Technologies Internet
Web Services

Open Source Frameworks (OSF)
Master of Science in Engineering (MSE)

Olivier Liechti
olivier.liechti@heig-vd.ch

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

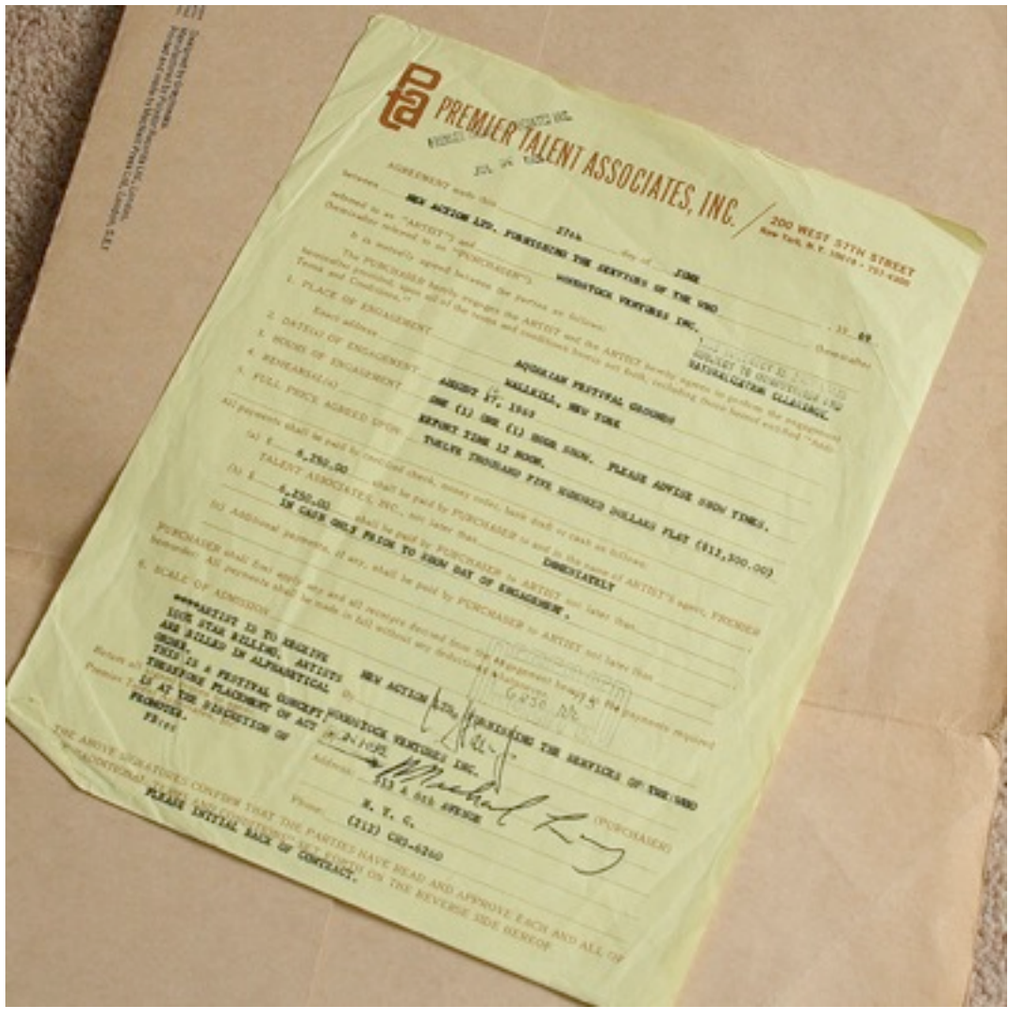
Agenda

- **The Notion of (Web) Service**
  - A generic concept, many implementations
  - **Describing** services, **discovering** services, **interacting** with services
- **“Big” Web Services (WS-*)**
  - Key Concepts
  - Big Web Services with Java EE (JAX-WS)
- **RESTful Web Services**
  - Key Concepts
  - RESTful APIs with Java EE (JAX-RS)

The Notion of (Web) Service

The Notion of Service

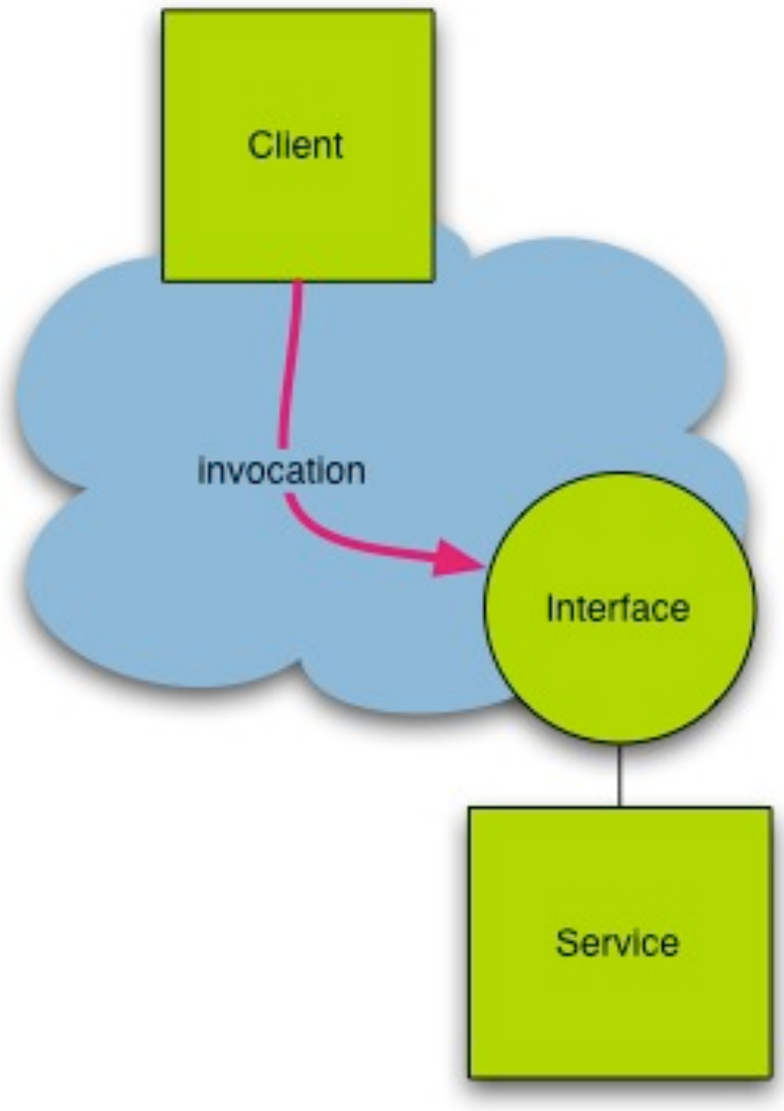
- The abstraction of service is **generic**:
  - It is key in the design of many systems.
  - “Small” systems (applications)
  - “Big” systems (information systems)
  - “Distributed” systems
- But what is a “**service**”?
  - It is “something” that provides access to some functionality.
  - The functionality should be described in an interface, which defines a contract between the client and the service provider.



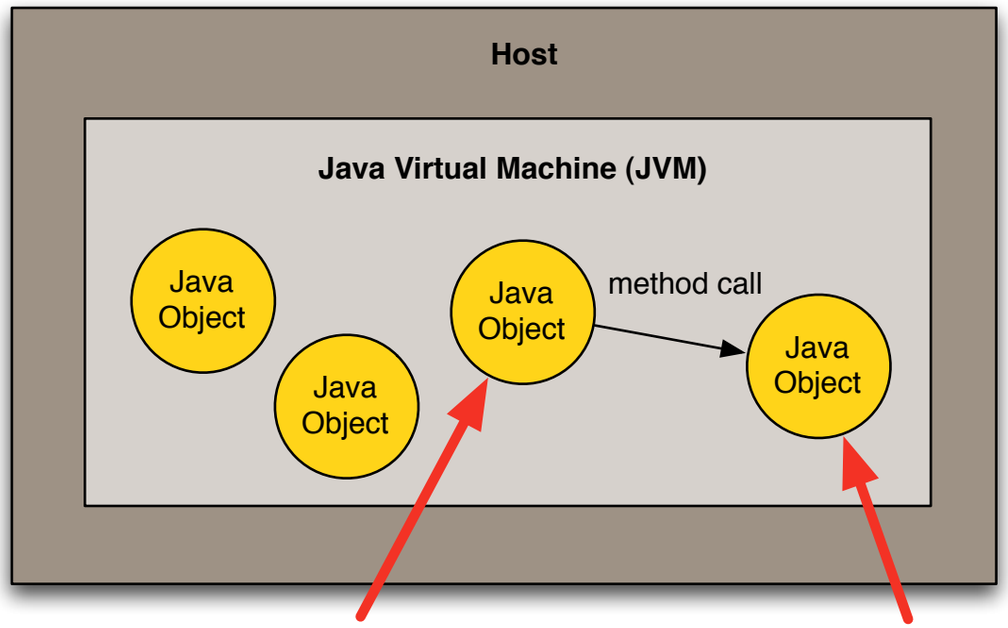
http://flickr.com/photos/ian-s/2655970731/sizes/l/#cc_license

How Can I Implement a Service?

- **Lots of different technologies** can be used to implement services.
- Just think about a basic **Java application**:
  - A class that implements an interface is an example of a service!
  - In that case, service providers and clients live in the same VM - service invocation ha
- Now, think about **distributed Java** applications:
  - RMI makes it possible to invoke a service running in a different JVM.
  - This is an example of Remove Procedure Call.



A Service inside a Single JVM



The Client

```
public class ClockClient

  private ClockService service;

...

  public String showTime() {
    System.out.println("It is " +
      service.getTime());
  }
}
```

ClockClient.java

The Service

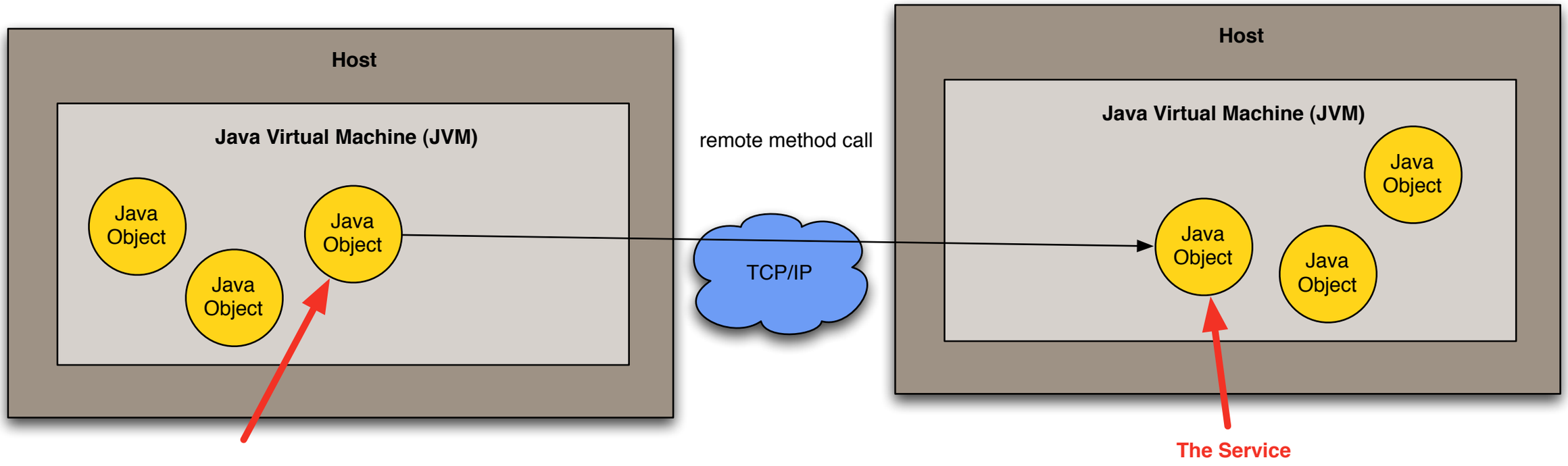
```
public interface ClockService {

  public String getTime();
}
```

ClockService.java

```
public class ClockServiceImpl
  implements ClockService {

  public String getTime() {
    return new java.util.Date();
  }
}
```

ClockServiceImpl.java

A Service in a Remote JVM




```
public class ClockClient

}
```

générer du XML (SOAP) ?

ouvrir un socket?

parser du XML (réponse SOAP)

ClockClient.java

```
public interface ClockService {

  public String getTime();
}
```

ClockService.java

```
public class ClockServiceImp
  implements ClockService {

}
```

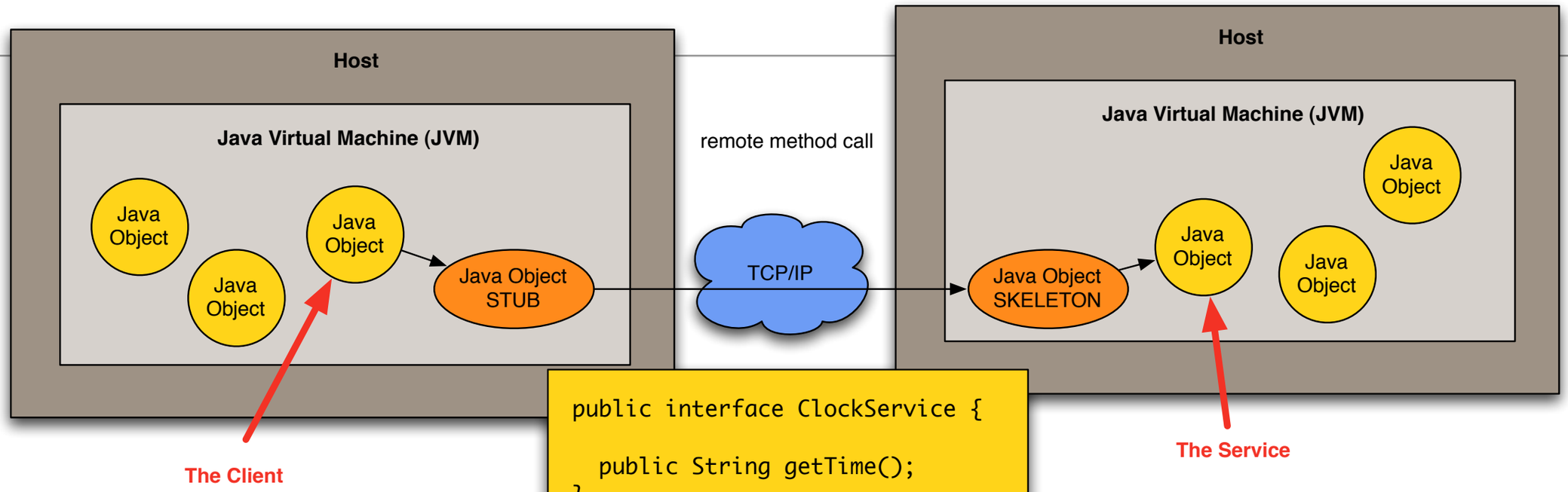
générer du XML (SOAP) ?

écouter sur un socket?

parser du XML (SOAP)

ClockServiceImpl.java

A Service in a Remote JVM




```
public interface ClockService {

  public String getTime();
}
```

ClockService.java

```
public class ClockClient

  private ClockService service;

...

  public String showTime() {
    System.out.println("It is " +
      service.getTime());
  }
}
```

ClockClient.java

```
public class ClockServiceStub
  implements ClockService {

  public String getTime() {
    ... bla bla... SOCKET ...
    XML... bla bla bla... SOAP ...
    bla bla OUTPUTSTREAM... bla bla
    INPUTSTREAM....

    return time;
}
```

ClockServiceStub.java

```
public class ClockServiceSkeleton {

  private ClockService realService;

  private void setup() { ... }
  private void listen() { ... }
  private void processCall {
    realService.getTime();
  }
}
```

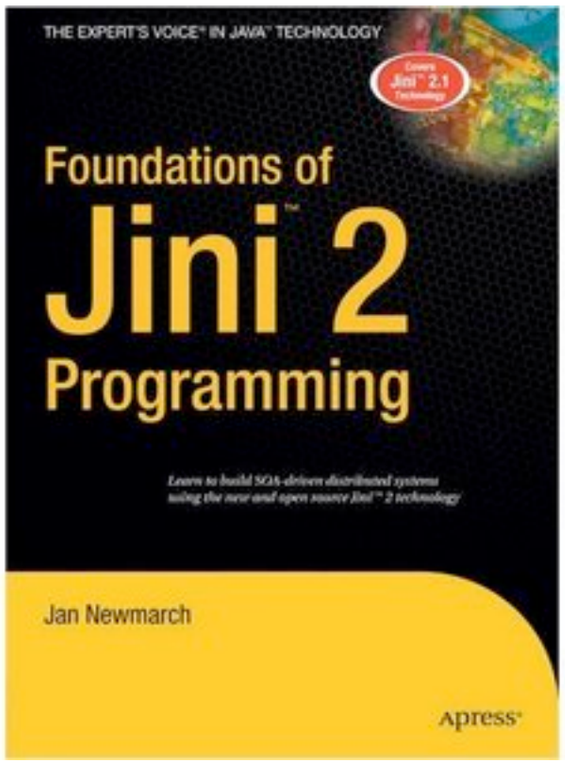
ClockServiceSkeleton.java

```
public class ClockServiceImpl
  implements ClockService {

  public String getTime() {
    return new java.util.Date();
  }
}
```

ClockServiceImpl.java

Generated automatically by special tools!

How Can I Implement a Service?

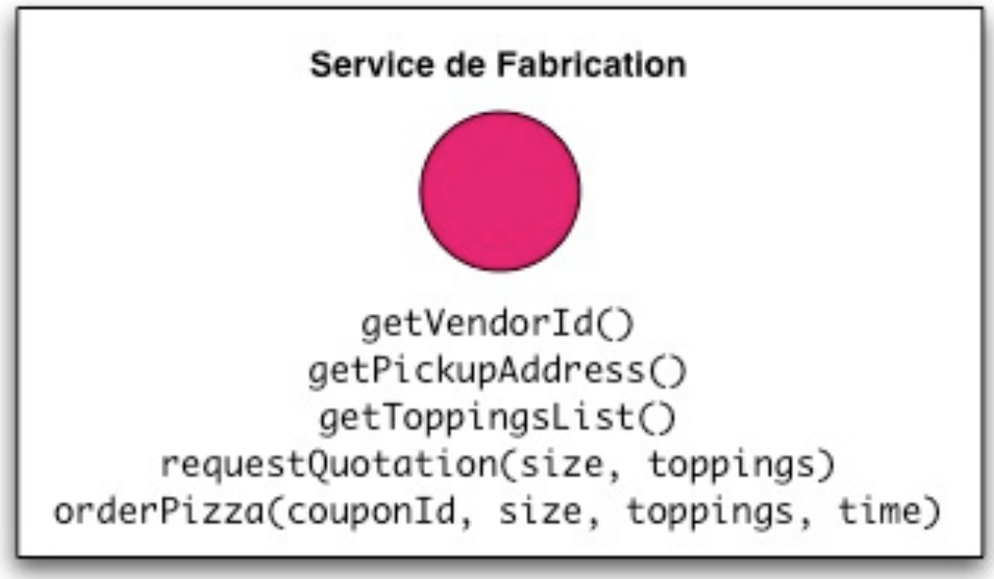


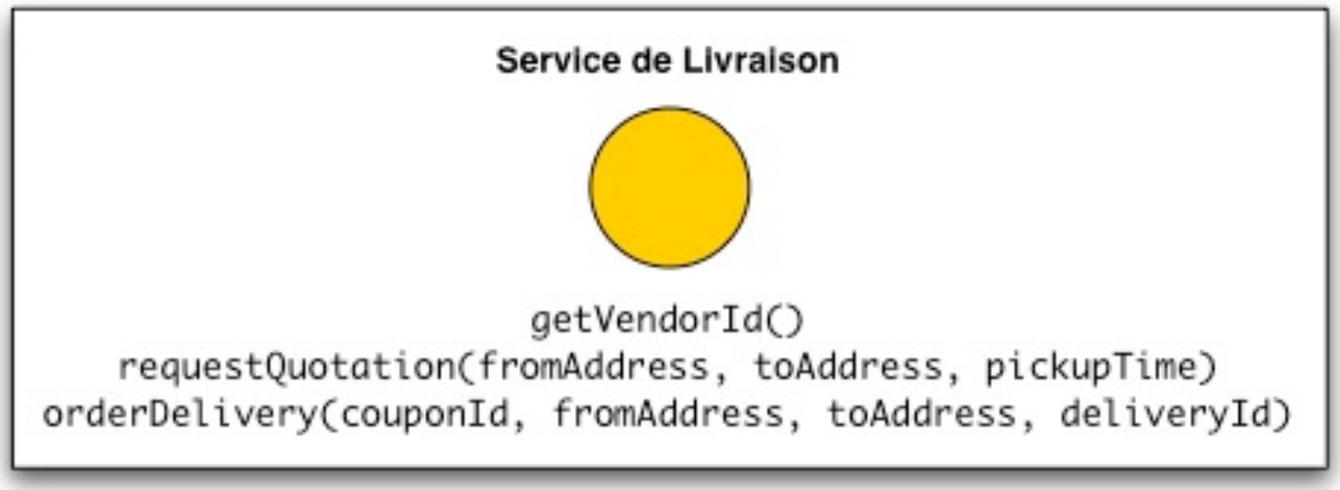
The Notion of Service Registry

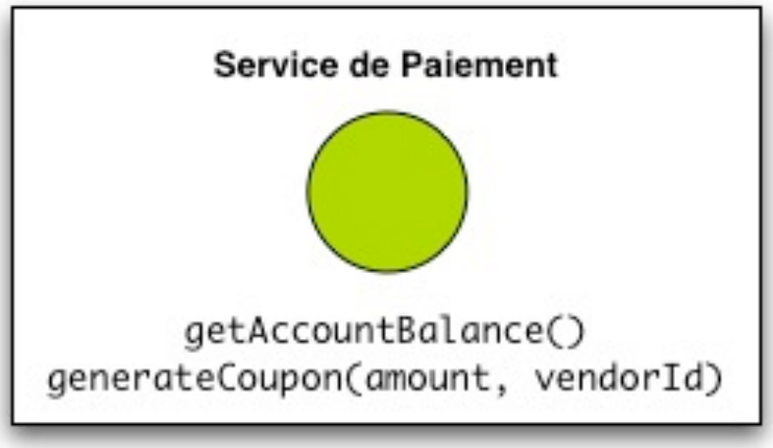
- Service Oriented Architectures go beyond simple service invocation:
  - Both in intra-, extra- and internet scenarios, the number of services is going to increase significantly.
  - More and more, building an application means reusing and combining these services.
  - Whether you call it a “mashup” or a “business process”, it’s the same idea!
- Example:
  - To eat a pizza, I need 1) a “pizza preparation” service, 2) a delivery service, 3) a payment service.
  - Different companies may provide these services (and not necessarily 24/7).
  - Questions: how to identify service providers? how to select service providers?

http://flickr.com/photos/qchristopher/1822166721/sizes/m/#cc_license

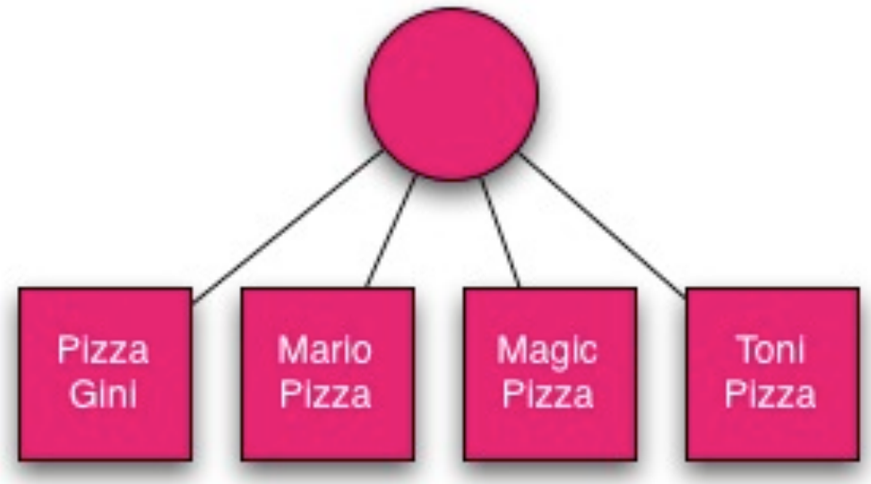
Example: Service Interfaces

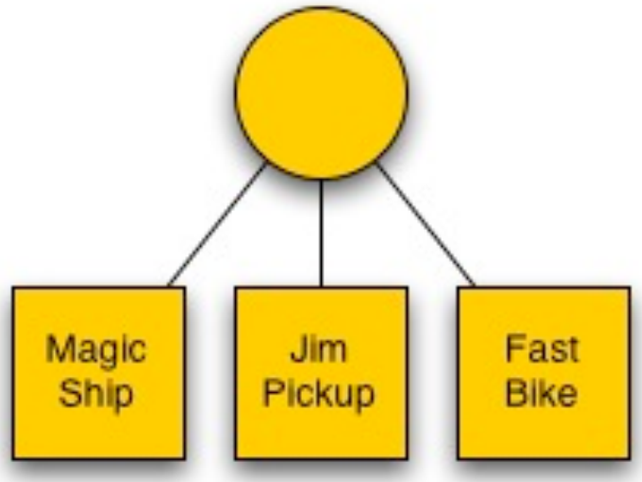


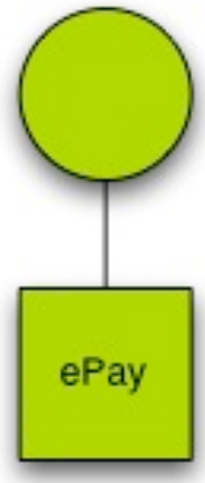




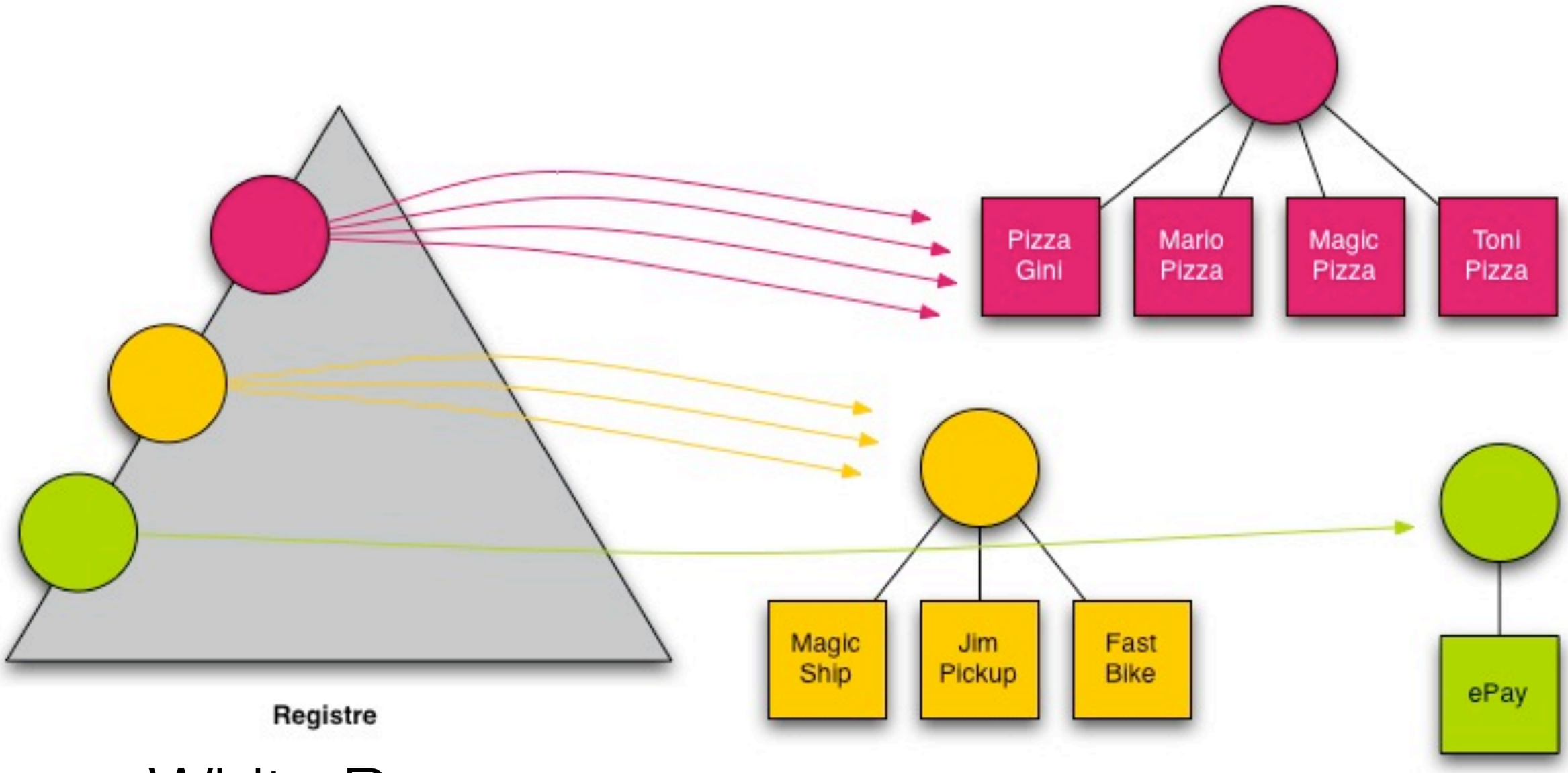
Example: Several Providers for a Service





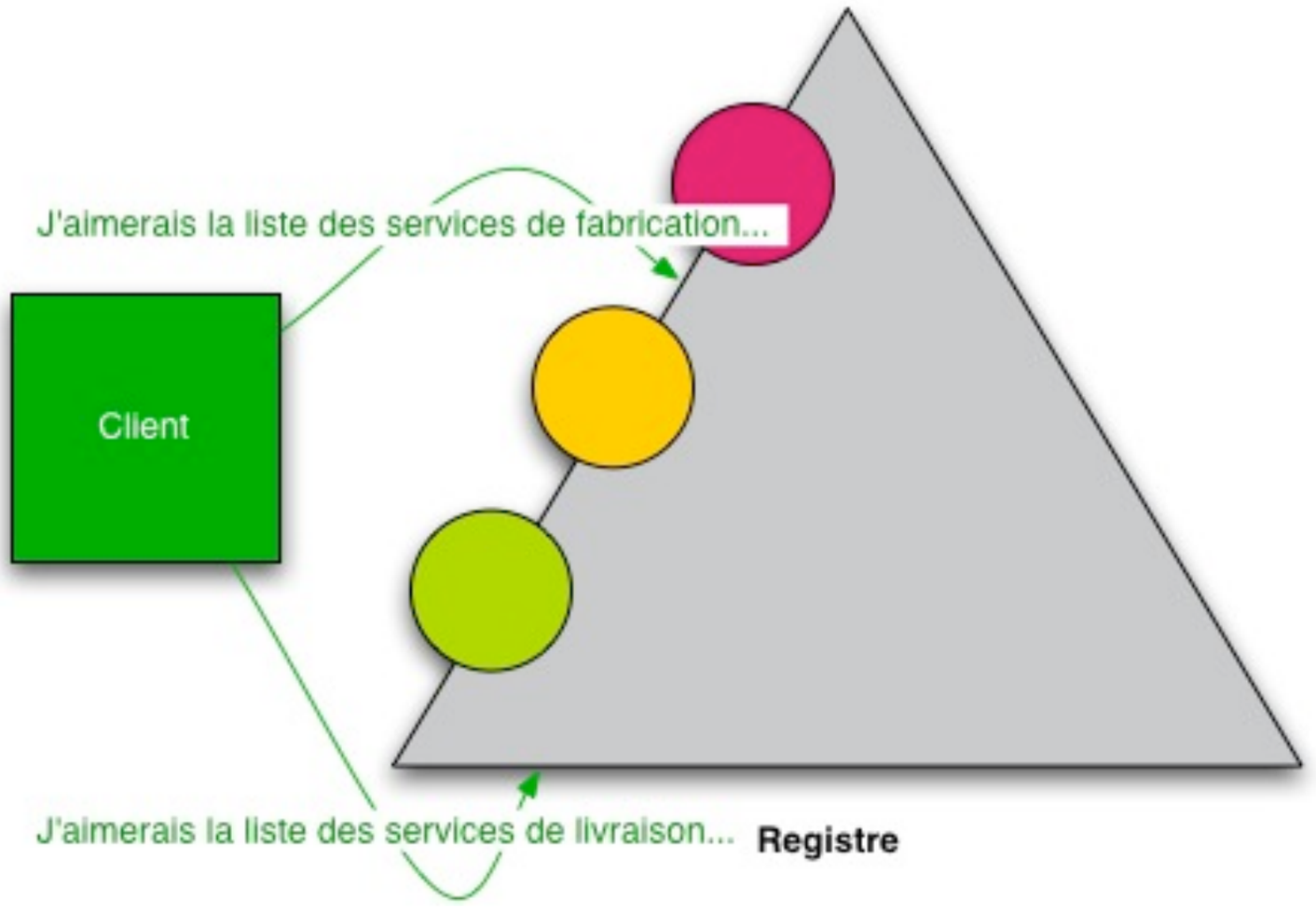


Example: Using a Service "Registry"

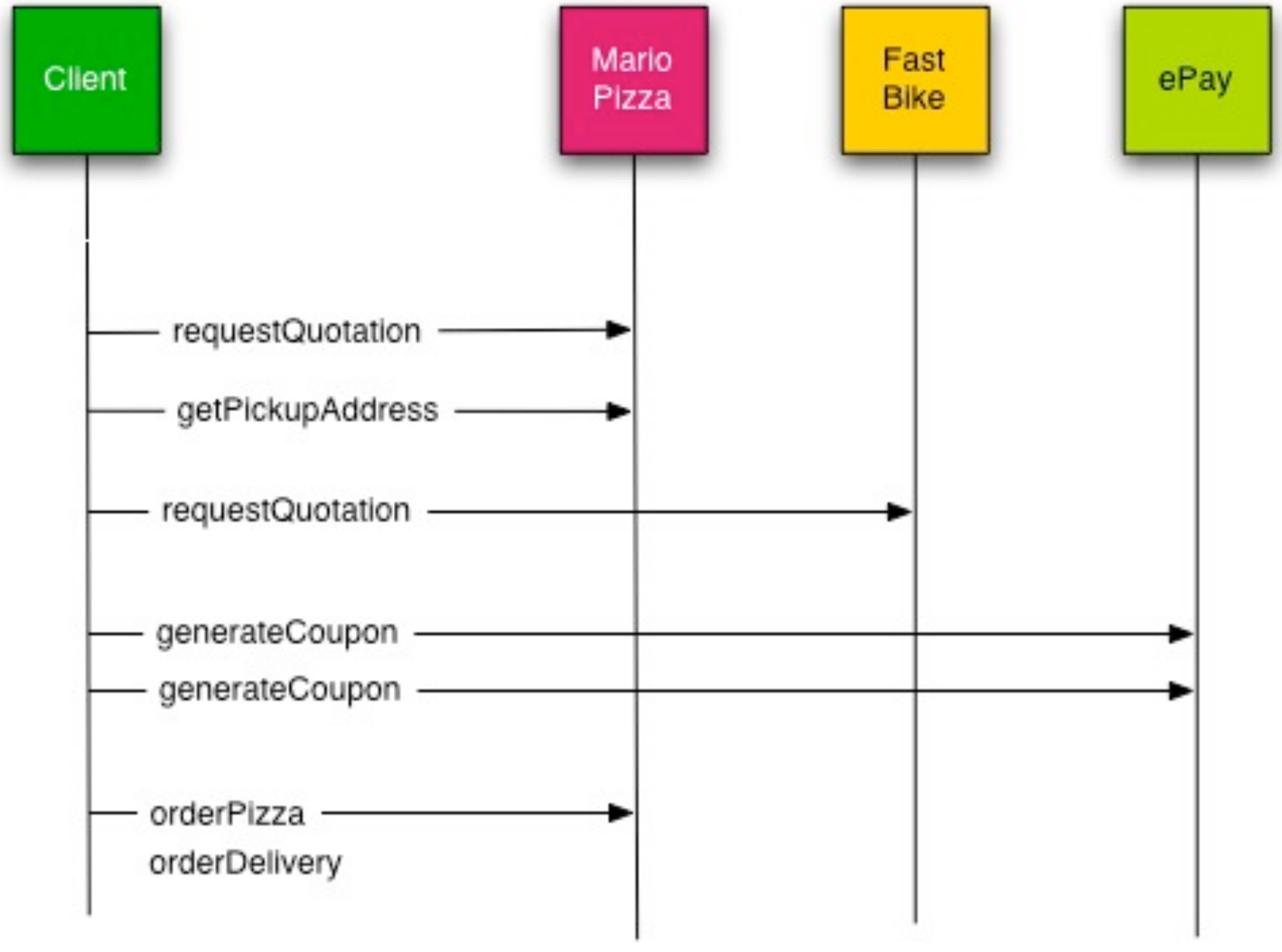


White Pages

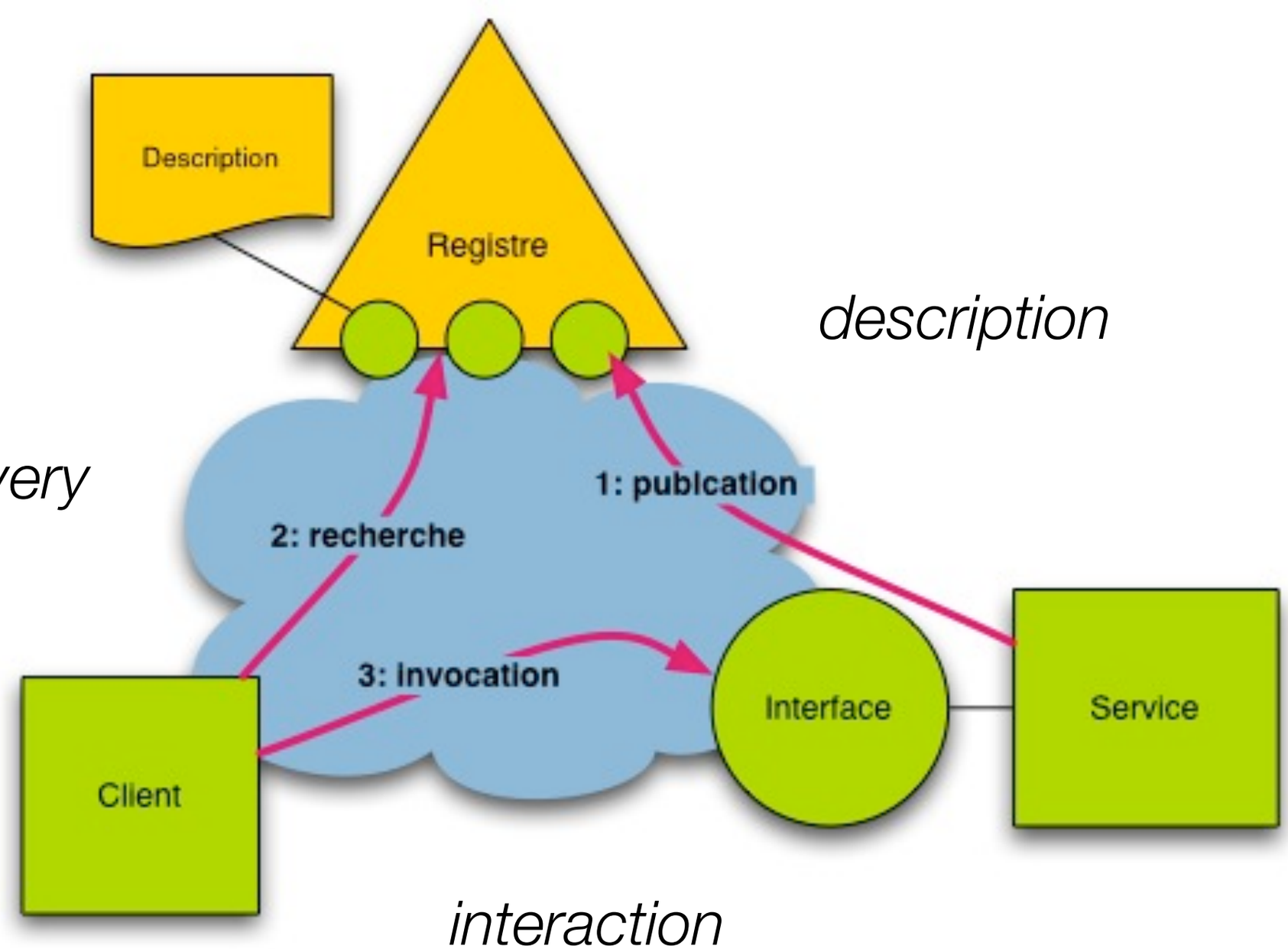
Example: Interacting with The Registry



Example: workflow



The Web Services Reference Architecture



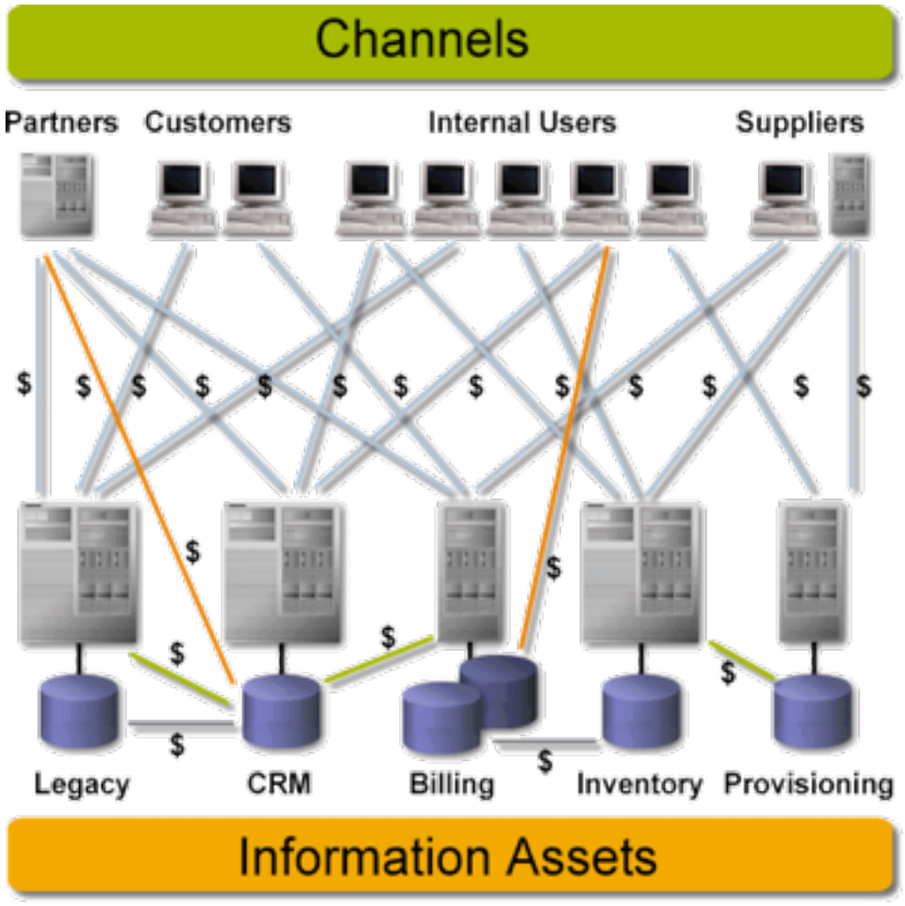
For a **Full** Service Architecture, We Need...

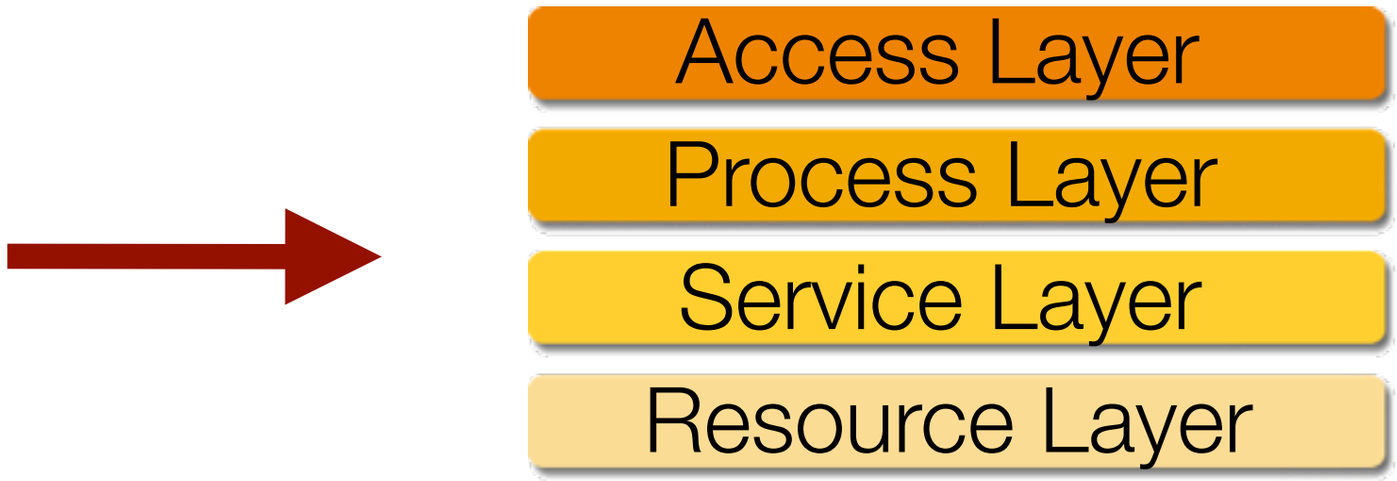
- A standardized format to **describe** service interfaces
  - Example: WSDL
- A standardized protocol to **invoke** services
  - Example: SOAP
- A **registry** service
  - Example: UDDI

Two Approaches to Web Services

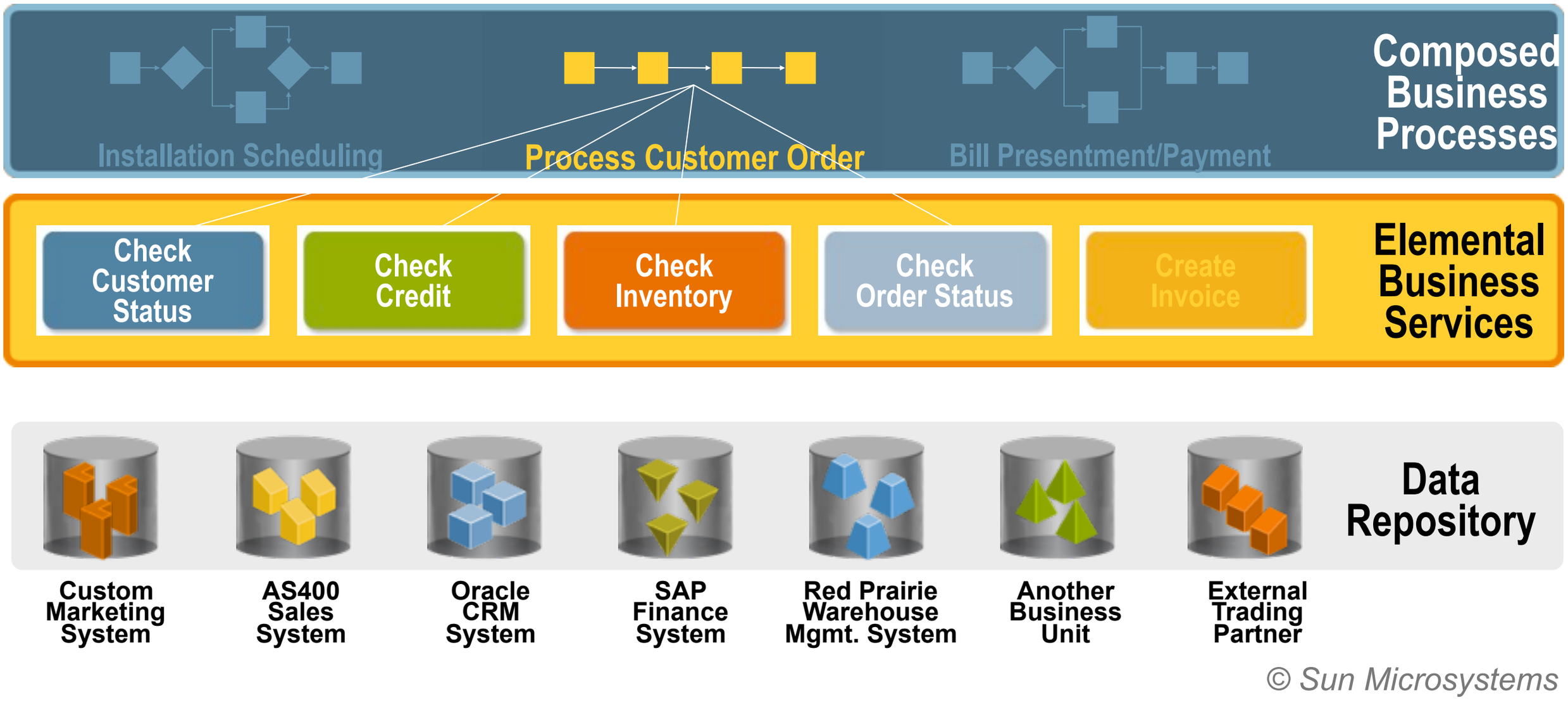
The Big Web Services Approach

Towards a Layered Architecture





Service Composition & Workflows



Big Web Services

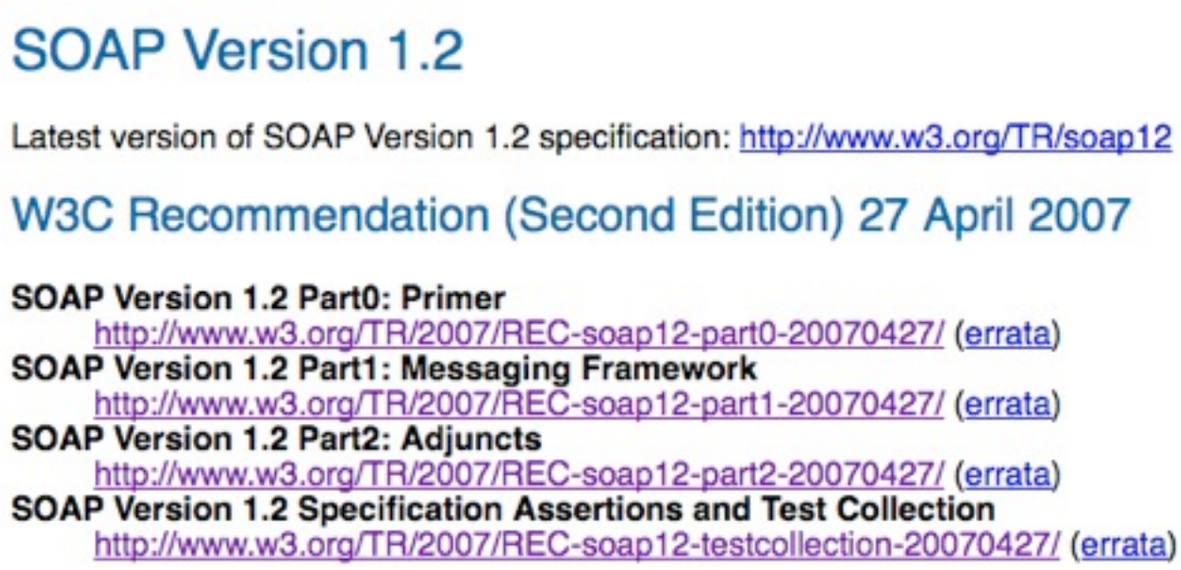
- Approach
  - Services are often designed and developed with a **RPC style** (even if Document-Oriented Services are possible).
- Core Standards
  - Simple Object Access Protocol (**SOAP**)
  - Web Services Description Language (**WSDL**)
- Benefits
  - **Very rich protocol stack** (support for security, transactions, reliable transfer, etc.)
- Problem
  - **Very rich protocol stack** (complexity, verbosity, incompatibility issues, theoretical human readability, etc.)

Web Services Standards Overview

http://www.innoq.com/soa/ws-standards/poster/
http://www.innoq.com/soa/ws-standards/

Simple Object Access Protocol

- Description
  - SOAP is a lightweight protocol for service invocation.
  - SOAP defines the structure of messages exchanged by clients and services.
  - SOAP messages can be exchanged via different transport protocols. HTTP is only one these protocols.
- Origin
  - SOAP a été spécifié suite à l'explosion de XML, en 1998.
- Specifications
  - La spécification SOAP 1.2 est une recommendation du W3C (27 avril 2007)
  - http://www.w3.org/TR/soap/



SOAP Version 1.2

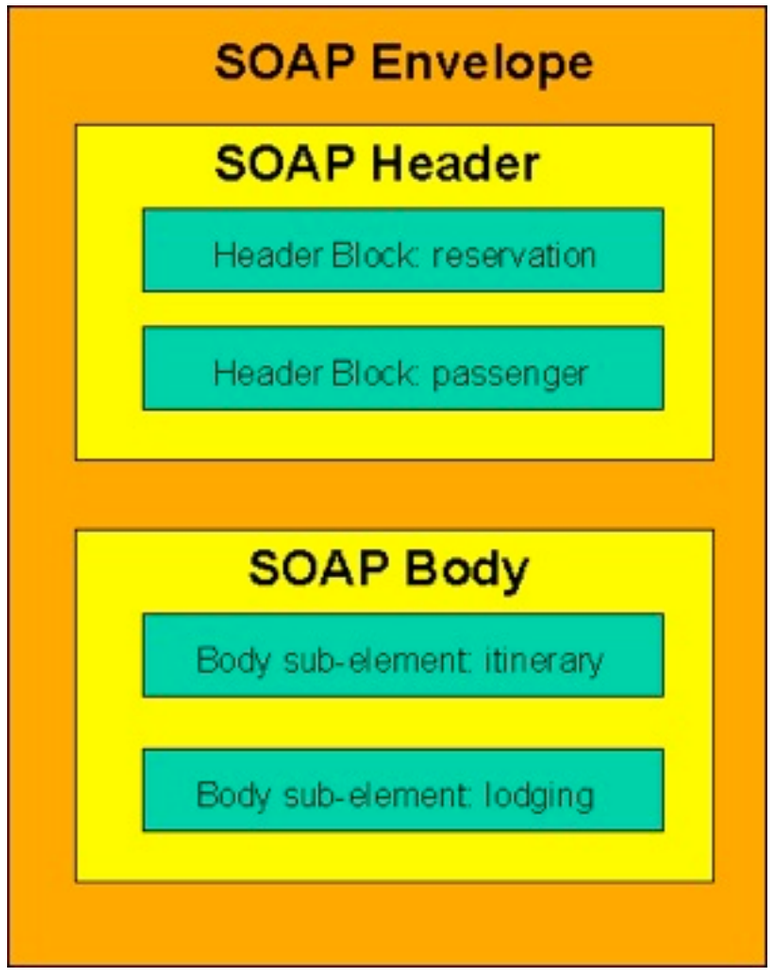
Latest version of SOAP Version 1.2 specification: http://www.w3.org/TR/soap12

W3C Recommendation (Second Edition) 27 April 2007

SOAP Version 1.2 Part0: Primer
http://www.w3.org/TR/2007/REC-soap12-part0-20070427/ (errata)
SOAP Version 1.2 Part1: Messaging Framework
http://www.w3.org/TR/2007/REC-soap12-part1-20070427/ (errata)
SOAP Version 1.2 Part2: Adjuncts
http://www.w3.org/TR/2007/REC-soap12-part2-20070427/ (errata)
SOAP Version 1.2 Specification Assertions and Test Collection
http://www.w3.org/TR/2007/REC-soap12-testcollection-20070427/ (errata)

Structure of a SOAP Message

- Header
  - Used to capture properties of the message or of the exchange.
  - Example: security management.
  - One of the extension points (all the properties have not been defined a priori).
- Body
  - The applicative payload.
  - Can capture a method invocation, with parameters.
  - Can capture a document (e.g. an order) to be processed by the service.



http://www.w3.org/TR/2007/REC-soap12-part0-20070427/

Example: The Flickr API

- **SOAP Request**

```
<?xml version="1.0" encoding="utf-8" ?>
<s:Envelope
        xmlns:s="http://www.w3.org/2003/05/soap-envelope"
        xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/1999/XMLSchema">

        <s:Body>
                <x:FlickrRequest xmlns:x="urn:flickr">
                        <method>flickr.test.echo</method>
                        <name>value</name>
                </x:FlickrRequest>
        </s:Body>

</s:Envelope>
```

Web Services Description Language (WSDL)

- SOAP is useful:
  - when we know “where” the service is (i.e. we know the service endpoint)
  - when we know the signature of the methods supported by the service
- But SOAP does not help in:
  - searching / looking up a service that complies to a certain interface
  - automatically generating a “stub” to be used on the client side
  - hence, we need a way to formally describe service interfaces!!
- WSDL: Web Services Description Language addresses this need
  - and thus allows the **automation** of procedures when dealing with web services.

As communications protocols and message formats are standardized in the web community, it becomes increasingly possible and important to be able to **describe the communications in some structured way**.

WSDL addresses this need by defining an XML grammar for **describing network services as collections of communication endpoints capable of exchanging messages**.

WSDL service definitions provide **documentation for distributed systems** and serve as a recipe for **automating** the details involved in applications communication.

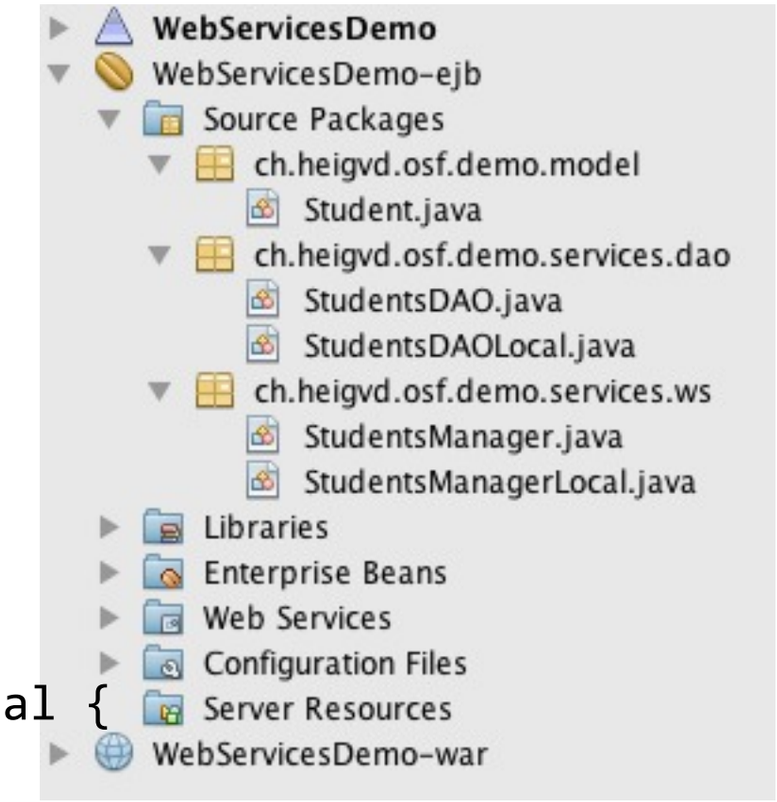
http://www.w3.org/TR/wsdl

Big Web Services & Java EE

Big Web Services with Java EE

- **JAX-WS**
  - JAX-WS makes it easier to write both **web services** and **web services clients**.
  - The JAX-WS **runtime** takes care of the SOAP and WSDL details and provides you with an object-oriented interface.
- Exposing your **Stateless Session Beans** with a Web Services interface$
  - Adding a single annotation will do the job.
  - JAX-WS relies on conventions for generating the WSDL interface; you can customize the schema with various annotations.

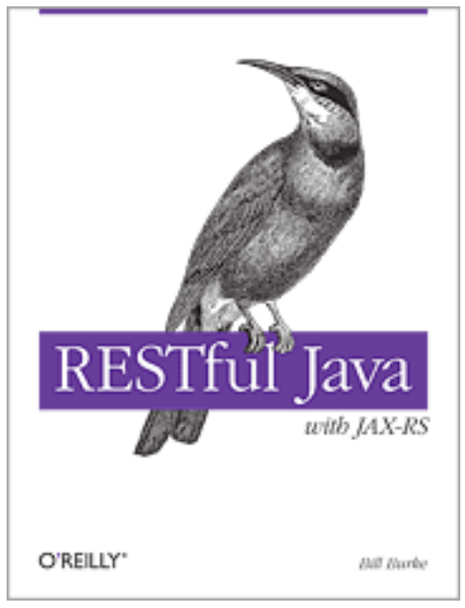
Demo

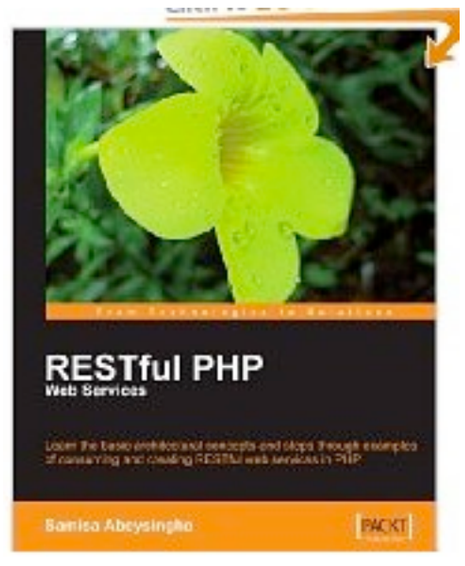



```
@Stateless
@WebService
public class StudentsManager implements StudentsManagerLocal {

    @EJB StudentsDAOLocal studentsDAO;

    public void createStudent(String firstName, String lastName) {
        studentsDAO.createStudent(firstName, lastName);
    }

    public Student findStudentById(long id) {
        return studentsDAO.findStudentById(id);
    }
}
```

The RESTful Approach

RESTful Web Services





Remote Procedure Call (RPC)

- What do we do when we use the RPC architecture style?
  - We use NAMES to identify services;
  - We use VERBS to define the functions of services (their interface).
- Examples:
  - "CustomerManager" service, with methods such as "addCustomer", "deleteCustomer", "findCustomerByName", etc.
  - "ClockService" service, with methods "getCurrentTime", "getCurrentTimeInSanFrancisco", "getCurrentTimeInTokyo", etc.
- That is typically what we do when use create Stateless Session Beans.
- That is typically what we do when we use SOAP and WSDL.

The REST Architectural Style

- REST: REpresentational State Transfer
- REST is an architectural style for building distributed systems.
- REST has been introduced in Roy Fielding's Ph.D. thesis (Roy Fielding has been a contributor to the HTTP specification, to the apache server, to the apache community).
- The WWW is one example for a distributed system that exhibits the characteristics of a REST architecture.

HTTP is a protocol for interacting with "resources"

What is a “Resource”

- At first glance, one could think that a “resource” is a file on a web server:
  - an HTML document, an XML document, a PNG document
- That fits the vision of the “static content” web
- But of course, the web is now more than a huge library of hypermedia documents:
  - through the web, we interact with services and a lot of the content is dynamic.
  - more and more, through the web we interact with physical objects (machines, sensors, actuators)
  - We need a more generic definition for resources!

What is a “Resource”?

- A resource is "something" that can be named and uniquely identified:
  - Example 1: an article published in the "24 heures" newspaper
  - Example 2: the collection of articles published in the sport section of the newspaper
  - Example 3: a person’s resume
  - Example 4: the current price of the Nestlé stock quote
  - Example 5: the vending machine in the school hallway
  - Example 6: the list of grades of the student Jean Dupont
- URL (Uniform Resource Locator) is a mechanism for identifying resources
  - Exemple 1: http://www.24heures.ch/vaud/vaud/2008/08/04/trente-etudiants-partent-rencontre-patrons
  - Exemple 2: http://www.24heures.ch/articles/sport
  - Exemple 5: http://www.smart-machines.ch/customers/heig/machines/8272

Resource vs. Representation

- A "resource" can be something intangible (stock quote) or tangible (vending machine)
- The HTTP protocol supports the exchange of data between a client and a server.
- Hence, what is exchanged between a client and a server is **not** the resource. It is a **representation** of a resource.
- Different representations of the same resource can be generated:
  - HTML representation
  - XML representation
  - PNG representation
  - WAV representation
- **HTTP provides the content negotiation mechanisms!!**

How Do We Interact With Resources?

- The HTTP protocol defines the standard methods. These methods enable the interactions with the resources:
  - **GET**: retrieve whatever information is identified by the Request-URI
  - **POST**: used to request that the origin server accept the entity enclosed in the request as a new subordinate of the ressource identified by the Request-URI in the Request-Line
  - **PUT**: requests that the enclosed entity be stored under the supplied Request-URI.
  - **DELETE**: requests that the origin server delete the ressource identified by the Request-URI.
  - HEAD: identical to GET except that the server MUST NOT return a message-body in the response
  - TRACE: used for debugging (echo)
  - CONNECT: reserved for tunneling purposes

Principles of a REST Architecture

- The state of the application is captured in a set of resources
  - Users, photos, comments, tags, albums, etc.
- Every resource can be identified with a standard format (e.g. URL)
- Every resource can have several representations
- There is one unique interface for interacting with resources (e.g. HTTP methods)
- The communication protocol is:
  - client-server
  - stateless
  - cacheable
- These properties have a huge positive impact on systemic qualities (scalability, performance, availability, etc.).
  - Reference: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

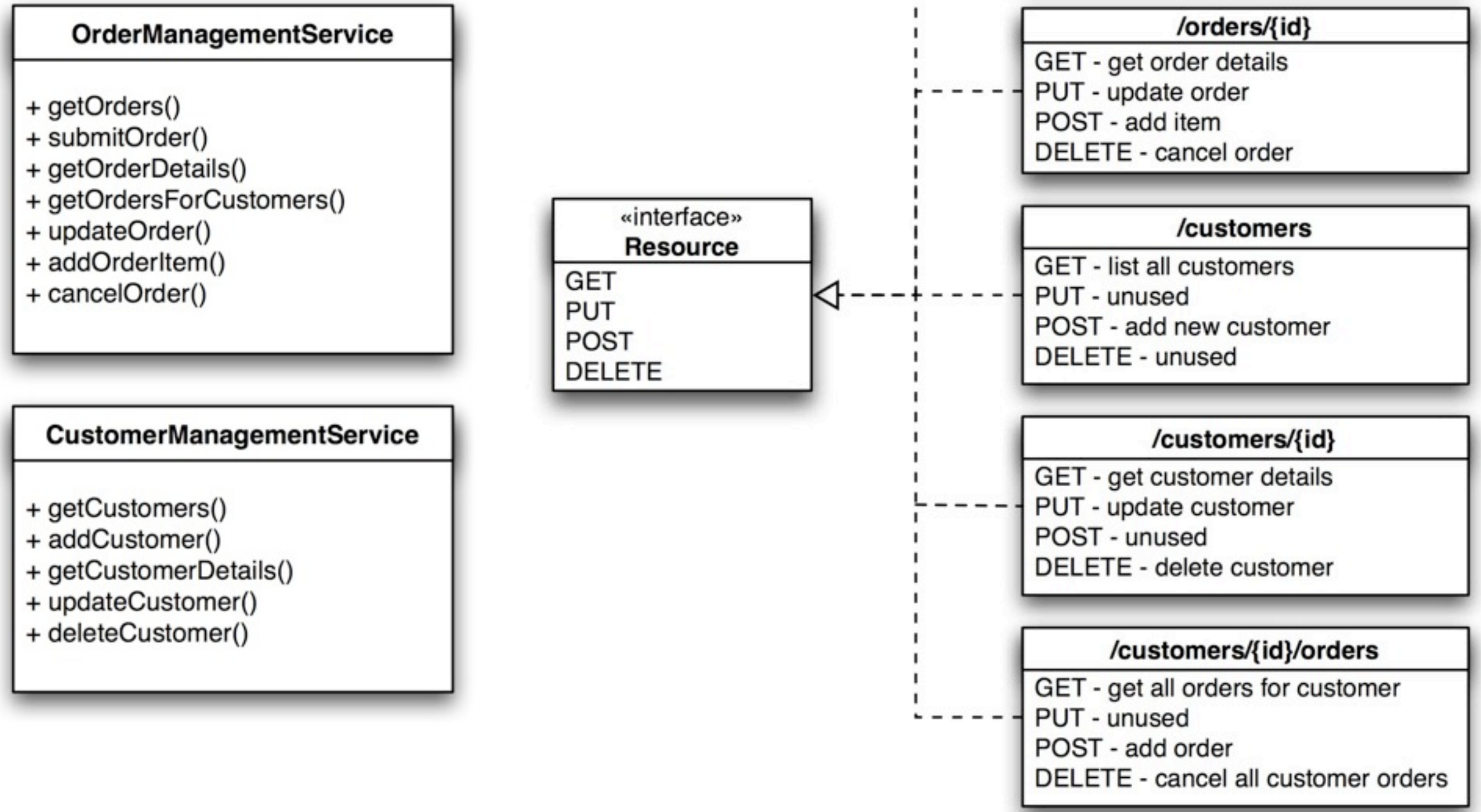
Design a RESTful system

- Start by identifying the resources - the NAMES in your system.
- Define the structure of the URLs that will be mapped to your resources.
- Some examples:
  - http://www.photos.com/users/oliechti identifies a resource of type "user". A client can do a "HTTP GET" to obtain a representation of the user or a “HTTP PUT” to update the user.
  - http://www.photos.com/users identifies a resource of type "collection of users". A client can do a "HTTP POST" to add users, or an "HTTP GET" to obtain the list of users.

Reference

- Very good article, with presentation of key concepts and illustrative examples:
  - http://www.infoq.com/articles/rest-introduction

RPC vs REST


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

How to write a “RESTful” Web Service?

- On the server side, one could do everything in a FrontController servlet:
  - Parse URLs
  - Do a mapping between URLs and Java classes that represent resources
  - Generate the different representations of resources
  - etc.
- But of course, there are frameworks that do exactly that for us.
- It is true for nearly every platform and language, including Java.
- There is even a JSR for that: JAX-RS (JSR 311).
  - Oracle provides the reference implementation, in the Jersey project (open source).

RESTful Services with Java EE

- **JAX-RS**
  - JAX-RS provides a programming model, classes and annotations for easily building RESTful APIs.
  - Jersey is the name of the standard JAX-RS implementation, which is bundled with the Glassfish application server.
- **JAXB**
  - You do not have to worry about the serialization of your business objects to XML or JSON. The framework will take care of (most of) the details for you.
- For all of your business “resources”, create a **JAX-RS resource class**
  - Use **annotations** to **route HTTP requests to your resource class and methods** (based on target URI, HTTP method, HTTP accept header, etc.)

Java is a trademark of Sun Microsystems, Inc.

Developing RESTful Web Services with JAX-RS

Marc Hadley
Paul Sandoz
Sun Microsystems, Inc

RESTful Application Cycle

- Resources are identified by URIs
- ↓
- Clients communicate with resources via requests using a standard set of methods
- ↓
- Requests and responses contain resource representations in formats identified by media types
- ↓
- Responses contain URIs that link to further resources

Resources are identified by URIs

- Resource == Java class
  - POJO
- No required interfaces
- ID provided by `@Path` annotation
  - Value is relative URI, base URI is provided by deployment context or parent resource
  - Embedded parameters for non-fixed parts of the URI
- Annotate class or “sub-resource locator” method

Resources are identified by URIs

```
>@Path("properties")
>public class SystemProperties {
>   @GET
>   List<SystemProperty> getProperties(...) {...}
>
>   @Path("{name}")
>   SystemProperty getProperty(...) {...}
>}
```

Standard Set of Methods

- Annotate resource class methods with standard method
  - **`@GET`**, **`@PUT`**, **`@POST`**, **`@DELETE`**, **`@HEAD`**
  - **`@HttpMethod`** meta-annotation allows extensions, e.g. WebDAV
- JAX-RS routes request to appropriate resource class and method
- Flexible method signatures, annotations on parameters specify mapping from request
- Return value mapped to response

Standard Set of Methods

```
>@Path("properties/{name}")
>public class SystemProperty {
>
>   @GET
>   Property get(@PathParam("name") String name)
>     {...}
>
>   @PUT
>   Property set(@PathParam("name") String name,
>     String value) {...}
>
>}
```

Resource Representations

- Representation format identified by media type.

E.g.:

- XML - application/properties+xml
- JSON - application/properties+json
- (X)HTML+microformats - application/xhtml+xml

- JAX-RS automates content negotiation
  - **`GET /foo`**
    `Accept: application/properties+json`

Resource Representations

```
>@GET
>@Produces("application/properties+xml")
>Property getXml(@PathParam("name") String name)
{
>   ...
>}
>
>@GET
>@Produces("text/plain")
>String getText(@PathParam("name") String name) {
>   ...
>}
```

Responses Contain Links

```
HTTP/1.1 201 Created
Date: Wed, 03 Jun 2009 16:41:58 GMT
Server: Apache/1.3.6
Location: http://example.com/properties/foo
Content-Type: application/order+xml
Content-Length: 184

<property self="http://example.com/properties/foo">
  <parent ref="http://example.com/properties/bar"/>
  <name>Foo</name>
  <value>1</value>
</order>
```

Responses Contain Links

```
@Context UriInfo i;

SystemProperty p = ...
UriBuilder b = i.getBaseUriBuilder();
URI u = b.path(SystemProperties.class)
    .path(p.getName()).build();

List<URI> ancestors = i.getMatchedURIs();
URI parent = ancestors.get(1);
```

Demo

```
@Path("/students")
public class StudentsResource {

      StudentsDAOLocal studentsDAO = lookupStudentsDAOLocal();

      @Context
      private UriInfo context;


      /**
       * Creates a new instance of StudentsResource
       */
      public StudentsResource() {
      }

      /**
       * Retrieves representation of the collection resource
       * @return an instance of List<Student>
       */
      @GET
       @Produces("application/xml, application/json")
      public List<Student> getXml() {
            // Let's generate random students for demo purposes...
            // don't try to understand the logic of this
            List<Student> dummyResult = new LinkedList<Student>();
            dummyResult.add(studentsDAO.findStudentById(42));
            dummyResult.add(studentsDAO.findStudentById(42));
            dummyResult.add(studentsDAO.findStudentById(42));
            dummyResult.add(studentsDAO.findStudentById(42));
            dummyResult.add(studentsDAO.findStudentById(42));
            return dummyResult;
      }

  ...
}
```