

# Лекция 3: Моделирование как инструмент архитектуры

Юрий Литвинов  
[y.litvinov@spbu.ru](mailto:y.litvinov@spbu.ru)

07.03.2023

# Моделирование

- ▶ **Модель** — упрощённое подобие объекта или явления
- ▶ Нужны для изучения некоторых их свойств, абстрагируясь от сложности “настоящего” объекта или явления
- ▶ Модели используются повсеместно
  - ▶ Математические модели
  - ▶ Модели как реальные объекты
  - ▶ Модели в разработке ПО

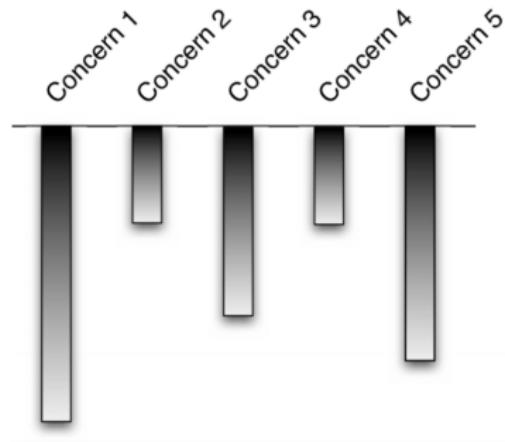
# Общие свойства моделей

- ▶ Содержат меньше информации, чем реальность
- ▶ Существуют для определённой цели
- ▶ Модели субъективны, что позволяет отделить существенные свойства от несущественных
- ▶ Моделирование ПО:
  - ▶ Модели предназначены прежде всего для управления сложностью
  - ▶ Позволяют понять, проанализировать и протестировать систему до её реализации

**All models are wrong, some are useful**

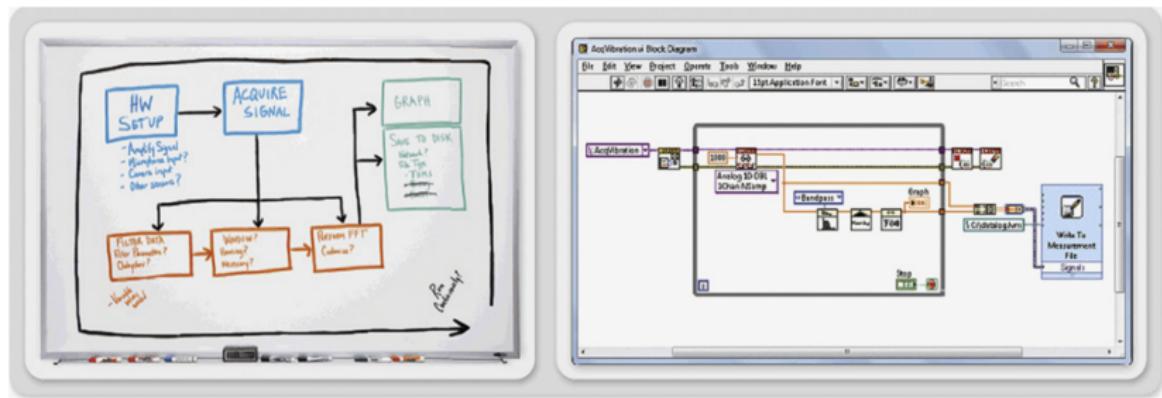
# Как выбрать, что моделировать?

- ▶ При моделировании надо определиться с:
  - ▶ Какие архитектурные решения нуждаются в моделировании
  - ▶ На каком уровне детализации
  - ▶ Насколько формально
- ▶ Необходимо учитывать соотношение трудозатрат и выгоды
  - ▶ Стоимость создания *и поддержания* модели не должна быть больше преимуществ от её использования



# Модели бывают разные

- ▶ Используемые нотации и способы моделирования зависят от целей моделирования
  - ▶ От неформальных набросков до исполнимых моделей



© N. Medvidovic

# Виды моделей

## Естественные языки

- ▶ Обычный текст — вполне себе инструмент моделирования
- ▶ Очень выразителен, не требует специальных знаний, максимально гибок
- ▶ Неоднозначен, неформален, не строг, слишком многословен, бесполезен для автоматической обработки

*"The Lunar Lander application consists of three components: a **data store** component, a **calculation** component, and a **user interface** component.*

*The job of the **data store** component is to store and allow other components access to the height, velocity, and fuel of the lander, as well as the current simulator time.*

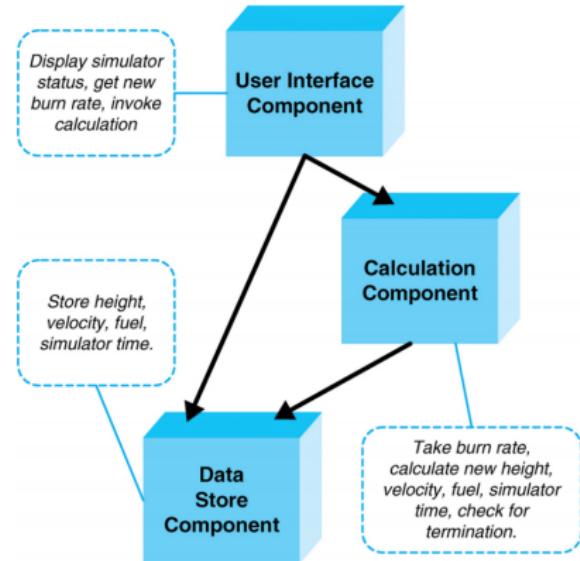
*The job of the **calculation** component is to, upon receipt of a burn-rate quantity, retrieve current values of height, velocity, and fuel from the data store component, update them with respect to the input burn-rate, and store the new values back. It also retrieves, increments, and stores back the simulator time. It is also responsible for notifying the calling component of whether the simulator has terminated, and with what state (landed safely, crashed, and so on).*

*The job of the **user interface** component is to display the current status of the lander using information from both the calculation and the data store components. While the simulator is running, it retrieves the new burn-rate value from the user, and invokes the calculation component."*

© N. Medvidovic

# Неформальные графические модели

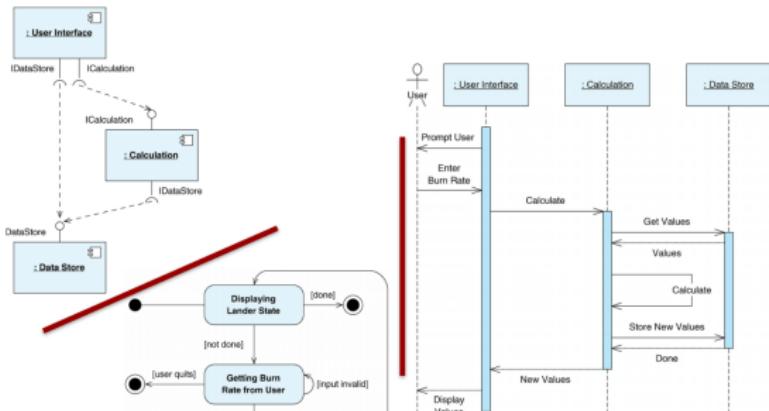
- ▶ Диаграммы, рисуемые в PowerPoint, Inkscape и подобном
- ▶ Могут быть красивыми, как правило, простые, очень гибкая нотация
- ▶ Неформальны, неоднозначны, не строги
  - ▶ Но часто воспринимаются наоборот
- ▶ Практически бесполезны для автоматической обработки



© N. Medvidovic

# UML и SysML

- ▶ Несколько слабо связанных нотаций (“диаграмм”)
- ▶ Поддерживают много точек зрения, общеприняты, широкая поддержка инструментами
- ▶ Нет строгой семантики, сложно обеспечить консистентность, сложно расширять



© N. Medvidovic

# AADL и другие текстовые формальные языки

- ▶ Хороши для моделирования встроенных систем и систем реального времени
- ▶ Описывают одновременно “железо” и “софт”, продвинутые инструменты анализа
- ▶ Слишком многословны и детальны, сложны в изучении и использовании

```

data lander_state_data
end lander_state_data;
bus lan_bus_type
end lan_bus_type;

bus implementation lan_bus_type.ethernet
properties
    Transmission_Time => 1 ms .. 5 ms;
    Allowed_Message_Size => 1 b .. 1 kb;
end lan_bus_type.ethernet;
system calculation_type
features
    network : requires bus access
        lan_bus.calculation_to_datastore;
    request_get   : out event port;
    response_get  : in event data port lander_state_data;
    request_store : out event port lander_state_data;
    response_store : in event port;
end calculation_type;

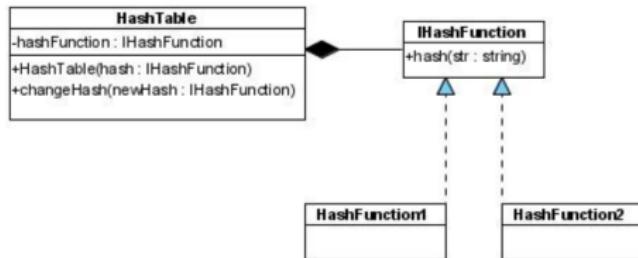
system implementation calculation_type.calculation
subcomponents
    the_calculation_processor :
        processor calculation_processor_type;
    the_calculation_process : process
        calculation_processor_type.one_thread;

```

© N. Medvidovic

# Вернёмся к визуальным моделям

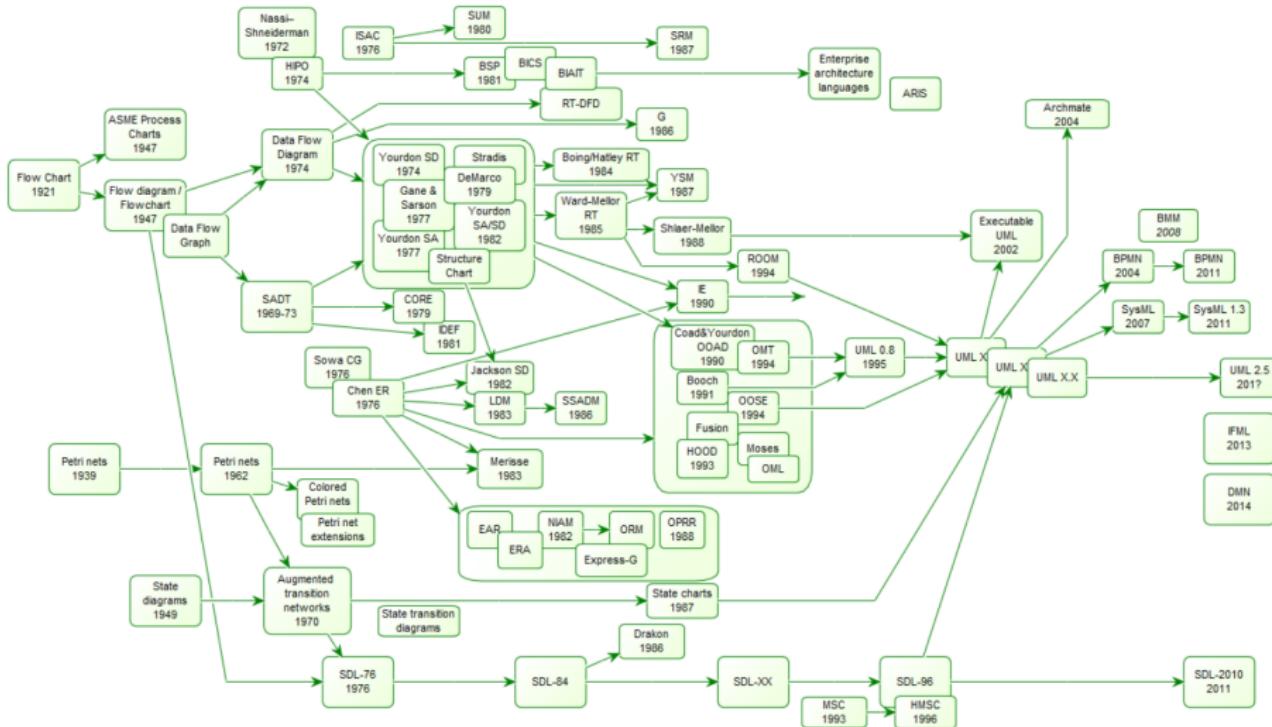
- ▶ **Метафора визуализации** — договорённость о том, как будут представляться сущности языка
- ▶ **Точка зрения моделирования** — какой аспект системы и для кого моделируется
- ▶ Бывают одноразовые модели, документация и графические исходники
  - ▶ **Семантический разрыв** — неспособность модели полностью специфицировать систему



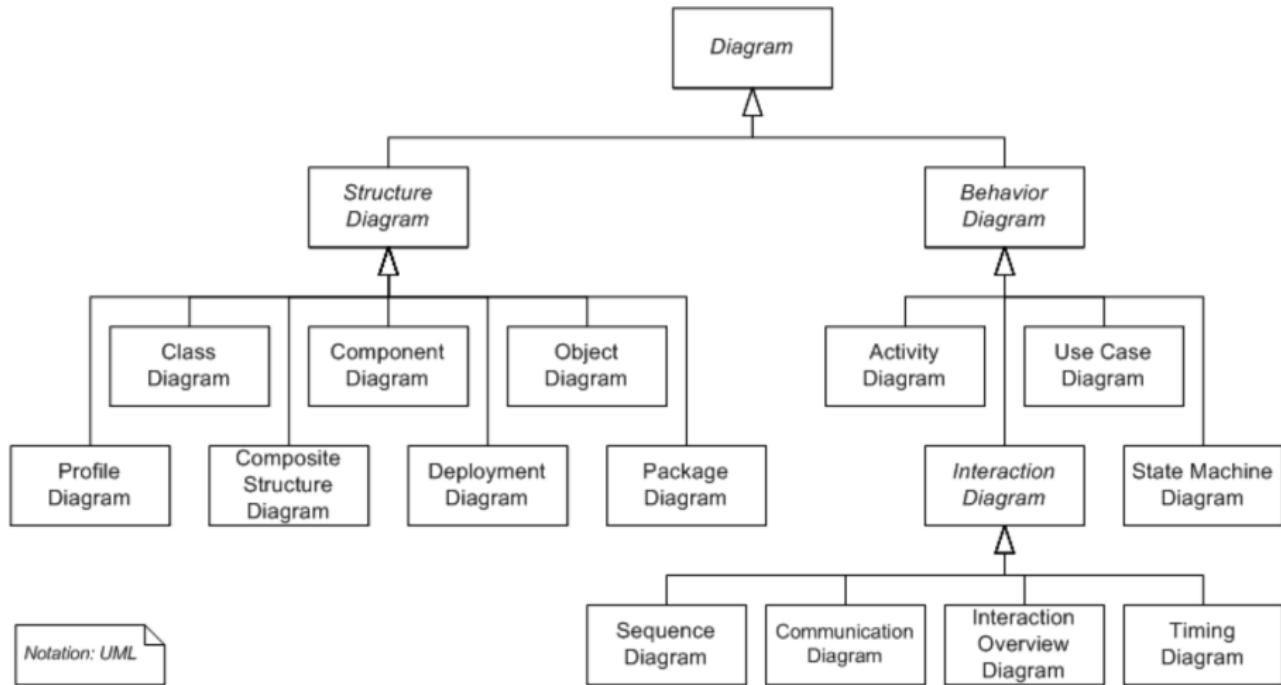
# Unified Modeling Language

- ▶ Семейство графических нотаций
  - ▶ 14 видов диаграмм
- ▶ Общая метамодель
- ▶ Стандарт под управлением Object Management Group
  - ▶ UML 1.1 — 1997 год
  - ▶ UML 2.0 — 2005 год
  - ▶ UML 2.5.1 — декабрь 2017 года
- ▶ Прежде всего, для проектирования ПО
  - ▶ После UML 2.0 стали появляться нотации и для инженеров
- ▶ Расширяем
  - ▶ Профили — механизм легковесного расширения
  - ▶ Метамоделирование

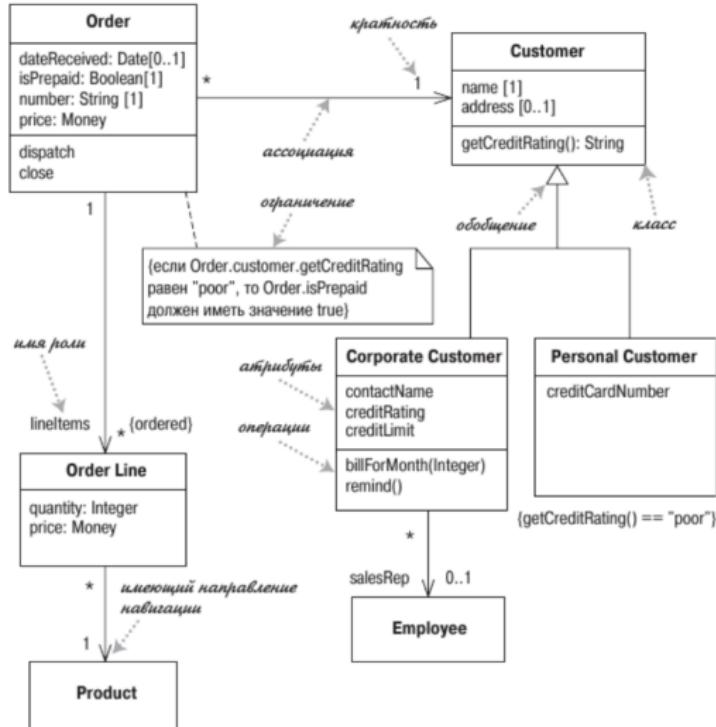
# История



# Виды диаграмм



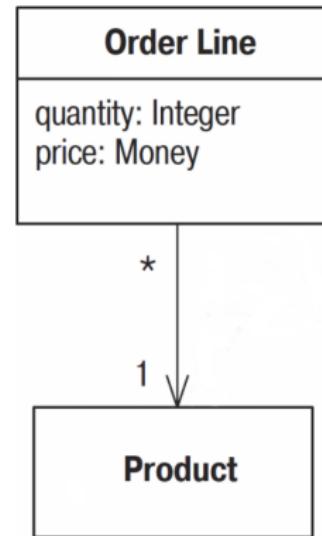
# Диаграмма классов



© М. Фаулер. “UML. Основы”

# Как это связано с кодом

```
public class OrderLine {
    private int quantity;
    private Product product;
    public int getQuantity() {
        return quantity;
    }
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
    public Money getPrice() {
        return product.getPrice().multiply(quantity);
    }
}
```



# Двунаправленные ассоциации



```

class Car {
    public Person Owner {
        get { return _owner; }
        set {
            if (_owner != null)
                _owner.friendCars().Remove(this);
            _owner = value;
            if (_owner != null)
                _owner.friendCars().Add(this);
        }
    }
    private Person _owner;
}
  
```

```

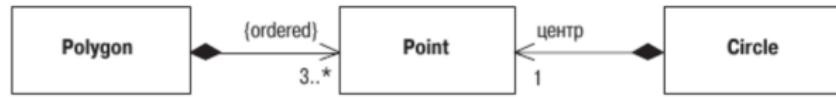
class Person {
    public IList Cars {
        get { return ArrayList.ReadOnly(_cars); }
    }
    public void AddCar(Car arg) {
        arg.Owner = this;
    }
    private IList _cars = new ArrayList();
    internal IList friendCars() {
        // должен быть использован
        // только Car.Owner
        return _cars;
    }
}
  
```

# Агрегация и композиция

Агрегация:

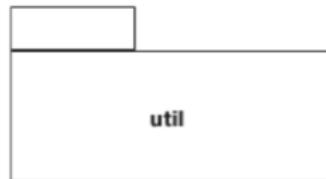


Композиция:



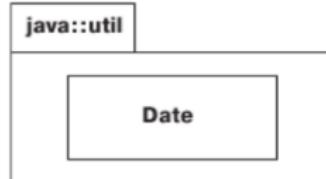
© М. Фаулер. “UML. Основы”

# Диаграммы пакетов

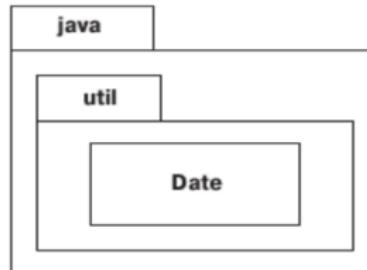


Содержимое, перечисленное в прямоугольнике

Содержимое в виде диаграммы в прямоугольнике



Полностью определенное имя пакета



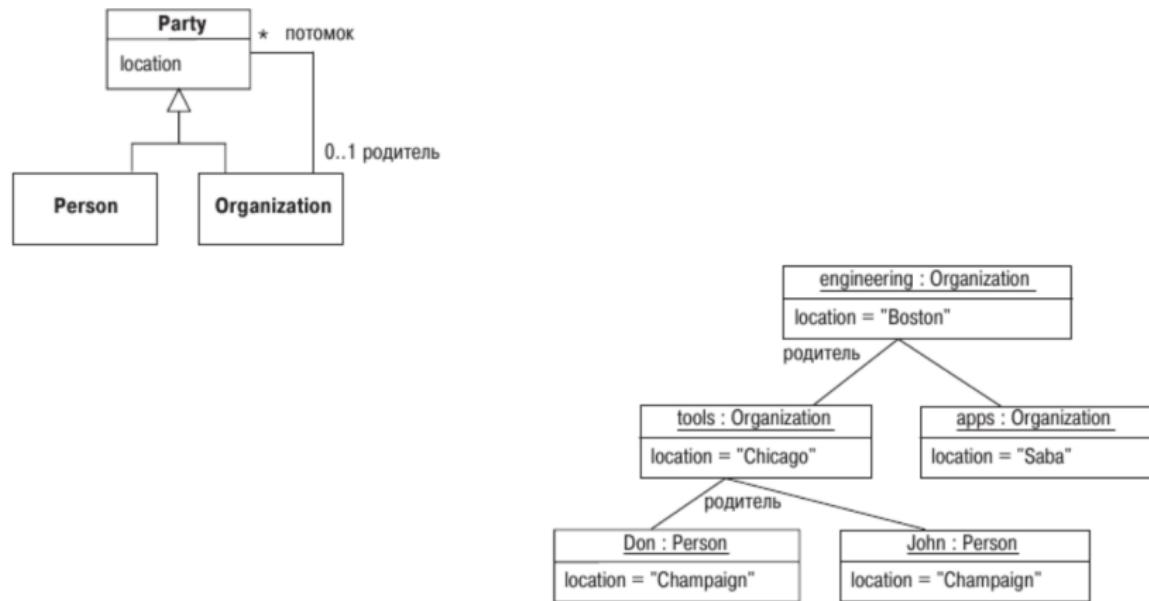
Вложенные пакеты



Полностью определенное имя класса

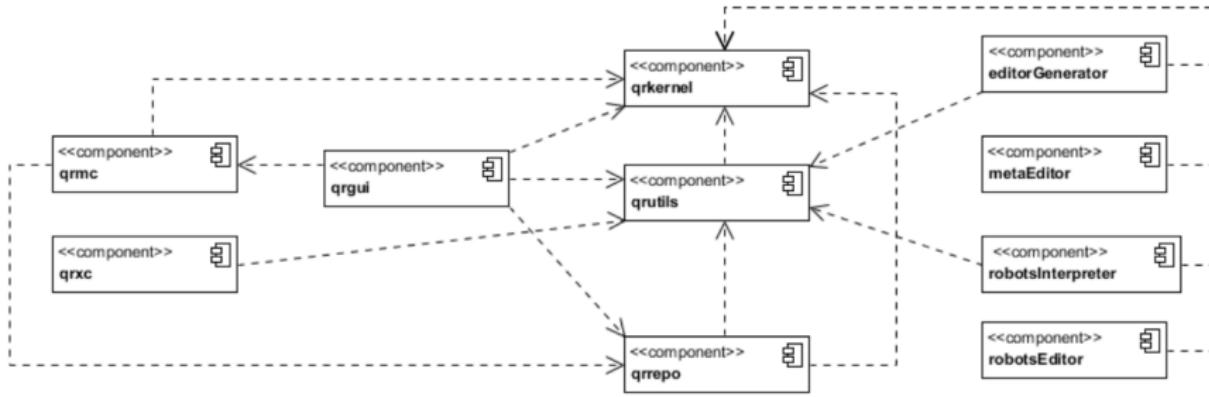
© M. Фаулер. “UML. Основы”

# Диаграммы объектов

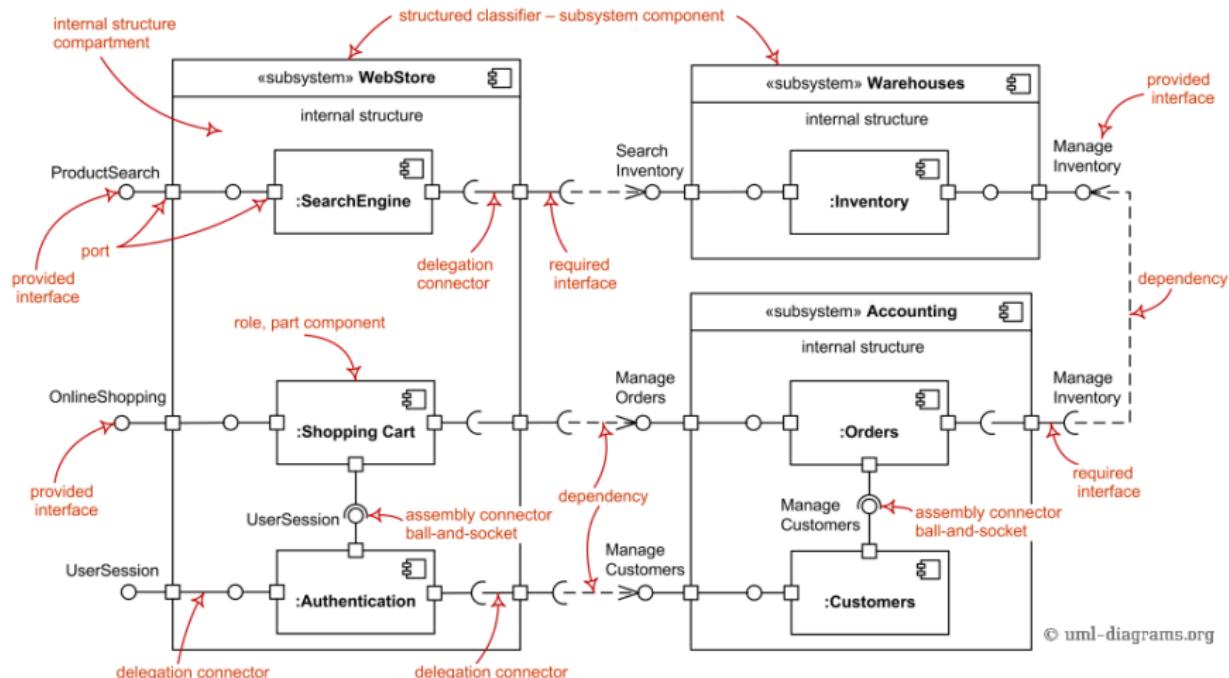


© М. Фаулер. “UML. Основы”

# Диаграммы компонентов



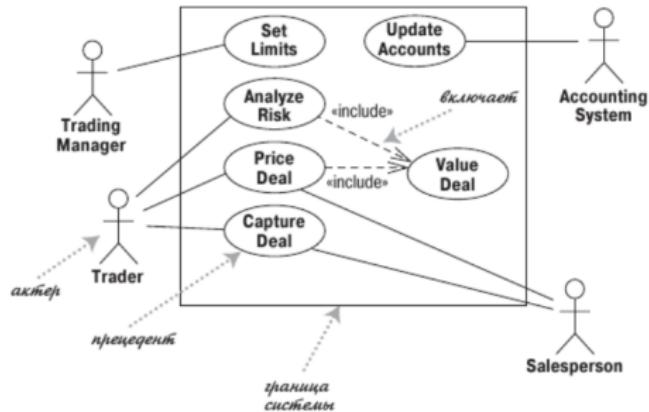
# Более подробно



# Диаграмма случаев использования UML

## Диаграмма прецедентов

- ▶ Ивар Якобсон, 1992 год
- ▶ Акторы (или актёры, роли) — внешние сущности, использующие систему
  - ▶ Люди или другие программные системы
- ▶ Случаи использования (прецеденты) — цель использования системы актором
  - ▶ Раскрываются в набор сценариев, описываемых чаще текстом



© М. Фаулер, UML. Основы

# Сценарий использования, типичная структура

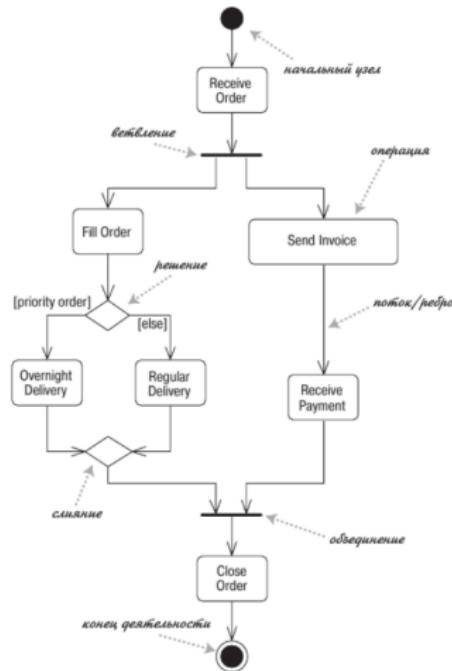
- ▶ Заголовок (цель основного актора)
- ▶ Заинтересованные лица, акторы, основной актор
- ▶ Предусловия
- ▶ Триггеры (активаторы)
- ▶ Основной порядок событий
- ▶ Альтернативные пути и расширения
- ▶ Постусловия

<b>Use Case Name:</b> Request a chemical	<b>ID:</b> UC-2	<b>Priority:</b> High
<b>Actor:</b> Lawn Chemical Applicator (LCA)		
<b>Description:</b> The Lawn Chemical Applicator (LCA) specifies the lawn chemical needed for a job by entering its name or ID number. The system satisfies the request by reserving the quantity requested or the quantity available and notifying the Chemical Supply Warehouse of the pick-up.		
<b>Trigger:</b> A Lawn Chemical Applicator (LCA) needs a chemical for a job.		
<b>Type:</b> <input checked="" type="checkbox"/> External <input type="checkbox"/> Temporal		
<b>Preconditions:</b>		
<ol style="list-style-type: none"> <li>1. The LCA identity is authenticated.</li> <li>2. The LCA has necessary training and credentials on file.</li> <li>3. The Chemical Supply datastore is up-to-date and on-line.</li> </ol>		
<b>Normal Course:</b>		
<ol style="list-style-type: none"> <li>1.0 Request a lawn chemical from the chemical supply warehouse.</li> <li>1. The LCA specifies a chemical needed and the quantity needed</li> <li>2. The system lists chemical and quantity on hand from Chemical Supply datastore             <ol style="list-style-type: none"> <li>a. If the quantity on hand is less than the quantity needed, the LCA specifies the quantity he will take</li> <li>b. Purchasing is notified of chemical shortage</li> </ol> </li> <li>3. The system gives the LCA a Chemical Pick-up Authorization for the quantity requested</li> <li>4. The system notifies the Chemical Supply Warehouse of the chemical pick-up</li> <li>5. The system stores the Lawn Chemical Request in the Chemical Request datastore</li> </ol>		
<b>Postconditions:</b>		
<ol style="list-style-type: none"> <li>1. The Lawn Chemical Request is stored in the Chemical Management System.</li> <li>2. The Chemical Pick-up Authorization is produced for the LCA.</li> <li>3. The Chemical Supply Warehouse is notified of the chemical pick-up.</li> <li>4. Purchasing is notified of chemical outage.</li> </ol>		
<b>Exceptions:</b>		
<p>E1: Chemical is no longer approved for use (occurs at step 1)</p> <ol style="list-style-type: none"> <li>1. The system displays message. "That chemical is no longer approved for use"</li> <li>2. The system asks the LCA if he wants to request another chemical or to exit             <ol style="list-style-type: none"> <li>3a. The LCA asks to request another chemical</li> <li>4a. The system starts Normal Course again</li> <li>3b. The LCA asks to exit</li> <li>4b. The system terminates the use case</li> </ol> </li> </ol>		

# Диаграмма активностей UML

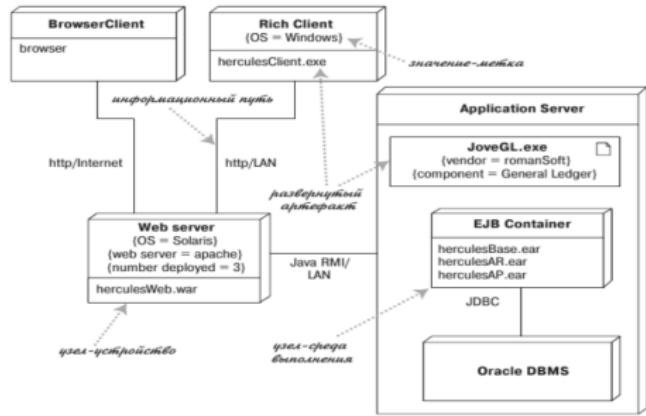
## Диаграммы деятельности

- ▶ Используются для моделирования бизнес-процессов, тоже на первых этапах
  - ▶ Может быть визуализацией сценария использования
- ▶ Иногда — для моделирования алгоритма
- ▶ Расширенные блок-схемы
- ▶ Семантика на основе сетей Петри



# Диаграмма развёртывания UML

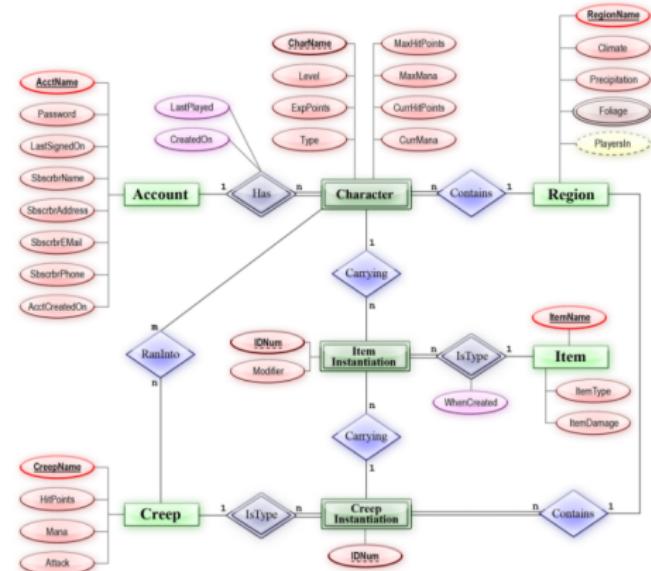
- ▶ Показывает отображение компонентов и физических артефактов на реальные (или виртуальные) устройства
- ▶ Бывает полезна на начальных этапах проектирования, даже до диаграмм компонентов



© М. Фаулер, UML. Основы

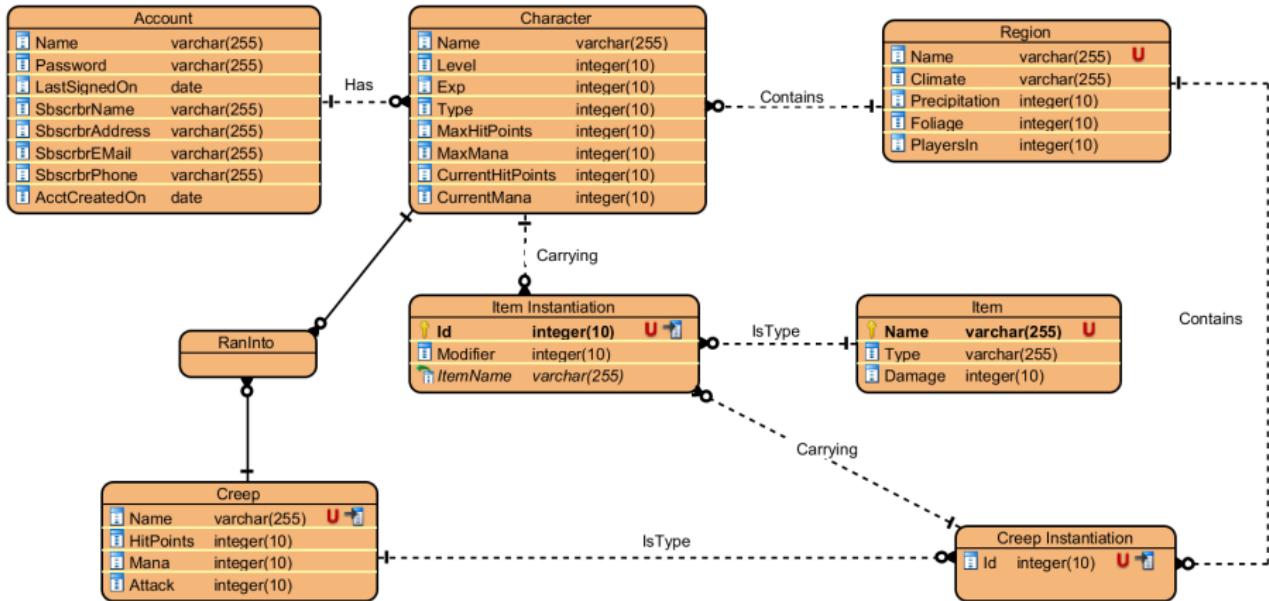
# Диаграммы “Сущность-связь”

- ▶ Описывают концептуальную модель предметной области
- ▶ Идеальны для моделирования схем реляционных баз данных
- ▶ 1976 год, Питер Чен



© <https://ru.wikipedia.org>

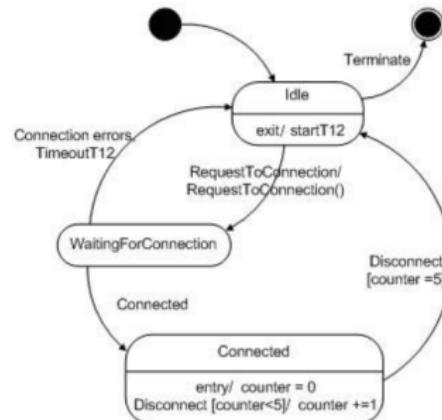
# Нотация “Вороньей лапки”



# Диаграммы конечных автоматов

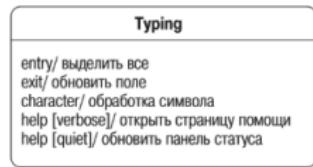
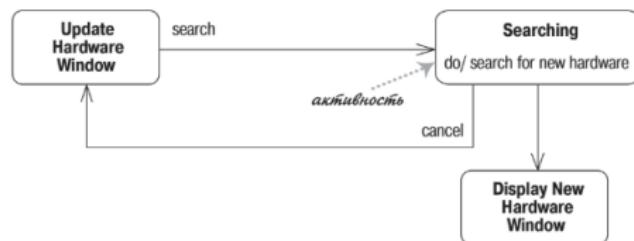
## Диаграммы состояний

- ▶ Состояния объекта как часть жизненного цикла
- ▶ Моделирование реактивных объектов
  - ▶ Например, сетевое соединение
  - ▶ Или знакомый пример с торговым автоматом
- ▶ Имеют исполнимую семантику
- ▶ Д. Харел, 1987

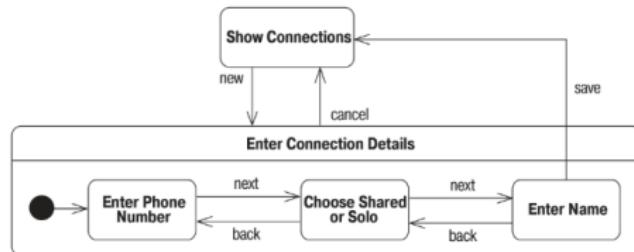


# Диаграммы конечных автоматов, особенности

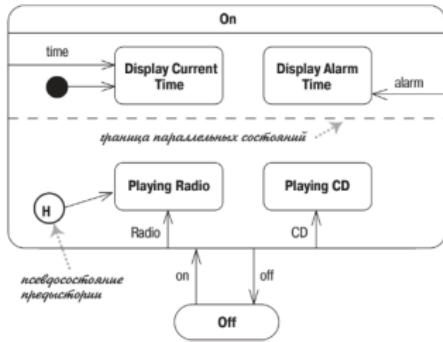
## Активности:



## Вложенные состояния:



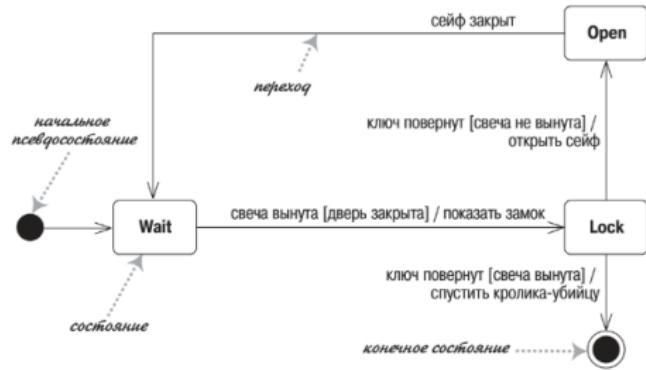
## Параллельные состояния, псевдосостояние истории:



© M. Фаулер, UML. Основы

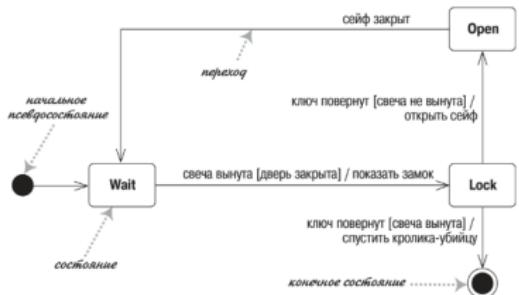
# Генерация кода

```
public void handleEvent(PanelEvent anEvent) {
    switch (currentState) {
        case PanelState.Open:
            switch (anEvent) {
                case PanelEvent.SafeClosed:
                    currentState = PanelState.Wait;
            }
            break;
        case PanelState.Wait:
            switch (anEvent) {
                case PanelEvent.CandleRemoved:
                    if (isDoorOpen) {
                        revealLock();
                        currentState = PanelState.Lock;
                    }
            }
            break;
        case PanelState.Lock:
            switch (anEvent) {
                case PanelEvent.KeyTurned:
                    if (isCandleIn) {
                        openSafe();
                        currentState = PanelState.Open;
                    } else {
                        releaseKillerRabbit();
                        currentState = PanelState.Final;
                    }
            }
            break;
    }
}
```



© M. Фаулер, UML. Основы

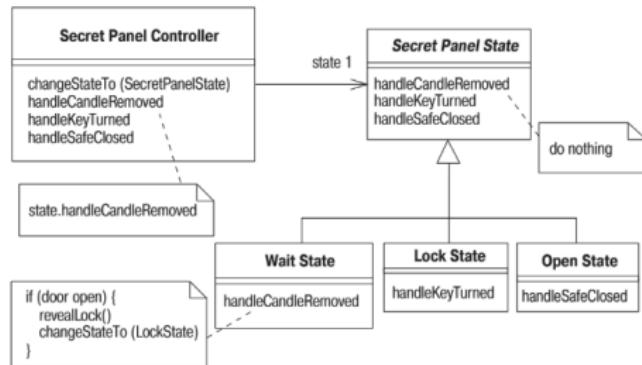
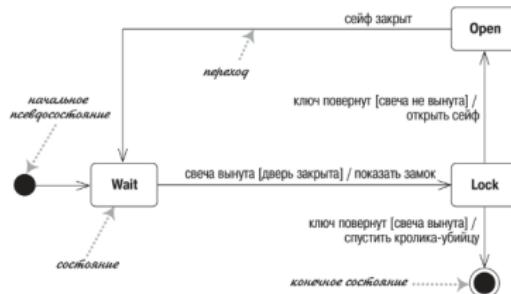
# Таблица состояний



Исходное состояние	Целевое состояние	Событие	Защита	Процедура
Wait	Lock	Candle removed (свеча удалена)	Door open (дверца открыта)	Reveal lock (показать замок)
Lock	Open	Key turned (ключ повернут)	Candle in (свеча на месте)	Open safe (открыть сейф)
Lock	Final	Key turned (ключ повернут)	Candle out (свеча удалена)	Release killer rabbit (освободить убийцу-кролика)
Open	Wait	Safe closed (сейф закрыт)		

© М. Фаулер, UML. Основы

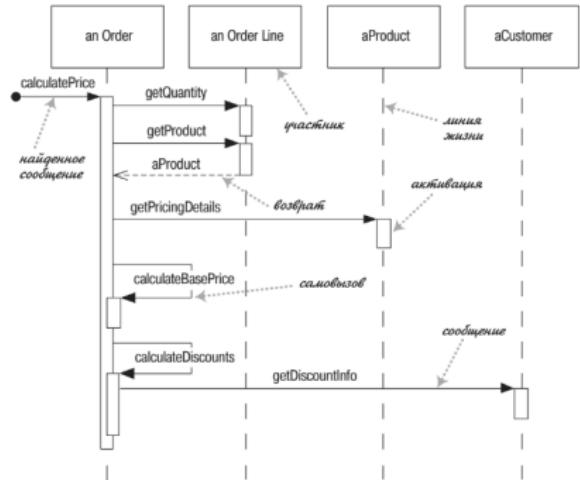
# Паттерн “Состояние”



© М. Фаулер, UML. Основы

# Диаграммы последовательностей

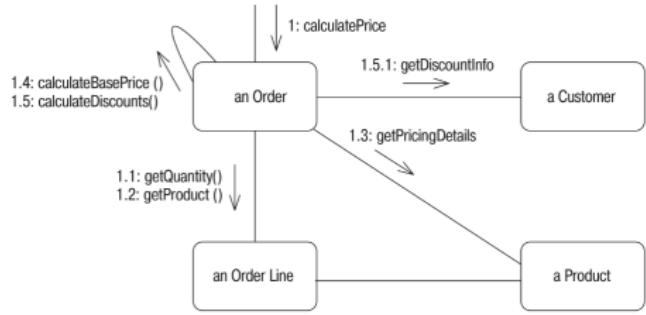
- ▶ Применяются для визуализации взаимодействия между объектами
  - ▶ Особо удобно для асинхронных вызовов
  - ▶ Телекоммуникационные протоколы
- ▶ Могут применяться на этапе анализа предметной области
- ▶ Могут применяться для составления плана тестирования
- ▶ И даже для визуализации логов работающей системы



© М. Фаулер, UML. Основы

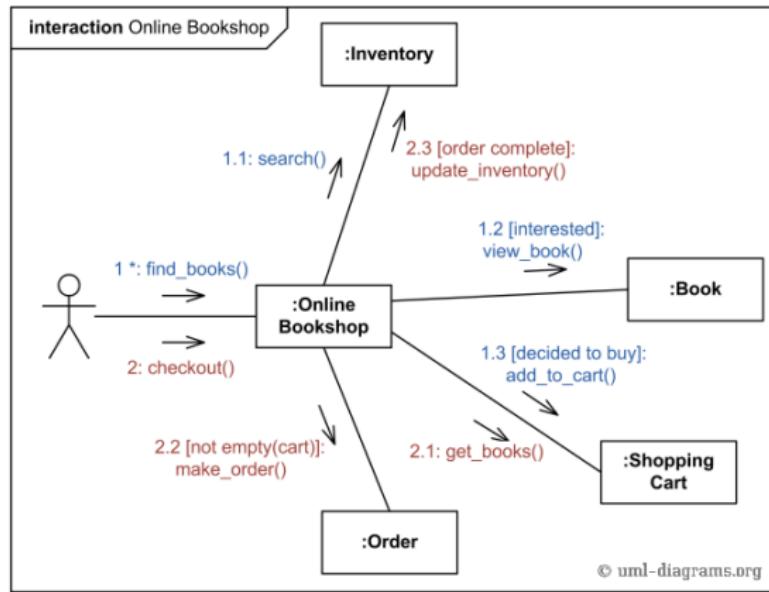
# Коммуникационные диаграммы

- ▶ Применяются для визуализации взаимодействия между объектами
  - ▶ Более легковесный аналог диаграмм последовательностей
  - ▶ Тоже отображают один сценарий взаимодействия



© М. Фаулер, UML. Основы

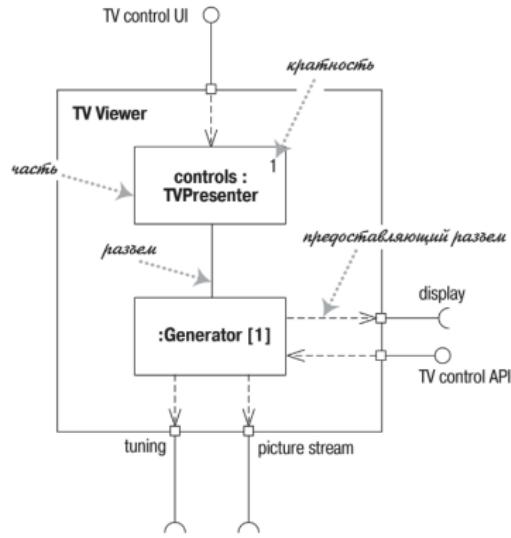
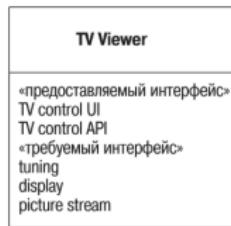
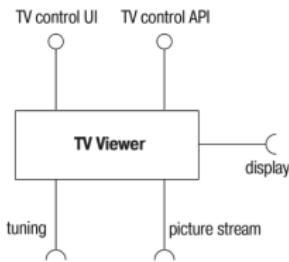
# Коммуникационные диаграммы, пример



© <http://www.uml-diagrams.org/>

# Диаграммы составных структур

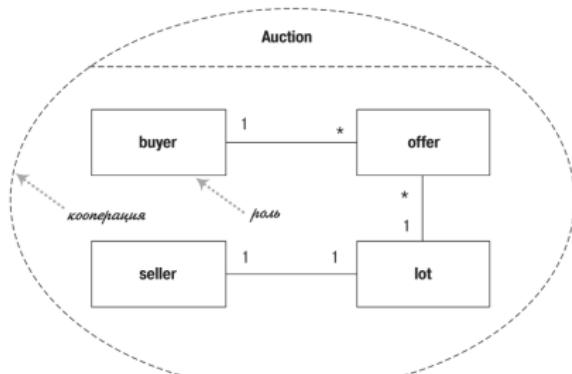
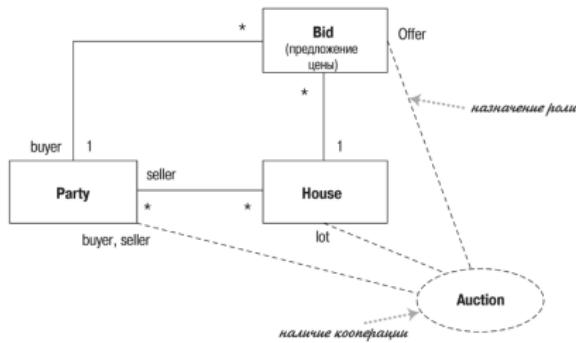
- ▶ По сути, продвинутые диаграммы компонентов
- ▶ Внутри компоненты не другие компоненты, а части (роли)



© M. Фаулер, UML. Основы

# Диаграммы коопераций

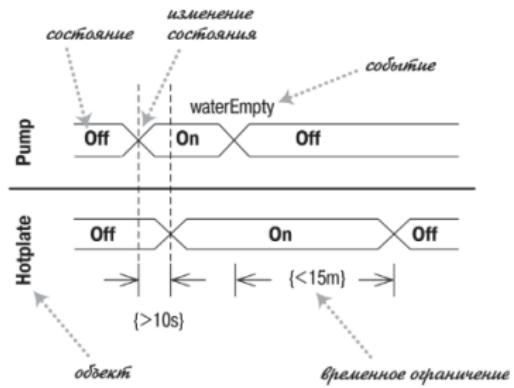
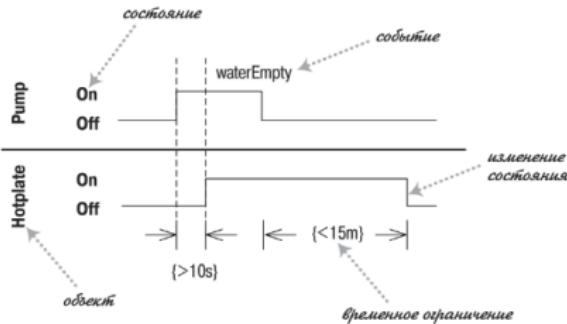
- ▶ Показывают взаимодействие между объектами (ролями) в рамках одного сценария использования



© M. Фаулер, UML. Основы

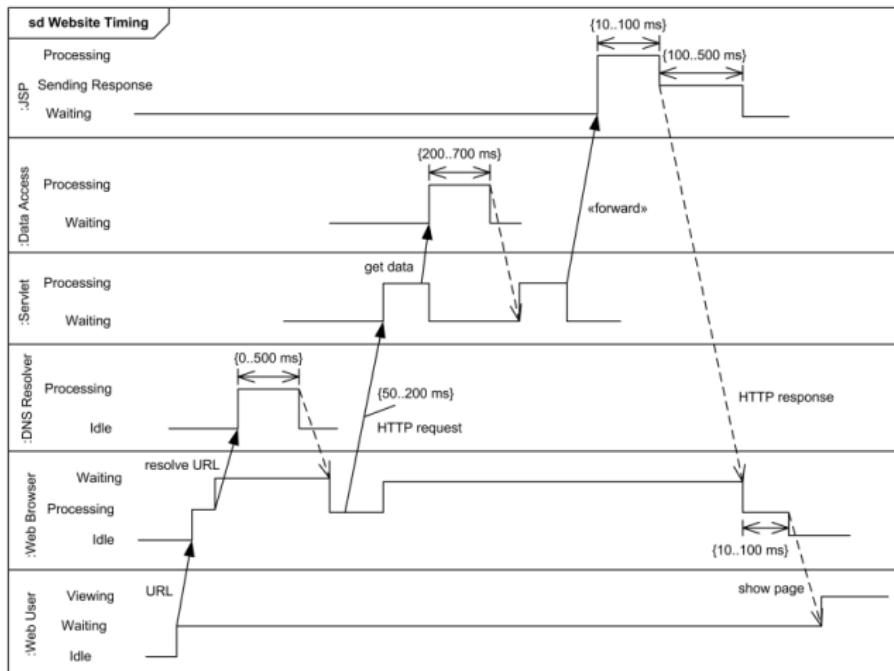
# Временные диаграммы

- ▶ Для моделирования временных ограничений в системах реального времени



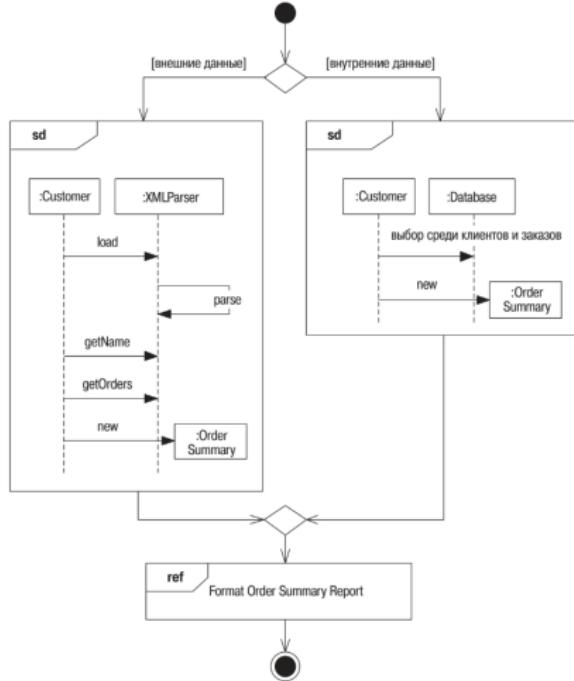
© M. Фаулер, UML. Основы

# Временная диаграмма, пример



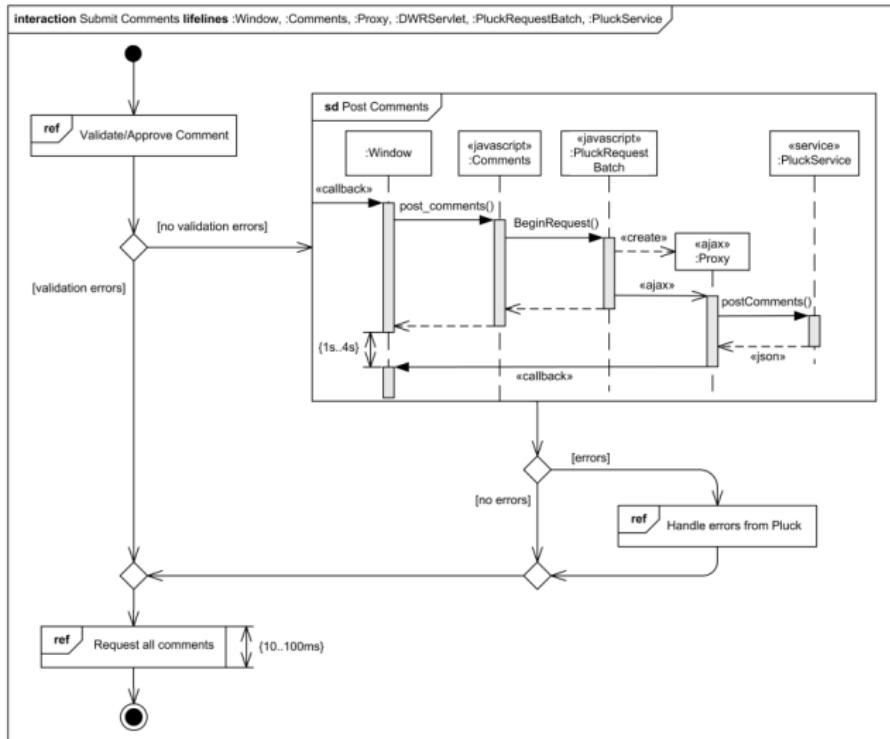
# Диаграммы обзора взаимодействия

- ▶ Диаграммы активностей + диаграммы последовательностей
- ▶ Применяются при наличии взаимодействия со сложной логикой, когда фреймы неудобны



© М. Фаулер, UML. Основы

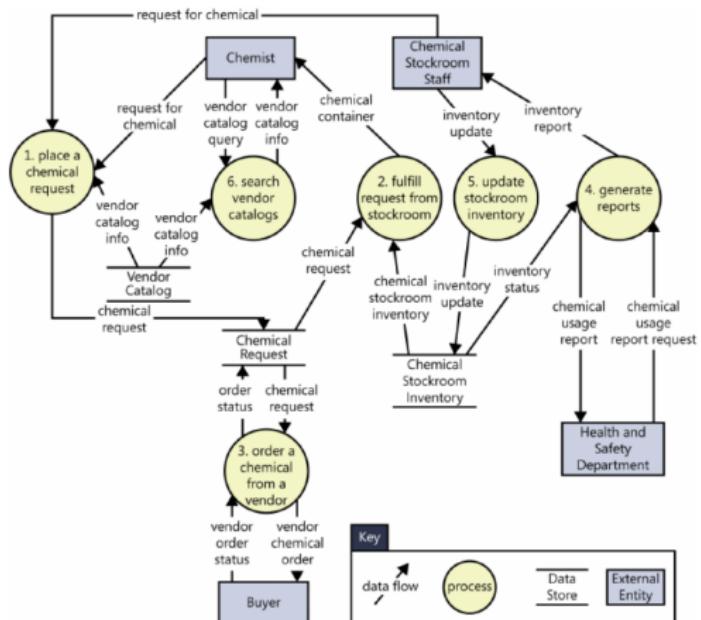
# Диаграмма обзора взаимодействия, пример



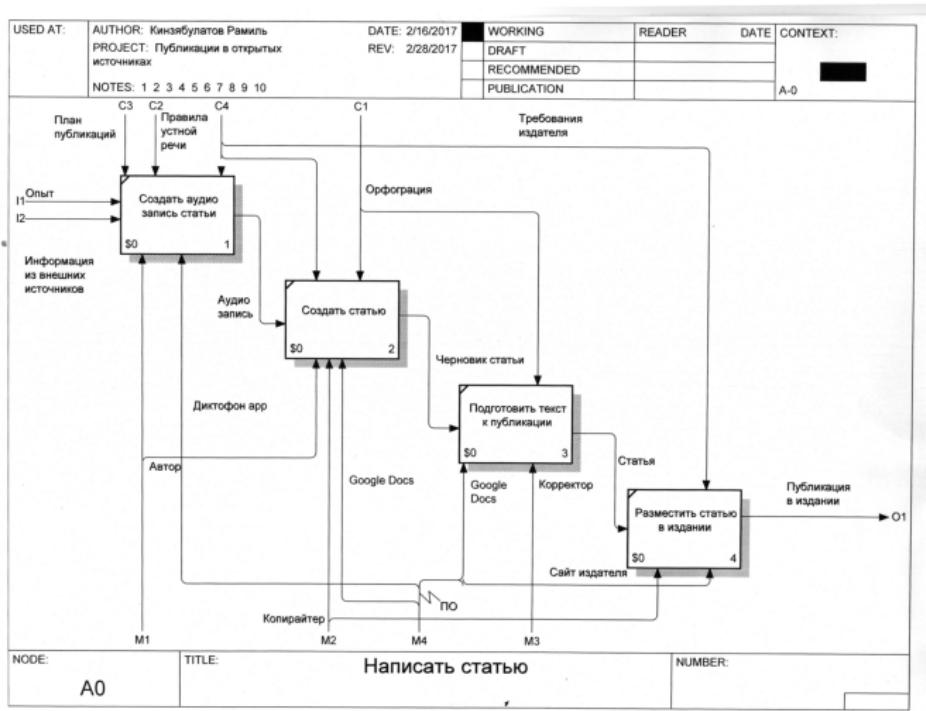
# Диаграммы потоков данных

DFD

- ▶ Показывают обмен данными в системе
- ▶ Внешние сущности, процессы внутри системы, потоки данных



# Диаграммы IDEF0

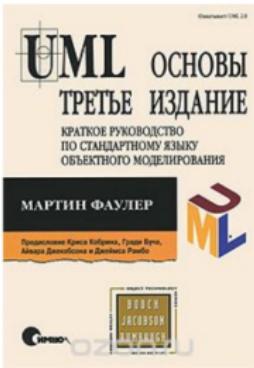


© <https://habrahabr.ru/post/322832/>

# Примеры CASE-инструментов

- ▶ “Рисовалки”
  - ▶ Visio
  - ▶ Dia
  - ▶ SmartDraw
  - ▶ LucidChart
  - ▶ Creately
  - ▶ Diagrams.Net
- ▶ Полноценные CASE-системы
  - ▶ Enterprise Architect
  - ▶ Rational Software Architect
  - ▶ MagicDraw
  - ▶ Visual Paradigm
  - ▶ GenMyModel
- ▶ Забавные штуки
  - ▶ <https://www.websequencediagrams.com/>
  - ▶ <http://yuml.me/>
  - ▶ <http://plantuml.com/>

# Книжка



М. Фаулер, UML. Основы. Краткое руководство по стандартному языку объектного моделирования. СПб., Символ-Плюс, 2011. 192 С.