http://mauriziostorani.wordpress.com/2008/10/09/differences-and-similarities-between-design-patterns-and-architectural-styles/

Курс: «Математические модели комплексов программ»

Модуль: **«Архитектура программных систем»**

Лекция 8 (Князьков К.В.)

Что такое архитектурный стиль?

‘A set of **design rules** that **identify** the **kinds** of **components** and **connectors** that may be used to compose a system or subsystem, together with local or global **constraints** on the way the **composition** is done’

— Shaw & Clements, 1996

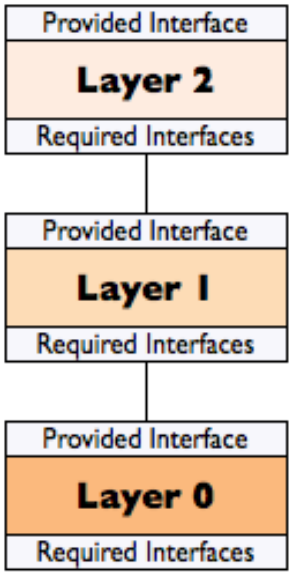
Структурные стили

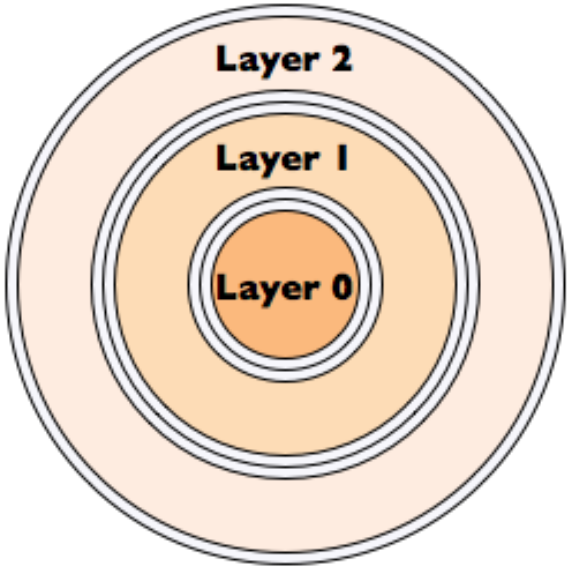
Monolitic – Монолитная

- Архитектуру невозможно разбить на отдельные функциональные компоненты
- Бизнес логика, пользовательский интерфейс и обработка данных – все содержится в одном слое
- Пример: Microsoft Word (*http://msdn.microsoft.com/en-us/library/aa480455.aspx*)

Layered – Слоеная архитектура

- Организация связанной функциональности в отдельные слои, которые сгруппированы вертикально один над другим
- Каждый слой агрегирует функции нижестоящего слоя
- Каждый элемент определенного слоя может взаимодействовать с элементами своего слоя и нижестоящего

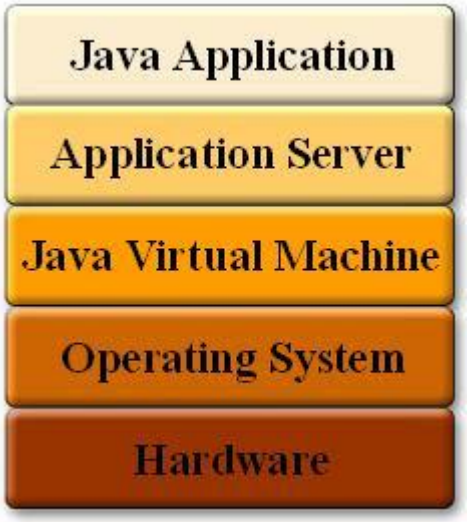


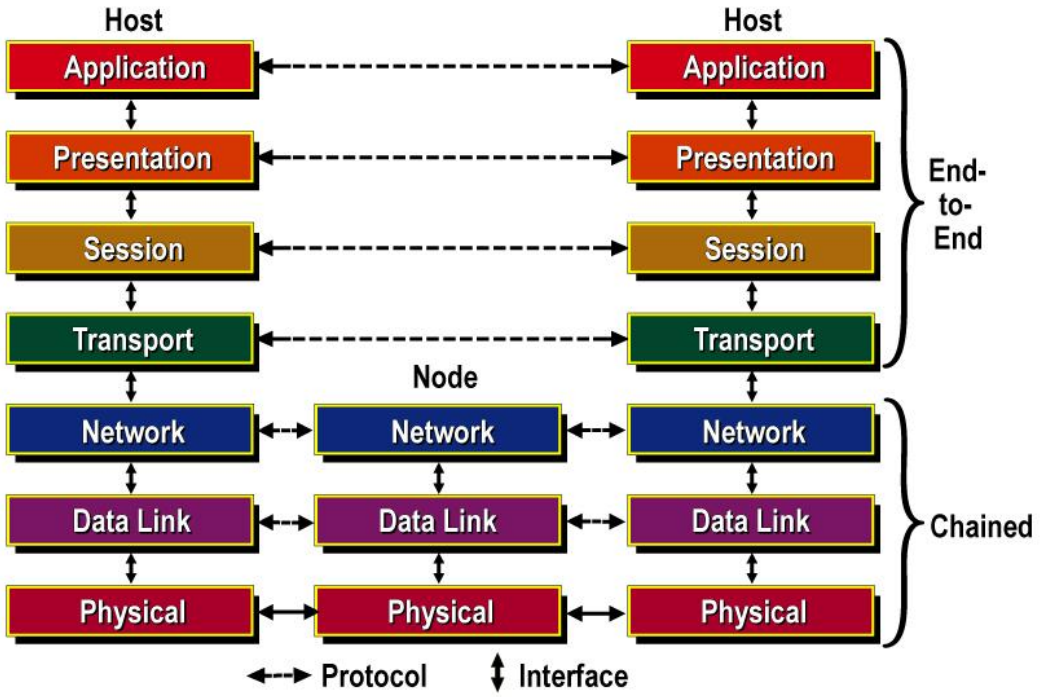


http://www.thomasalspaugh.org/pub/fnd/architecture.html#Layers

Примеры

- Виртуальные машины
- Операционная система
- Стек протоколов OSI



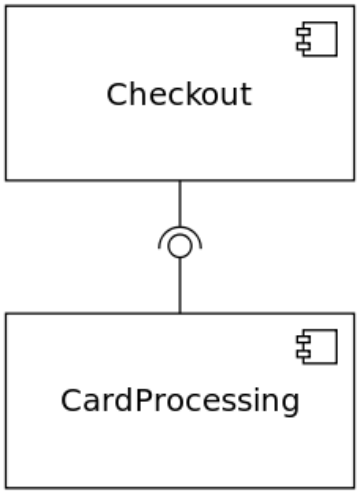


3-TIER – ТРЕХУРОВНЕВАЯ АРХИТЕКТУРА

- Первый уровень: уровень представления (пользовательский интерфейс, интерфейс веб-сервисов)
- Второй уровень: бизнес логика (основная логика приложения)
- Третий уровень: уровень хранения данных (базы данных)

Component-based – Компонентный стиль

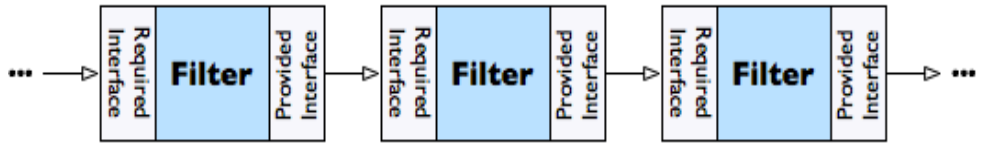
- Согласно стилю архитектура декомпозируется на набор функциональных компонентов, которые обладают четко определенным интерфейсом взаимодействия
- Платформа обеспечивает компоненты сервисными механизмами запуска, обработки ошибок и т.д.
- Компоненты
  - Повторно используемы
  - Заменяемы
  - Инкапсулированы
  - Независимы

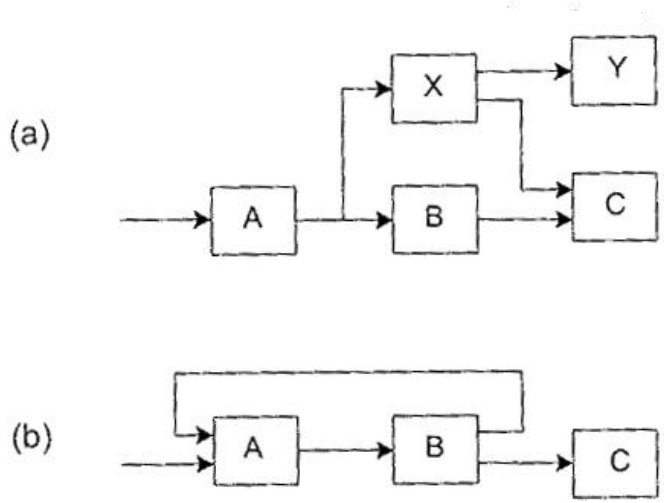


- Паттерны
  - DI

Pipes and filters – Каналы и фильтры

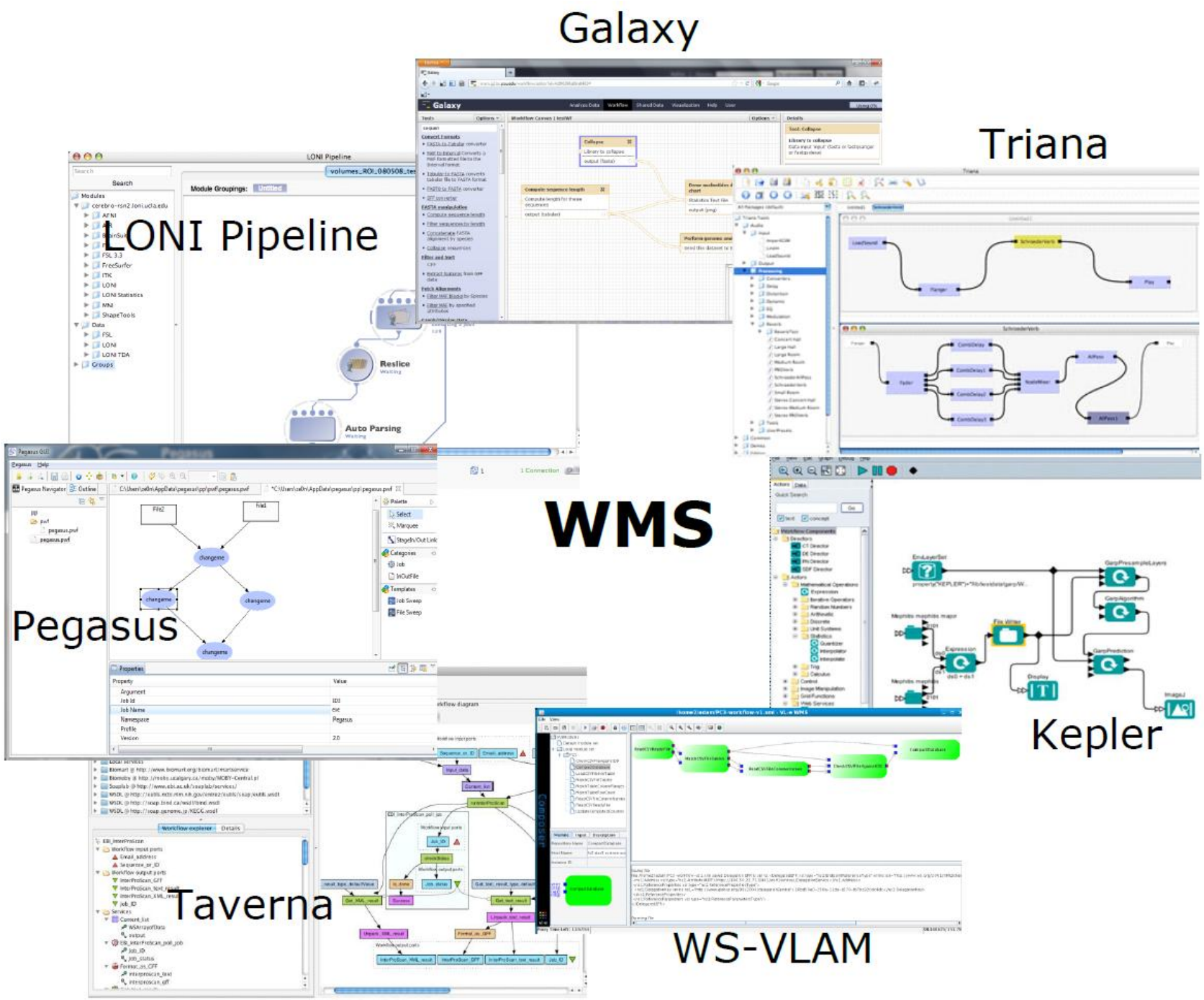
- Архитектура, основанная на потоках данных, представляет систему обработки данных, состоящую из отдельных модулей обработки, соединенных потоками.
- Можно разделить на: пакетную обработку (batch), непрерывную обработку
- Для моделирования хорошо подходят диаграммы потоков данных (DFD)





Примеры

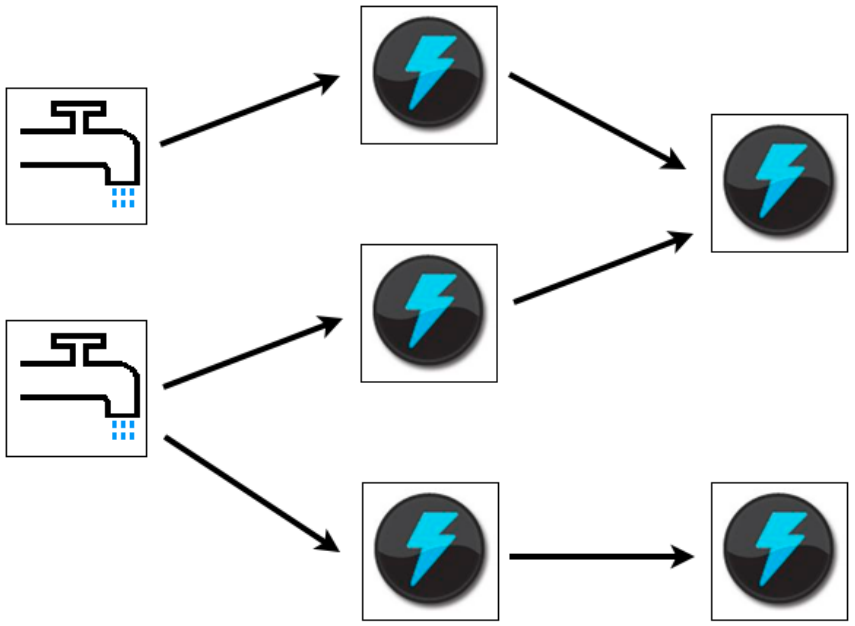
- Unix shell
  - ps | sort | less
- Обработка видео
- Workflow Management Systems
- Real-Time Big Data (например, Apache Storm)
- Парадигма реактивного программирования


Galaxy
Triana
LONI Pipeline
WMS
Pegasus
Kepler
Taverna
WS-VLAM

Пример «Apache Storm»

https://storm.incubator.apache.org/

- Система потоковой обработки неограниченных объемов данных
- Любой язык программирования

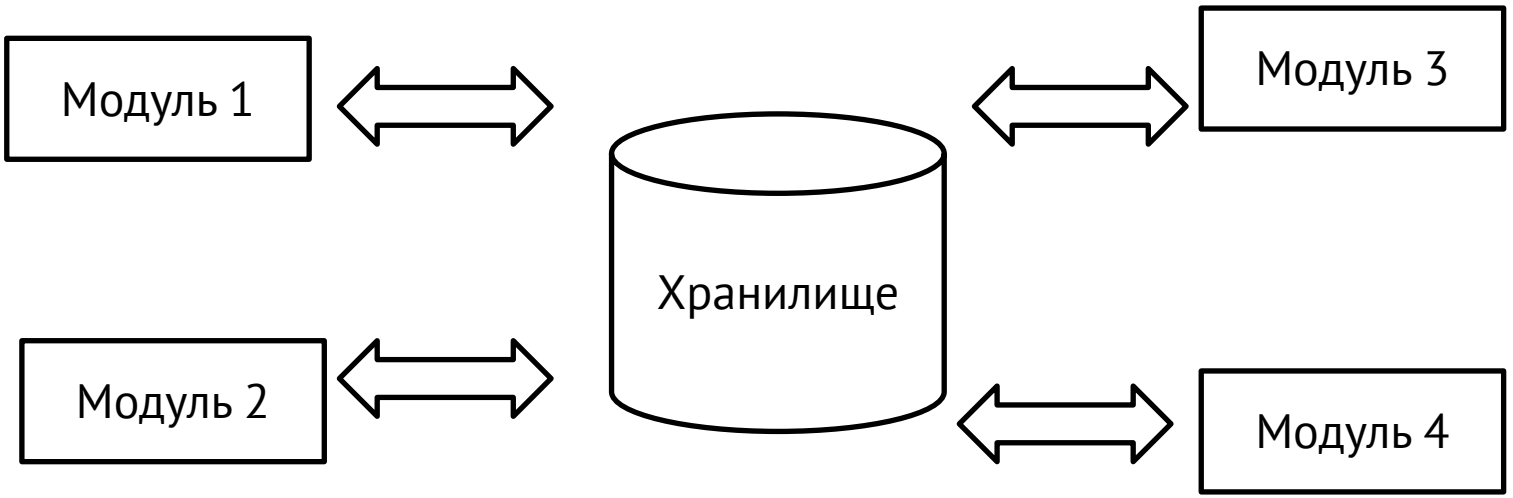


http://www.slideshare.net/ptgoetz/storm-hadoop-summit2014

Архитектурные стили на основе общей памяти

Data centric– архитектура вокруг централизованного хранилища

- Основная роль в системе лежит на хранилище данных

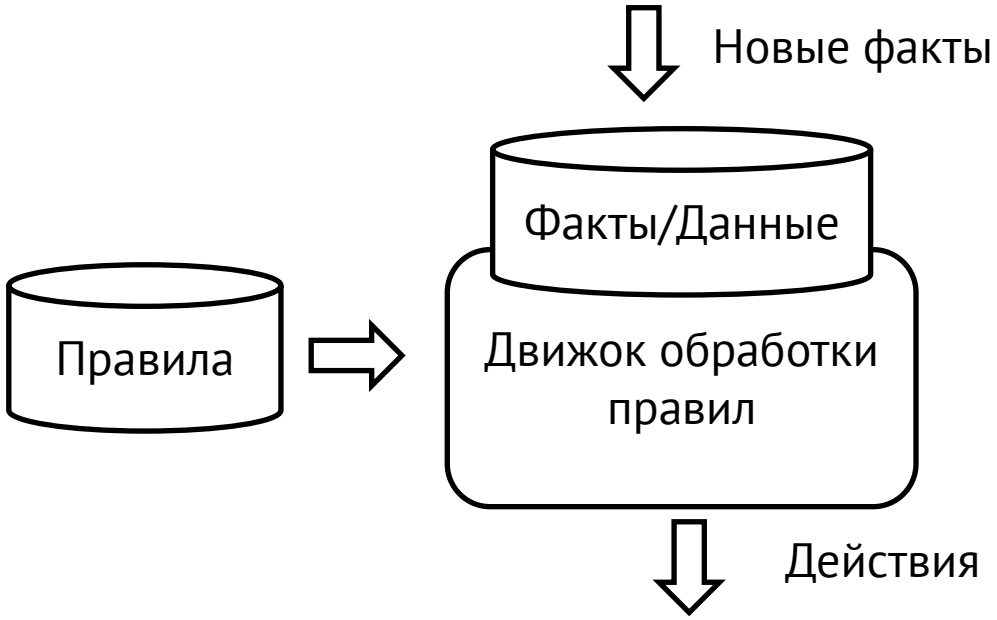


Примеры

- Системы Big Data

Rule-based– Системы, основанные на правилах

- Пользователи выражают требования или знания о системе в виде набора декларативных правил
- Два типа систем обработки правил: прямой вывод (от фактов), обратный вывод (от целей)



Примеры

- Системы исполнения бизнес правил (Business Rule Engine): Jboss Drools

```
rule “discount”
when
       c: Customer(married == true || age > 25)
Then
       c.addDiscount(10);
end
```

- Экспертные системы: CLIPS
- Языки программирования: Prolog

Blackboard– Доска или рабочая область

- Компоненты трех основных типов:
  - «Рабочая область» – репозиторий, содержащий входные данные и частичные решения;
  - Источники знаний – независимые модули, обладающие определенной экспертизой и знаниями для решения задач;
  - Контроллер – координирует источники знаний для решения определенной задачи;

Примеры

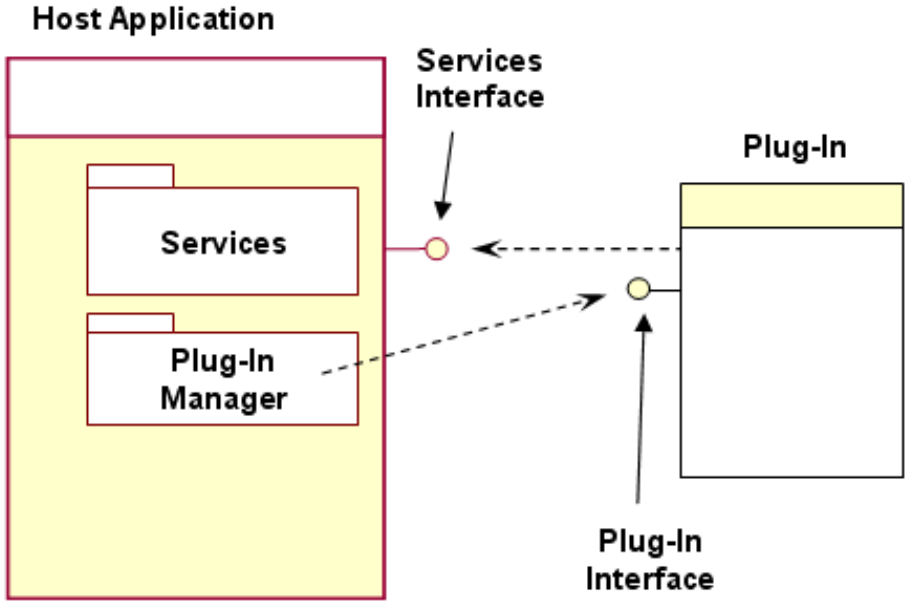
- Распознавание речи (Hearsay II)
- Идентификация и отслеживание транспортных средств
- Google Docs

Архитектурные стили на основе обмена сообщениями

Адаптивные системы

PLUGINS

- Плаги́н (англ. plug-in, от plug in «подключать») — независимо компилируемый программный модуль, динамически подключаемый к основной программе и предназначенный для расширения и/или использования её возможностей. Плагины обычно выполняются в виде разделяемых библиотек.



DSL

- **Предметно-ориентированный язык** (англ. Domain Specific language, **DSL**) — язык программирования, специализированный для конкретной области применения (в противоположность языку общего назначения, применимому к широкому спектру областей и не учитывающему особенности конкретных сфер знаний).

- Парадигма **языково-ориентированного программирования** (Language Oriented Programming), заключающаяся в разбиении процесса разработки программного обеспечения на стадии разработки предметно-ориентированных языков (DSL) и описания собственно решения задачи с их использованием.

Реализация DSL

- External DSL – язык, для которого парсер, интерпретатор или компилятор создаются на основном языке
- Особенности
  - Синтаксис создается с нуля и не ограничен ничем
  - Все инструменты для разработки необходимо создавать отдельно
- Библиотеки для создания Ext. DSL
  - ANTLR
  - Bison
  - …

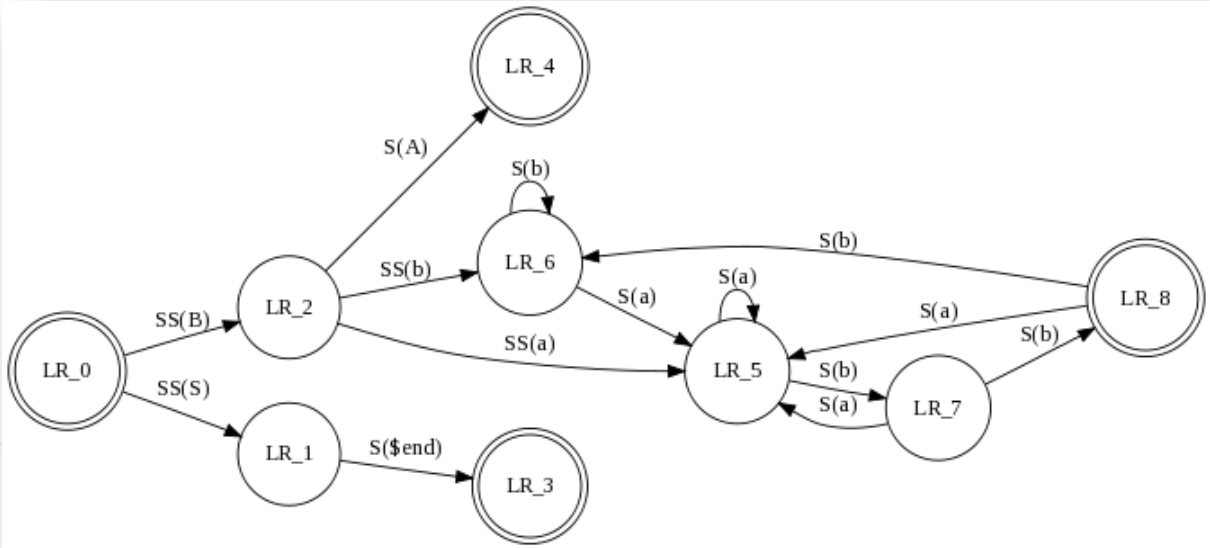
- Internal DSL – предметно-ориентированный язык, создаваемый на основе базового
- Особенности
  - Ограничение основного языка
  - Можно использовать существующие инструменты
- Примеры языков, на которых удобно создавать Int. DSL
  - Ruby
  - Lisp
  - …

http://www.maxkir.com/sd/languageWorkbenches.html#10

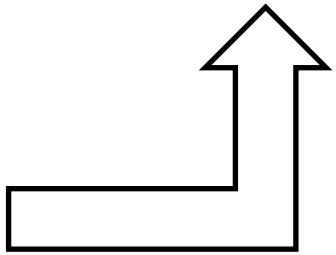
Примеры

- Unix shell
- regular expressions
- awk
- Gnuplot
- GraphVis
- Interface Description Language
- HTML, css
- make
- …

Примеры «GraphViz»

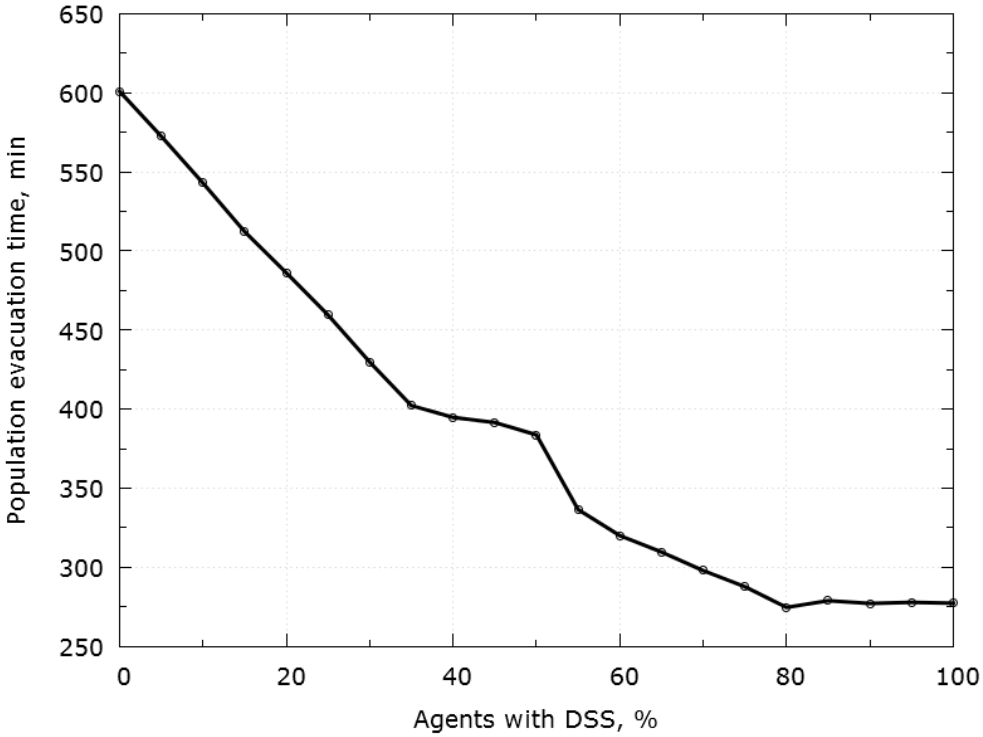



```
digraph finite_state_machine {
    rankdir=LR;
    size="8,5"
    node [shape = doublecircle]; LR_0 LR_3 LR_4 LR_8;
    node [shape = circle];
    LR_0 -> LR_2 [ label = "SS(B)" ];
    LR_0 -> LR_1 [ label = "SS(S)" ];
    LR_1 -> LR_3 [ label = "S($end)" ];
    LR_2 -> LR_6 [ label = "SS(b)" ];
    LR_2 -> LR_5 [ label = "SS(a)" ];
    LR_2 -> LR_4 [ label = "S(A)" ];
    LR_5 -> LR_7 [ label = "S(b)" ];
    LR_5 -> LR_5 [ label = "S(a)" ];
    LR_6 -> LR_6 [ label = "S(b)" ];
    LR_6 -> LR_5 [ label = "S(a)" ];
    LR_7 -> LR_8 [ label = "S(b)" ];
    LR_7 -> LR_5 [ label = "S(a)" ];
    LR_8 -> LR_6 [ label = "S(b)" ];
    LR_8 -> LR_5 [ label = "S(a)" ];
}
```



Примеры «GnuPlot»

```
set terminal pngcairo size 800,600 enhanced font 'Verdana,14'

set autoscale
set xtic auto
set ytic auto

#set ytic 70,10,190
#set xtic 0,10,100

#set yrange [70:190]
#set xrange [0:100]

set mxtics 2
set mytics 2

set xlabel "Agents with DSS, %"
set ylabel "Population evacuation time, min"

set grid ytics lt 0 lw 1 lc rgb "#bbbbbb"
set grid xtics lt 0 lw 1 lc rgb "#bbbbbb"

set datafile separator ";"

set output 'percents.xy.png'
plot "xy.csv" using 1:2 with lines lw 3 lc "black" title "", \
     "xy.csv" using 1:2 with points pt 6 lc "black" title ""

end
```



Примеры «IDL»

Google ProtoBuf

```
//polyline.proto

message Point {
  required int32 x = 1;
  required int32 y = 2;
  optional string label = 3;
}

message Line {
  required Point start = 1;
  required Point end = 2;
  optional string label = 3;
}

message Polyline {
  repeated Point point = 1;
  optional string label = 2;
}
```

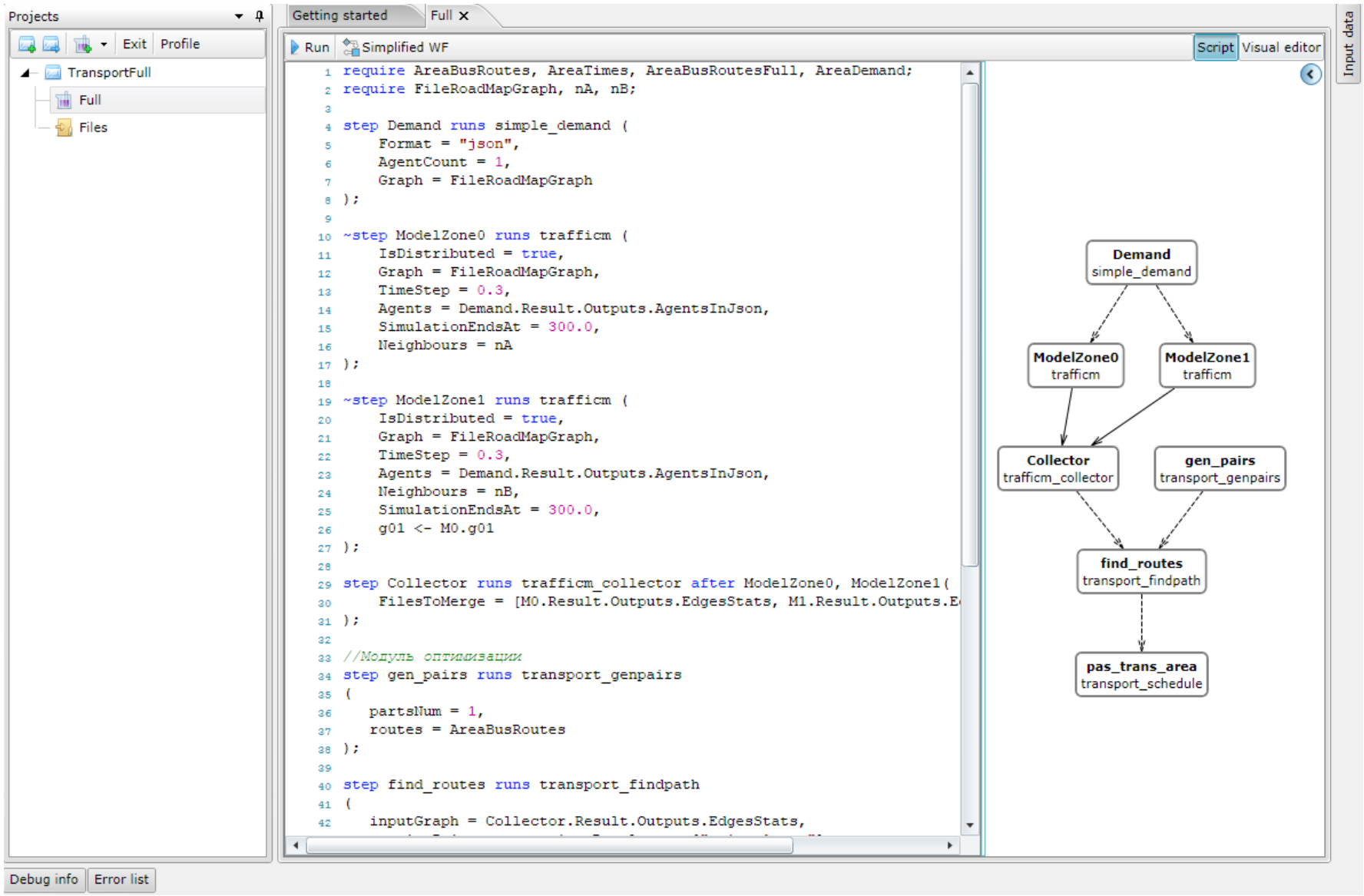
Apache Thrift

```
service Calculator extends shared.SharedService {

  /**
   * A method definition looks like C code. It has a return type, arguments,
   * and optionally a list of exceptions that it may throw. Note that argument
   * lists and exception lists are specified using the exact same syntax as
   * field lists in struct or exception definitions.
   */

   void ping(),

   i32 add(1:i32 num1, 2:i32 num2),

   i32 calculate(1:i32 logid, 2:Work w) throws (1:InvalidOperation ouch),

   /**
    * This method has a oneway modifier. That means the client only makes
    * a request and does not listen for any response at all. Oneway methods
    * must be void.
    */
   oneway void zip()

}
```

Примеры «EasyPackage»

```
name "BSM"
display_as "Baltic Sea Model"
vendor "BSM"
url "http://bcc.ru"
license "GPLv3"
description "BSM calculates prediction of the sea level"

inputs {

    public param {
        name "useSWAN"
        display "Use SWAN"
        type bool
        default false
    }

    public param {
        name "startCalcDate"
        required
        display "Start Calculation Date"
        type string
    }

    public param {
        name "useBSH"
        display "Use BSH"
        type bool
        default false
    }

    public param {
        name "useExternalProject"
        display "Use External Project"
        type bool
        default false
    }

    public param {
        name "useOldProject"
        display "Use old project?"
        type bool
        default false
    }
```

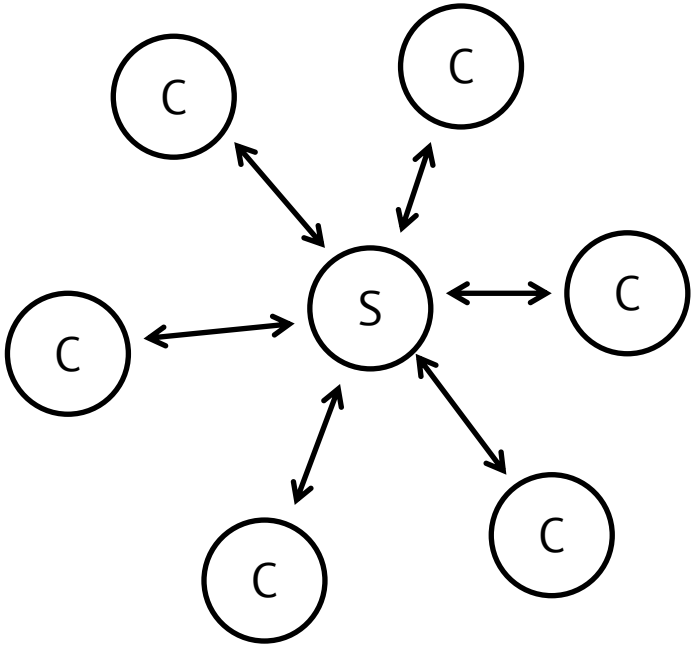
Примеры «EasyFlow»



Распределенные системы

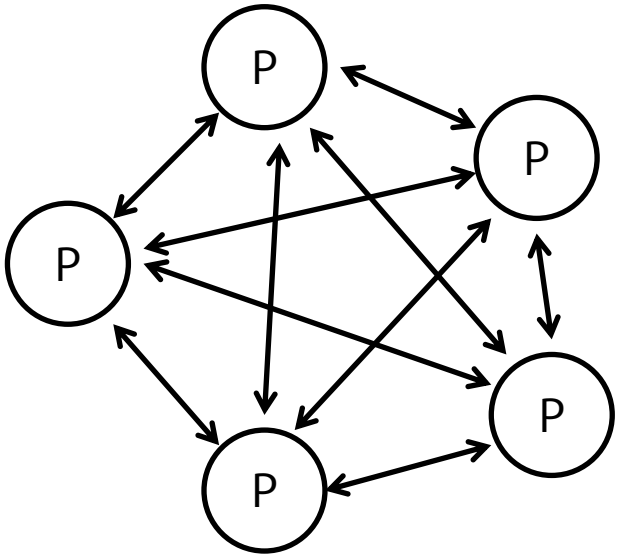
Client-server – клиент-сервер

- Разделение архитектуры на поставщиков услуг и потребителей услуг (два уровня).
- Клиент инициализирует взаимодействие за счет посылки запроса серверу. Сервер выполняет действия и возвращает ответ.
- Клиенты не взаимодействуют между собой



Peer-to-peer – пиринговая система

- Одноранговая сеть из равноправных элементов
- Элементы не гарантируют присутствие
- Элементы сочетают как клиентские функции, так и серверные
- Элементы содержат административные функции



Примеры

- Файлообменные сети
- Пиринговые сети распределенных вычислений
- Пиринговые финансовые сети (bitcoin)

SOA

- **Сервис-ориентированная архитектура** (SOA, англ. service-oriented architecture) — модульный подход к разработке программного обеспечения, основанный на использовании распределённых, слабо связанных заменяемых компонентов, оснащённых стандартизированными интерфейсами для взаимодействия по стандартизированным протоколам.

Веб-сервисы

Веб-сервис – программная система созданная для поддержки интероперабельного межкомпьютерного взаимодействия через сеть. *W3C*

Для чего нужны веб-сервисы:

- Стандартизованный способ взаимодействия между приложениями в Интернет
- Связь существующих неоднородных приложений (Java, .NET, Python…)
- Связь приложений на гетерогенных платформах (Windows, Unix…)
- Возможность переиспользования существующих компонентов
- Возможность интеграции ПО, разработанного разными командами

Ориентация на веб-сервисы

- **Четкие границы**
  - Пересечение границ– четко определено
  - Пересечение границ стоит ресурсов
- **Сервисы самостоятельны**
  - Мы должны понимать что используемый нами сервис будет развиваться и у нас нет контроля над этим
  - Сервисы управляются и разрабатываются независимо
  - Сервис, который мы используем может быть недоступен!
- **Сервисы предоставляют схему и контракт, но не код**
  - Сервисы взаимодействуют по контрактам, которые не меняются.
  - Сервисы предоставляют только контракт, реализация может меняться.
- **Совместимость сервисов определяется политикой**
  - Безопасность, гарантированная доставка , и прочее определяется политикой.
  - Требования и возможности сервиса также предоставляются политикой.

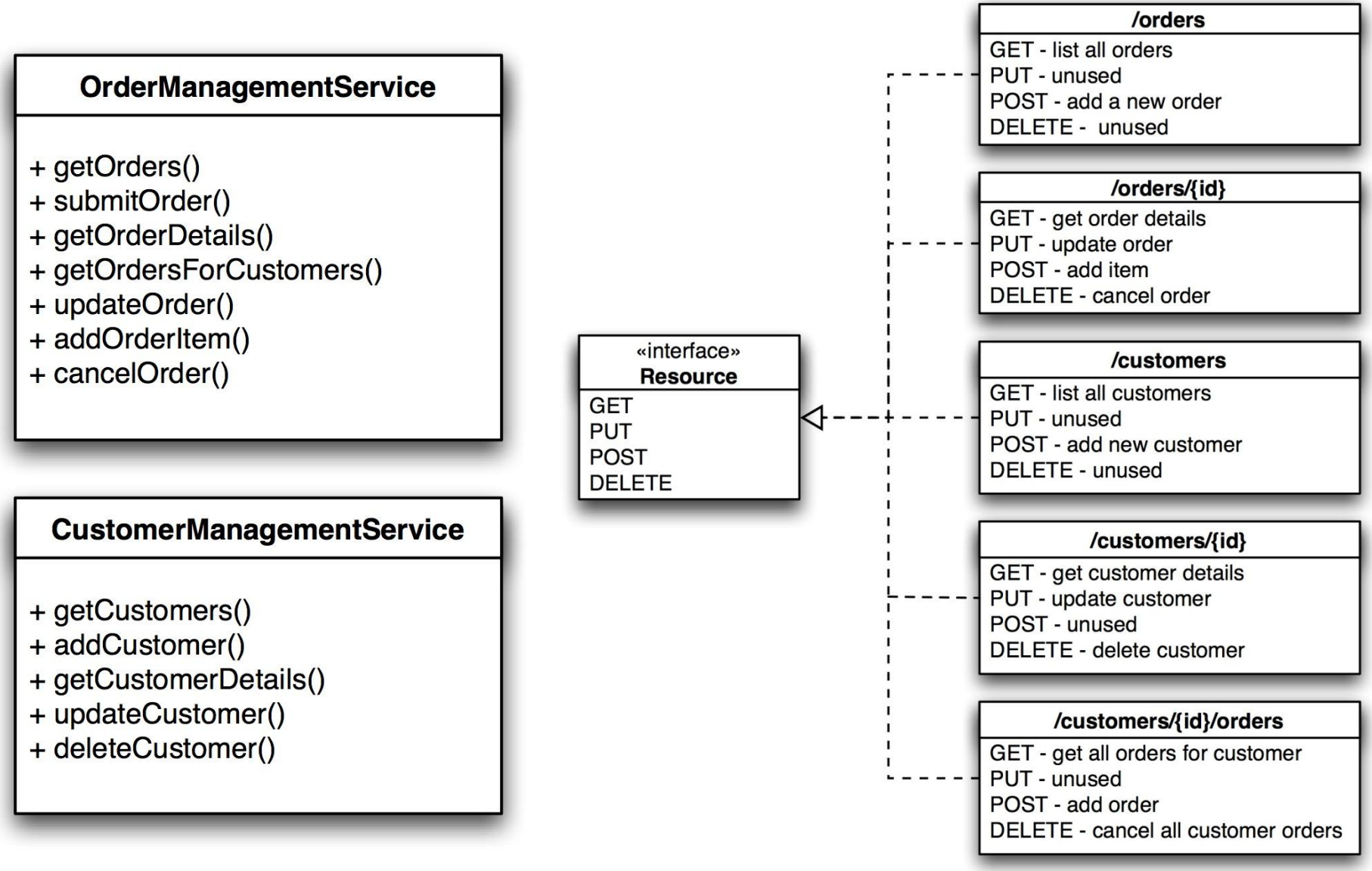
Representational state transfer

- Стиль предложен в диссертации Роя Филдинга (один из разработчиков HTTP) в 2000 году
- Можно перевести как «представление данных в удобном для клиента формате»
  [1]
- В основе подхода – ресурсы с идентификаторами
- API упрощен и создается согласно модели **CRUD** (create-read-update-delete)
  - POST — create (создать)
  - GET — read (прочитать, получить)
  - PUT — update (изменить, обновить или создать, если не существует, иногда применяется и в таком варианте)
  - DELETE — delete (удалить)

[1] http://habrahabr.ru/post/108993/

Модели проектирования веб-сервисов

	RPC *Remote Procedure Call*	REST *Representational State Transfer*
Клиентский контекст (состояние)	Хранится между запросами	Не хранится между запросами
Основной элемент	Метод	Ресурс
Протокол	SOAP	Использует возможности HTTP
Операции	Большое количество методов	Методы HTTP, реализующие CRUD
Адрес	Один или небольшое количество	Для каждого ресурса свой


OrderManagementService
+ getOrders()
+ submitOrder()
+ getOrderDetails()
+ getOrdersForCustomers()
+ updateOrder()
+ addOrderItem()
+ cancelOrder()
CustomerManagementService
+ getCustomers()
+ addCustomer()
+ getCustomerDetails()
+ updateCustomer()
+ deleteCustomer()
«interface»
Resource
GET
PUT
POST
DELETE
/orders
GET - list all orders
PUT - unused
POST - add a new order
DELETE - unused
/orders/{id}
GET - get order details
PUT - update order
POST - add item
DELETE - cancel order
/customers
GET - list all customers
PUT - unused
POST - add new customer
DELETE - unused
/customers/{id}
GET - get customer details
PUT - update customer
POST - unused
DELETE - delete customer
/customers/{id}/orders
GET - get all orders for customer
PUT - unused
POST - add order
DELETE - cancel all customer orders

Источники

- **Книги**
  - Мартин Фаулер, «Предметно-ориентированные языки программирования», Вильямс, 2011.
  - Э. Дж. Брауде. Технология разработки программного обеспечения. Питер, 2004.

- **Статьи и презентации**
  - Garlan, D., & Shaw, M. (1994). An introduction to software architecture.
  - Microsoft Application Architecture Guide, 2nd Edition, Chapter 3: Architectural Patterns and Styles http://msdn.microsoft.com/en-us/library/ee658117.aspx
  - Software architecture styles and patterns http://en.wikipedia.org/wiki/Software_architecture_styles_and_patterns
  - Henry Muccini, «Software Architecture: Styles», http://www.slideshare.net/henry.muccini/software-architecture-styles
  - David S. Rosenblum, «Advanced Analysis and Design: Architectural Styles» http://www.ccs.neu.edu/home/lieber/courses/csg110/sp08/lectures/april7/rosenblum-implicit-invoc.pdf

- **Дополнительно**
  - Введение в REST http://www.infoq.com/articles/rest-introduction
  - Построение хорошего REST API http://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf