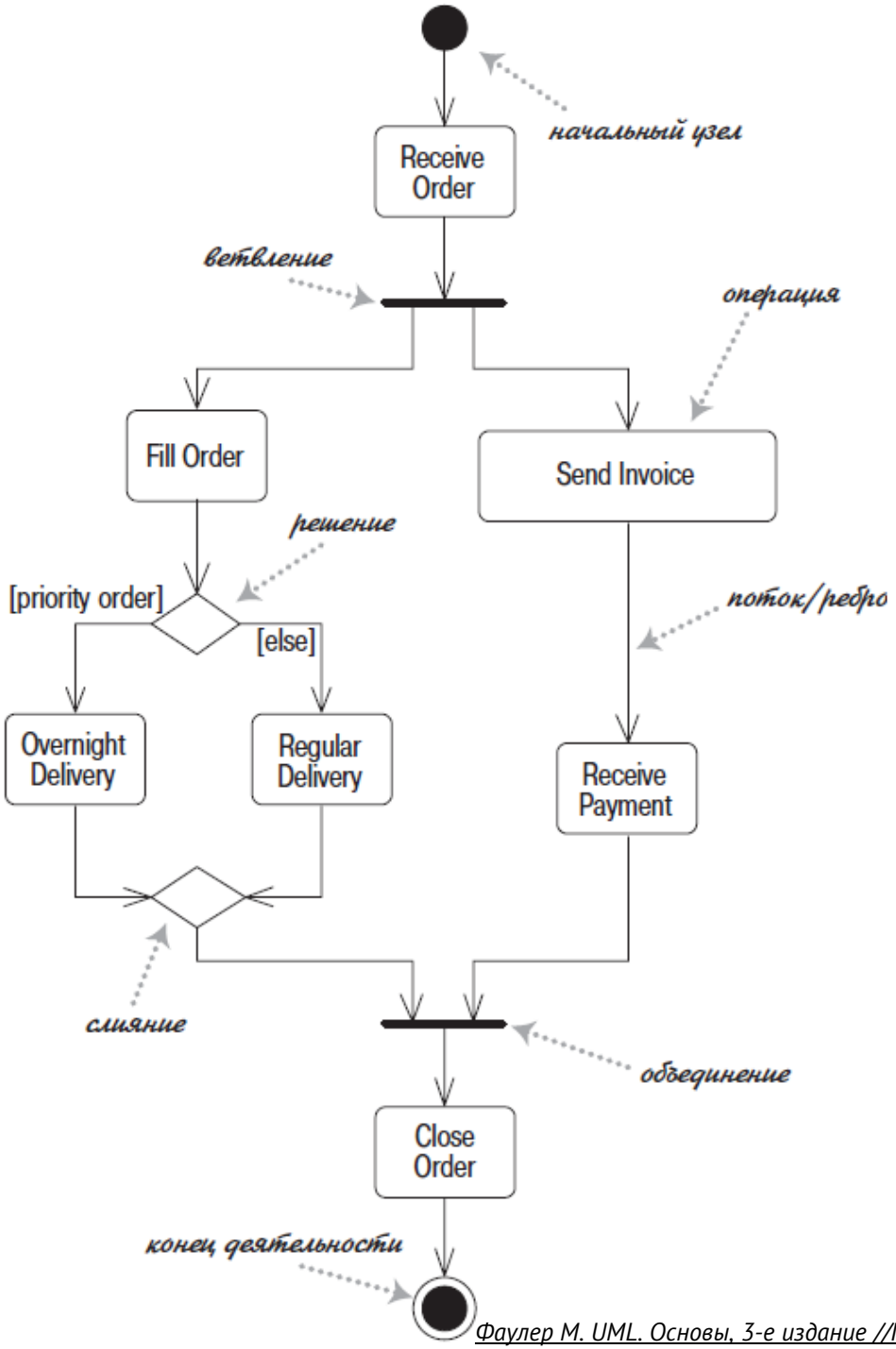
http://is.ifmo.ru/books/_book.pdf

UniMod

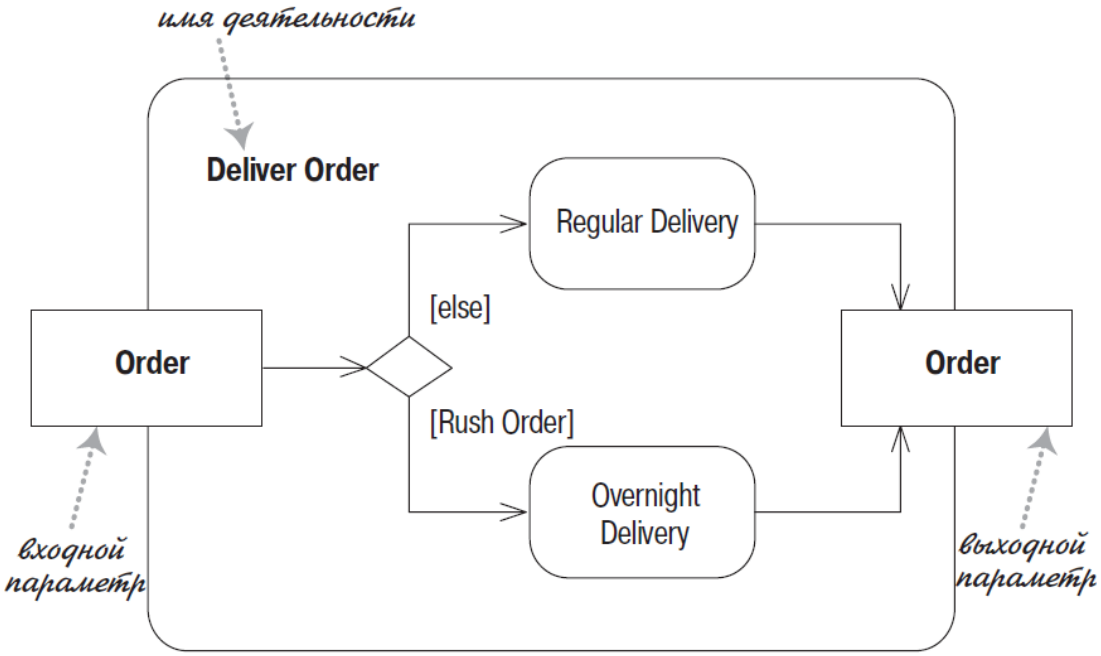


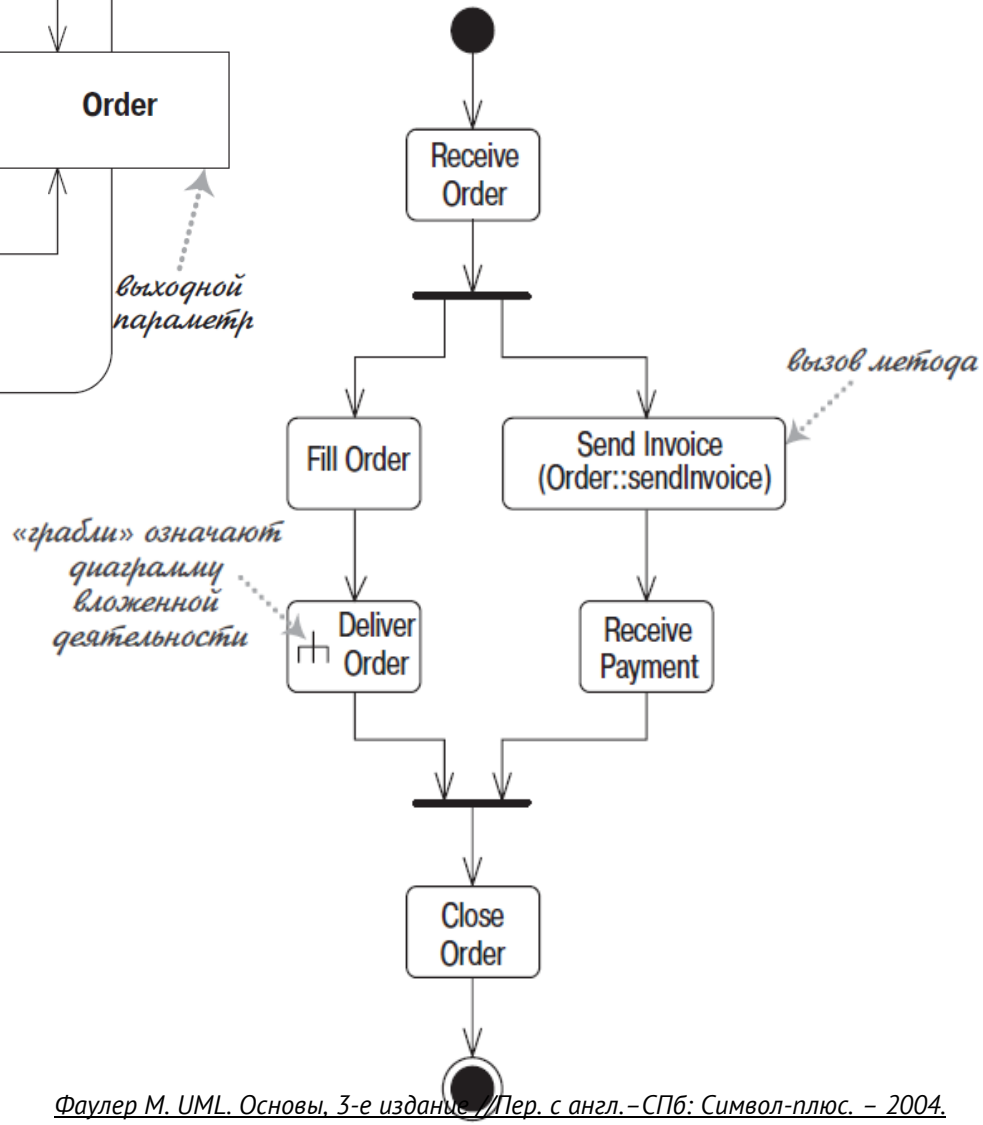
http://unimod.sourceforge.net/

Диаграммы деятельности (Activity diagrams)

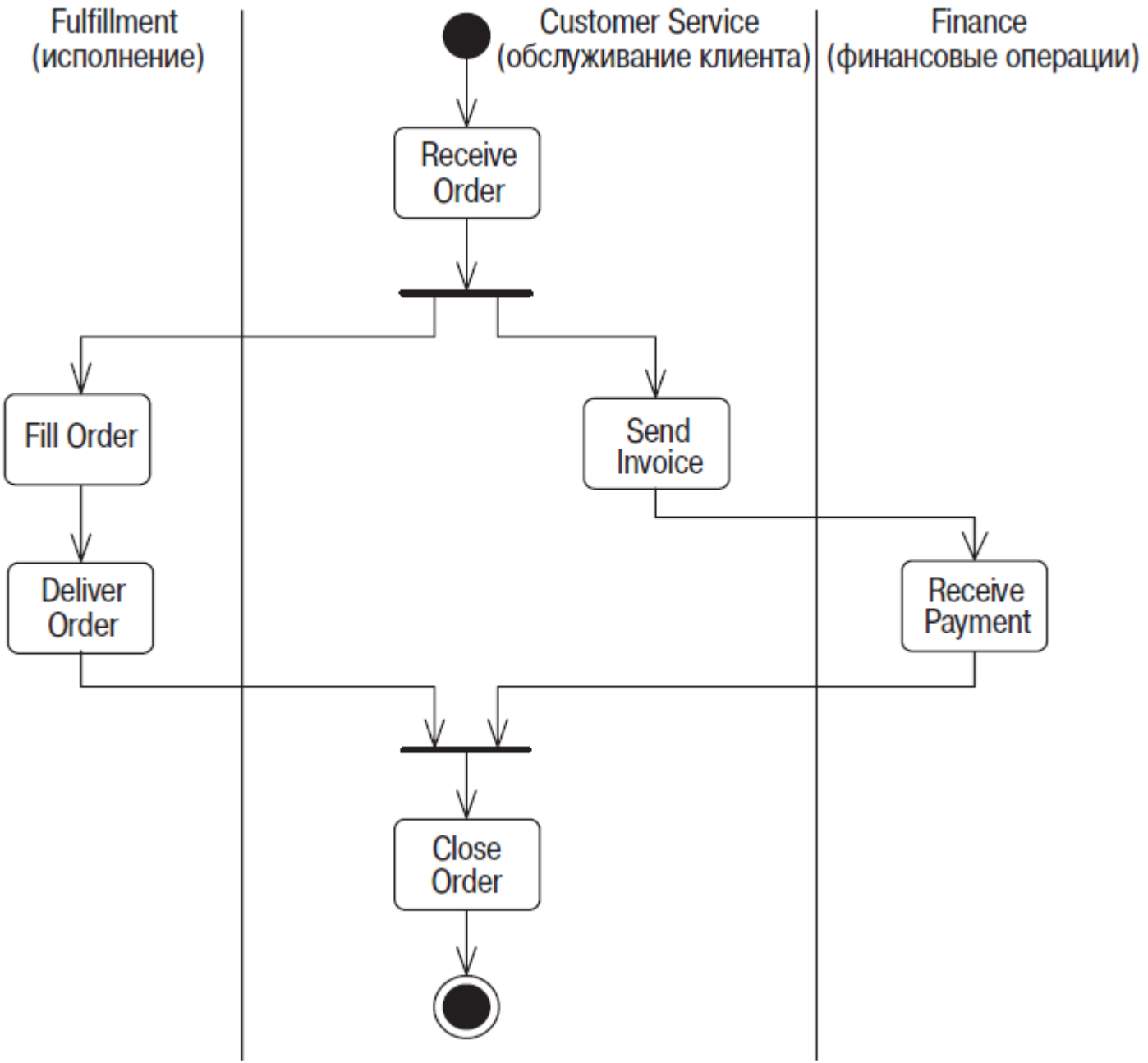


Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

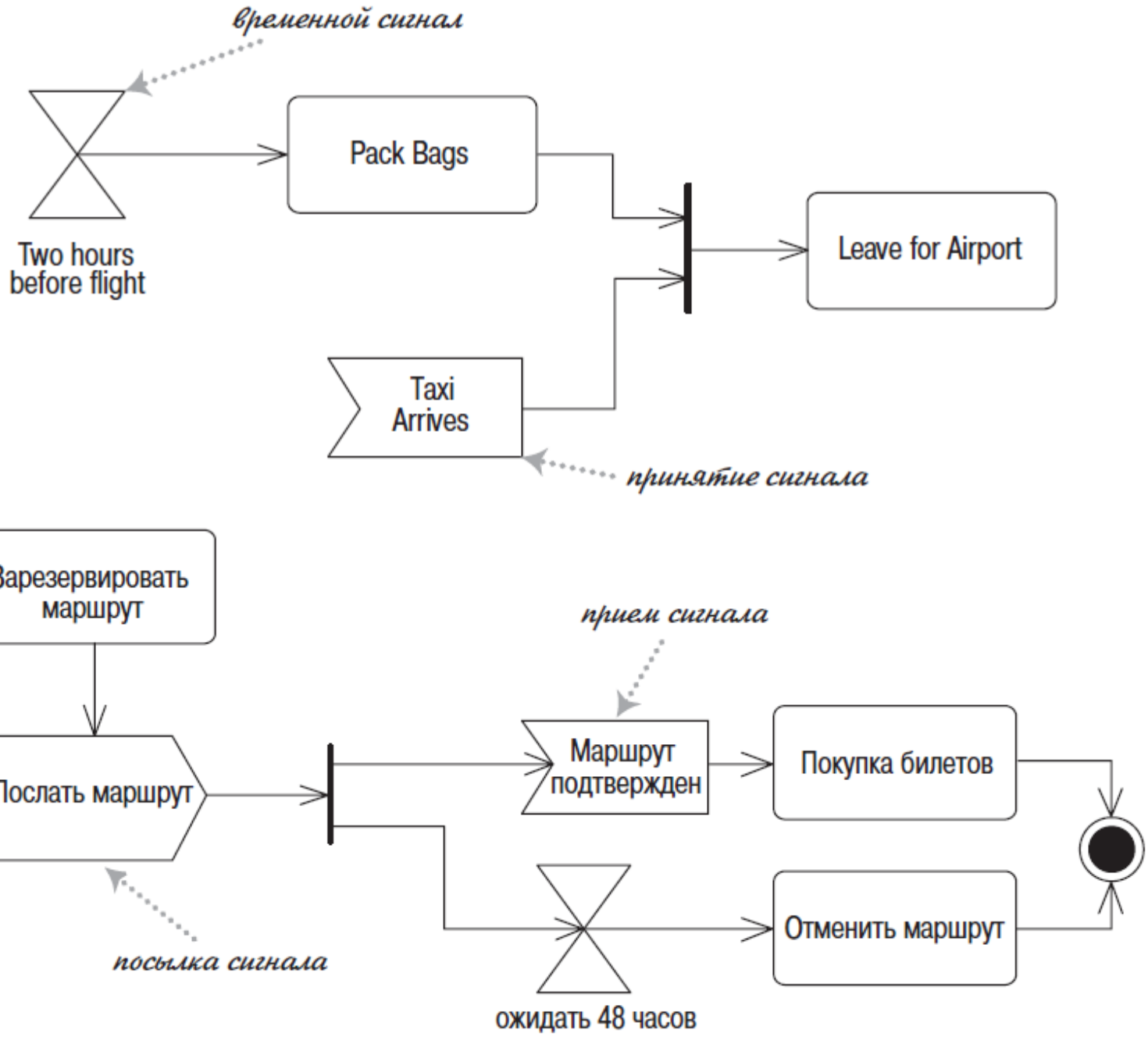




Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

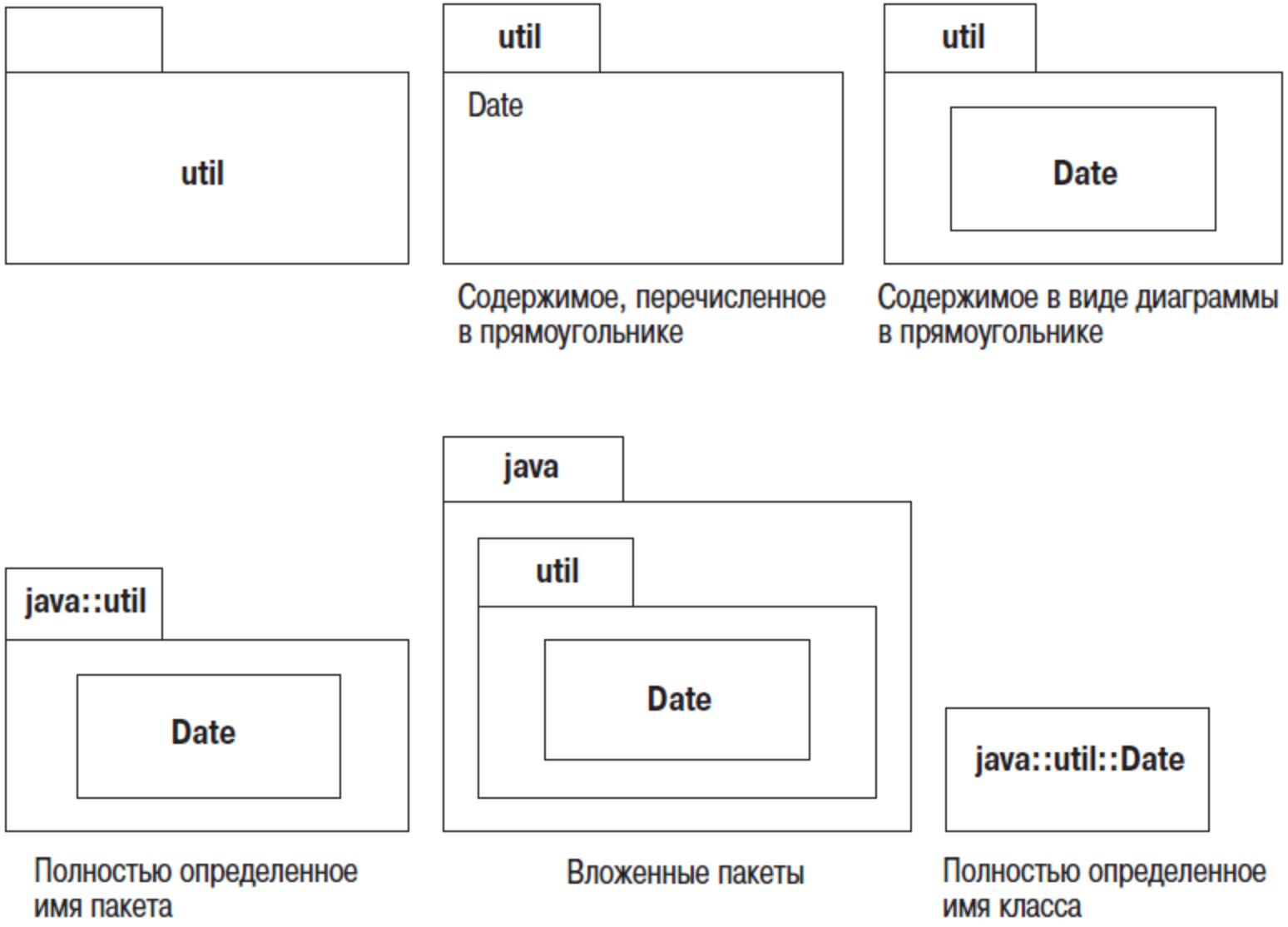


Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

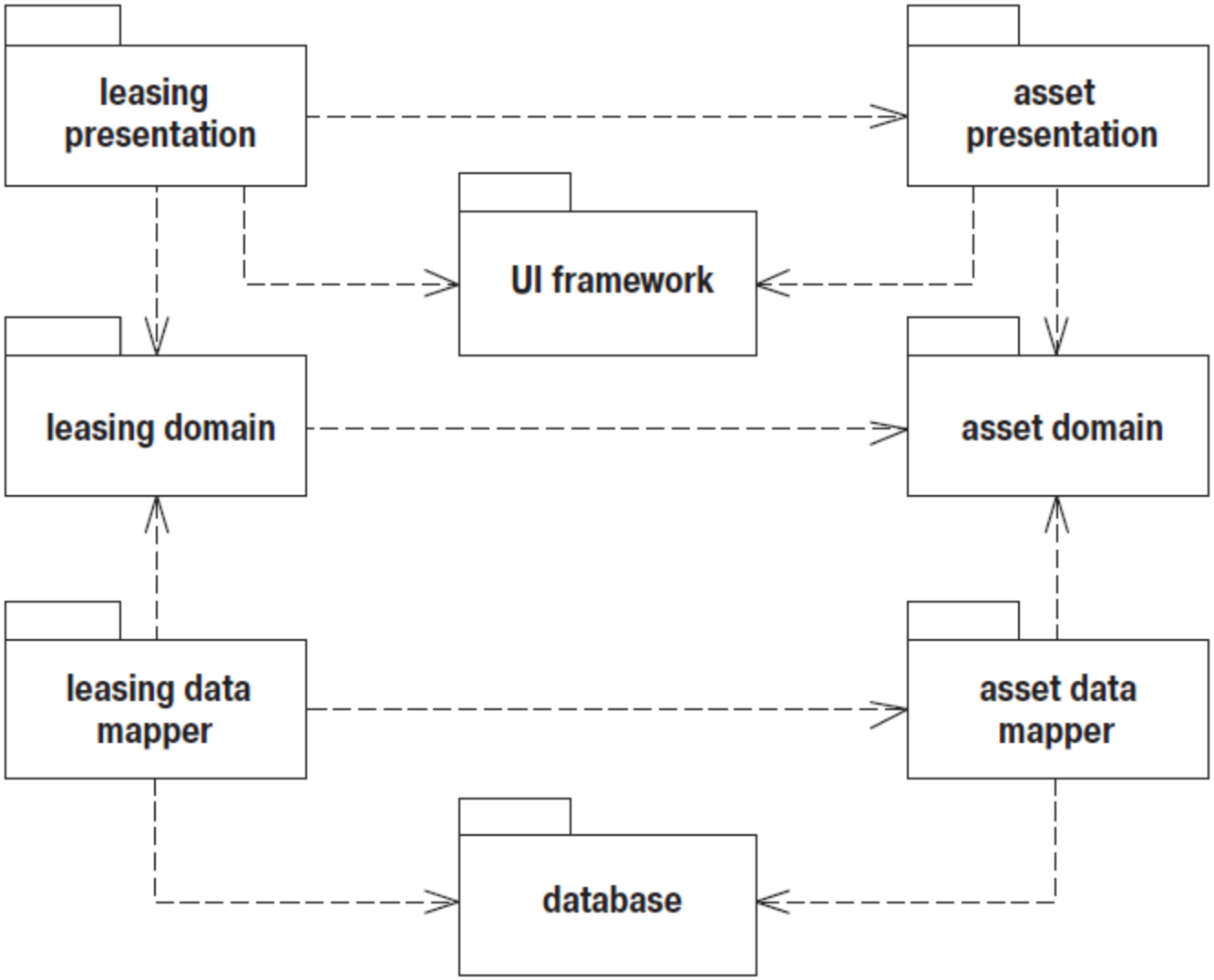


Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

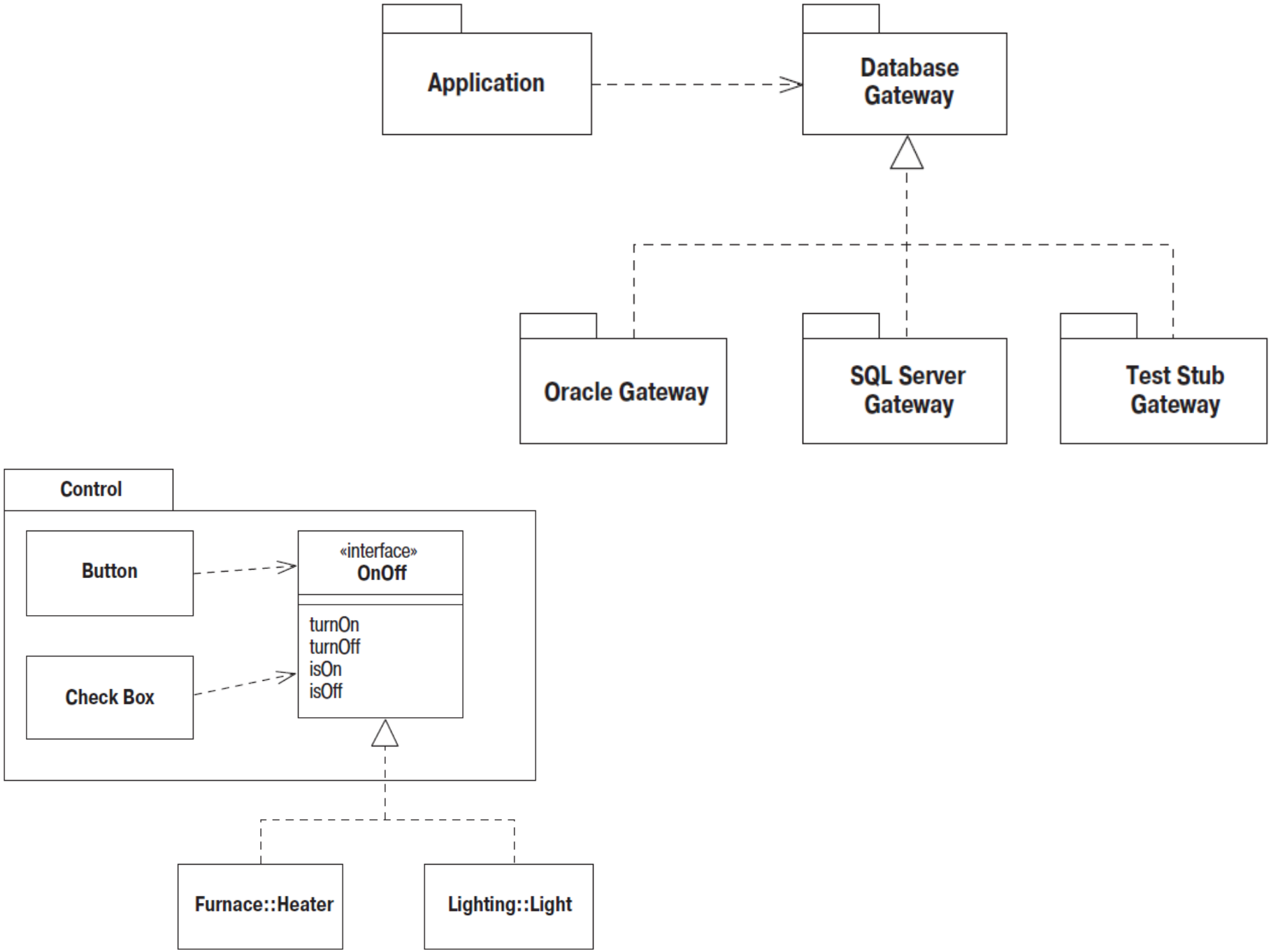
Диаграммы пакетов (package diagrams)



Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

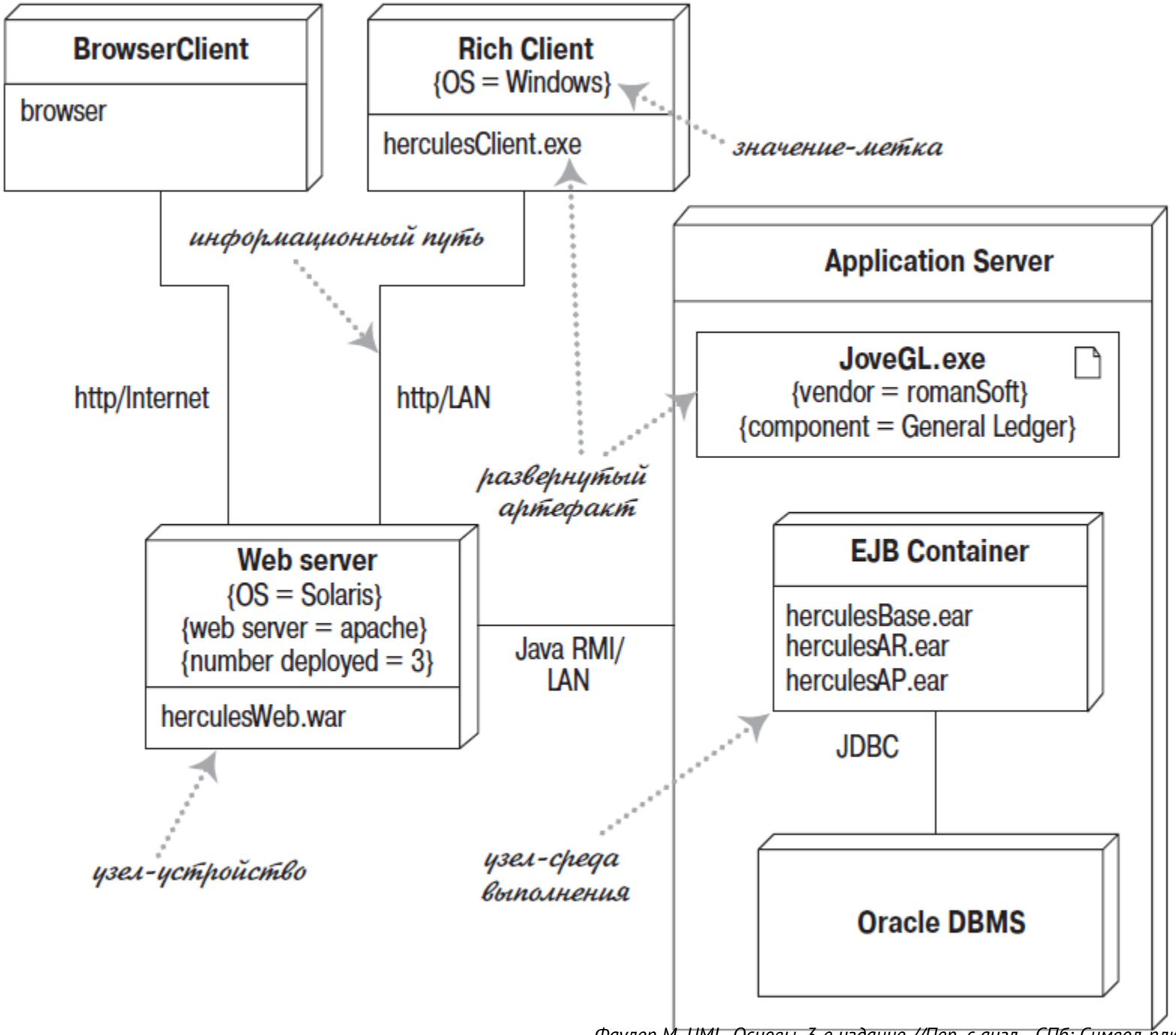


Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.



Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

Диаграммы развертывания (Deployment diagrams)



Фаулер М. UML. Основы, 3-е издание //Пер. с англ.–СПб: Символ-плюс. – 2004.

Источники

- **Книги**
  - Мартин Фаулер, Кендалл Скотт, «UML. Основы»
- **Полезные курсы**
  - Курс Дениса Иванова и Федора Новикова «UML для разработчиков» http://uml3.ru/library/uml_for_developers/uml_for_developers.html
- **Другие источники**
  - www.uml.org