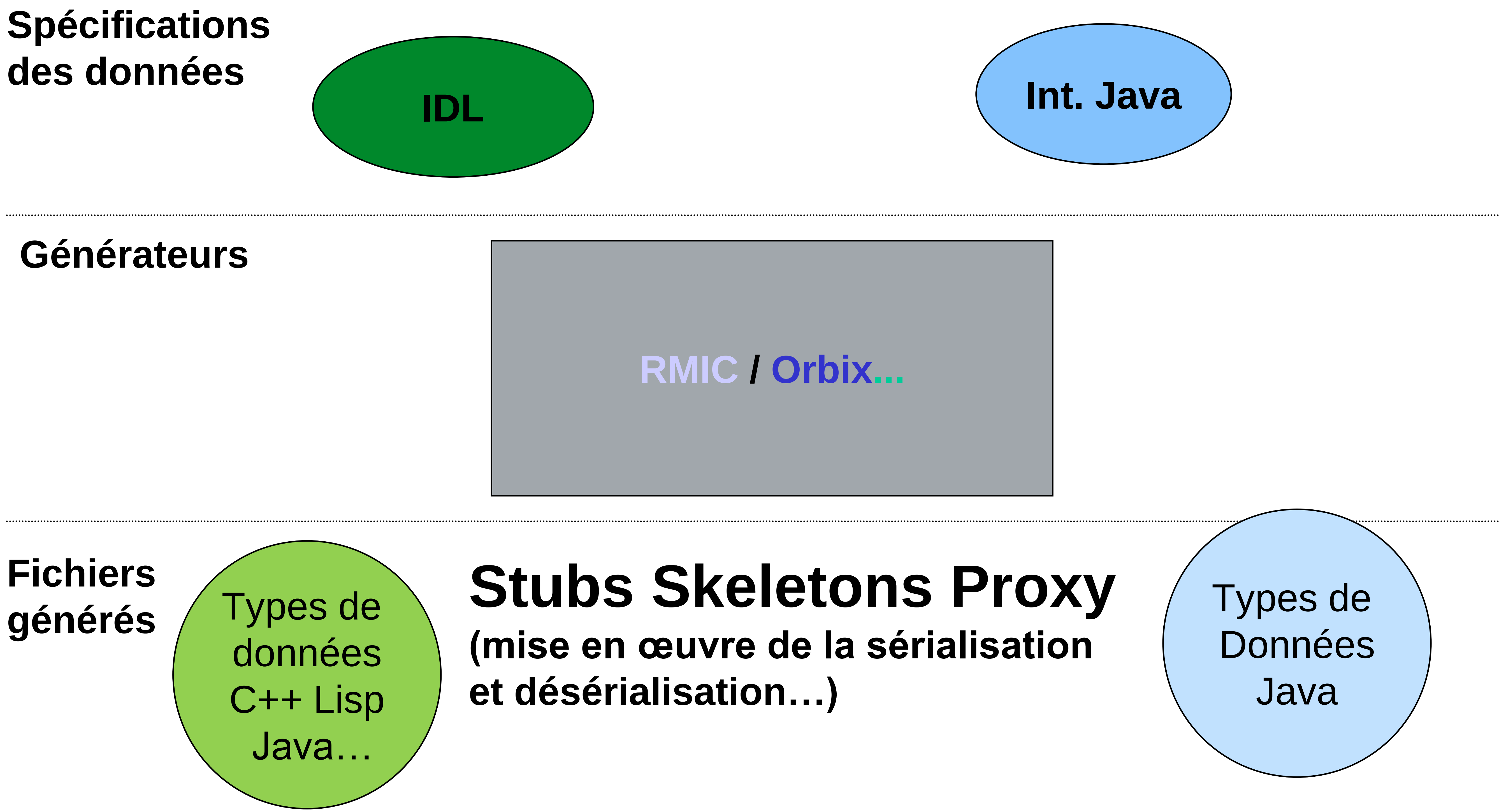
Des objets distribués aux composants

Générateurs


Spécifications des données
IDL
Int. Java
Générateurs
RMIC / Orbix...
Fichiers générés
Types de données C++ Lisp Java…
Stubs Skeletons Proxy
(mise en œuvre de la sérialisation et désérialisation…)
Types de Données Java

Protocoles d'application et Langages de spécifications

- Spécifications des types de données qui transitent sur le réseau

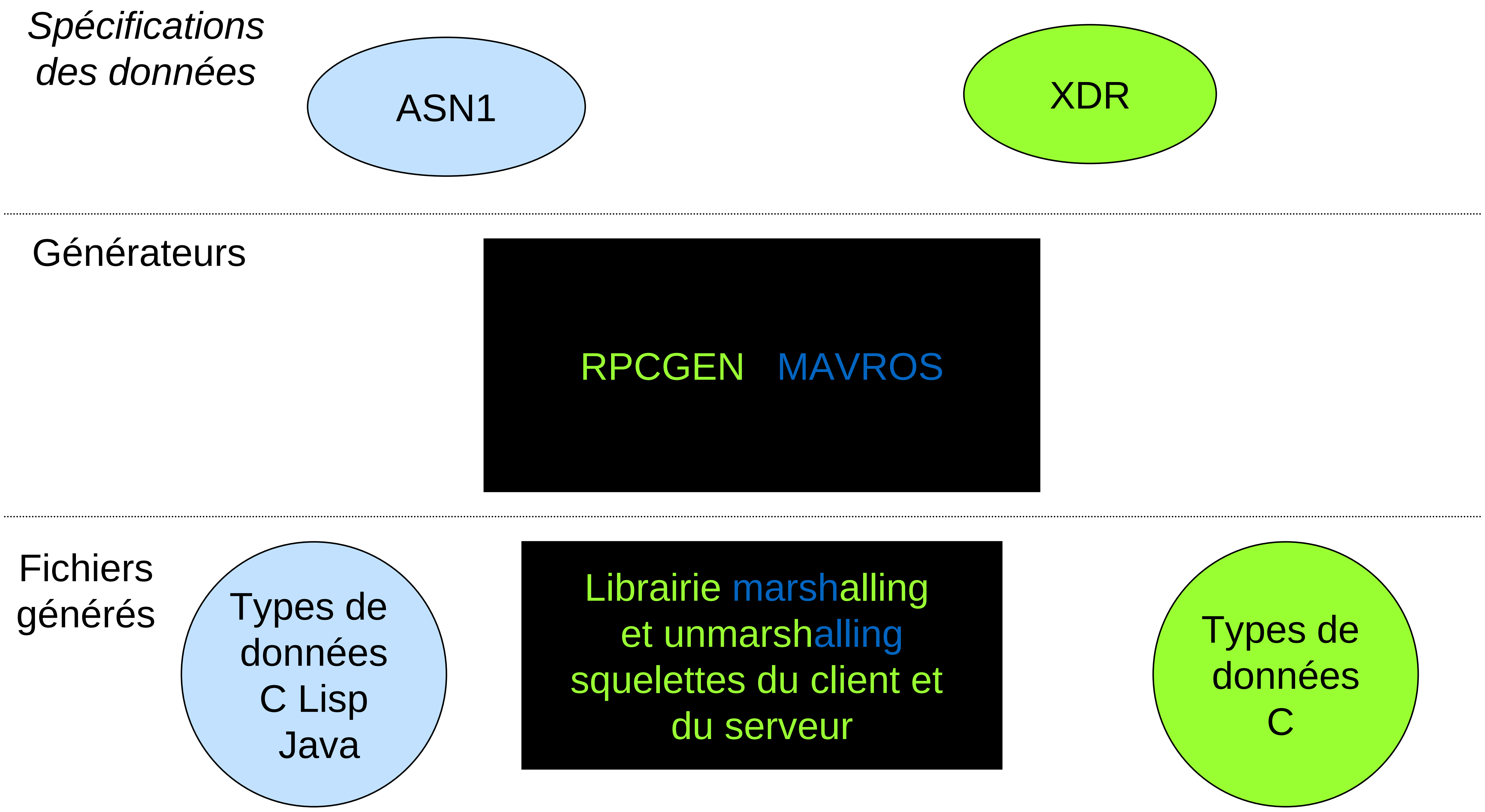
```
Protocole := CHOICE {
        requete [0] REQUETE,
        reponse [1] REPONSE }
```

ASN.1 et norme ISO

```
Programme reqrep {
        version {
            REPONSE rerep(REQUETE) = 1
        }= 1
} = 10000
```

XDR et RPC de SUN

Générateurs de Stubs


Spécifications des données
ASN1
XDR
Générateurs
RPCGEN
MAVROS
Fichiers générés
Types de données C Lisp Java
Librairie marshalling et unmarshalling squelettes du client et du serveur
Types de données C

Attention au vocabulaire

- Coté client :
  - stub en CORBA
  - proxy en OLE
  - stub/proxy en Java
- Côté Serveur :
  - stub en OLE
  - skeleton en CORBA
  - implémentation d’une interface en RMI

Services et Objets Distribués

Middleware CORBA

- Services normalisés
- Seulement certains sont implémentés
- Naming, Trading, Event

Middleware RMI

Des services en programmant avec Java Sécurité,Threads, Événements

Url et Web

Non intégrés à RMI

Un composant, c’est quoi ?

Une brique permettant la programmation par assemblage

Une solution facilitant le déploiement, la gestion du cycle de vie des applications logicielles

Une meilleure intégration des services

EJB – CORBA 3: Apports

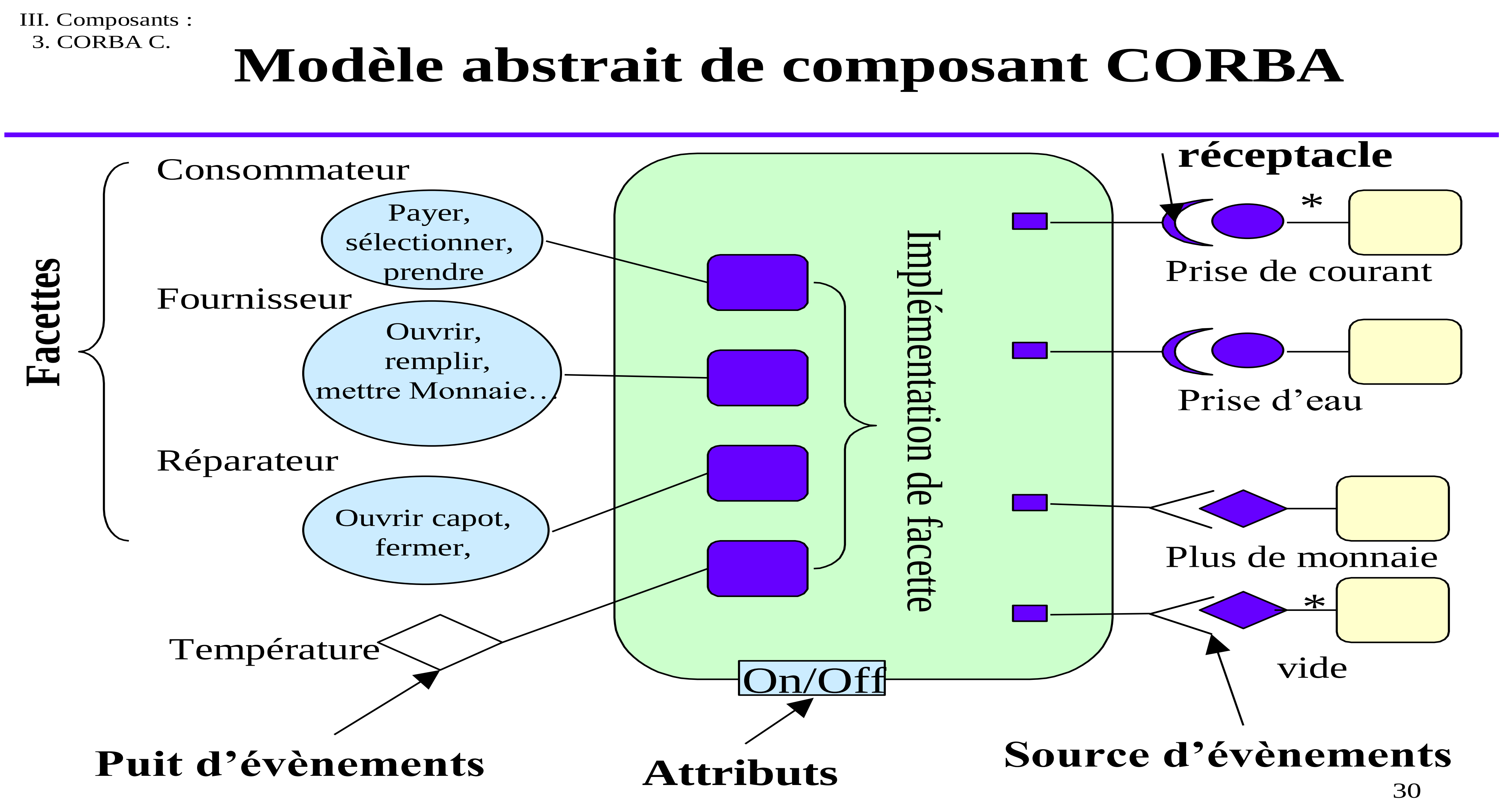
Interfaces entrées et sorties : ports requis et offerts

Conteneur : intégration des propriétés non fonctionnelles (sécurité, persistance, transaction)

Home : fabrique et navigation

Communication par envoi de message et notification (événement)

Exemple

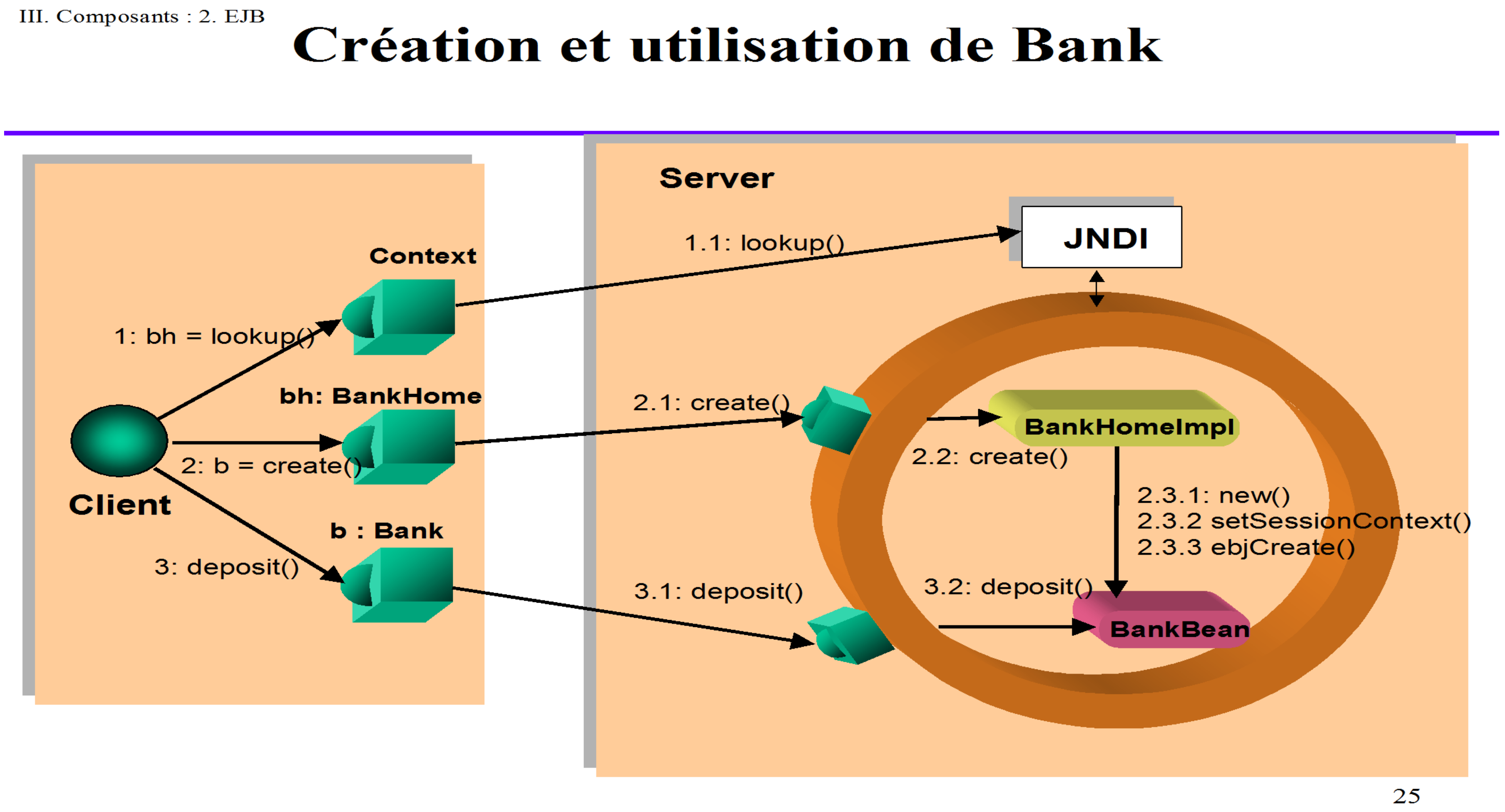


III. Composants :
3. CORBA C.

Modèle abstrait de composant CORBA

30

Exemple



Création et utilisation de Bank

Points communs avec les middlewares objets

Langages de description : CIDL ou Interfaces Java

Infrastructure : ORB / RMI

Marshalling : repose sur Corba / RMI

Nommage : Home ++

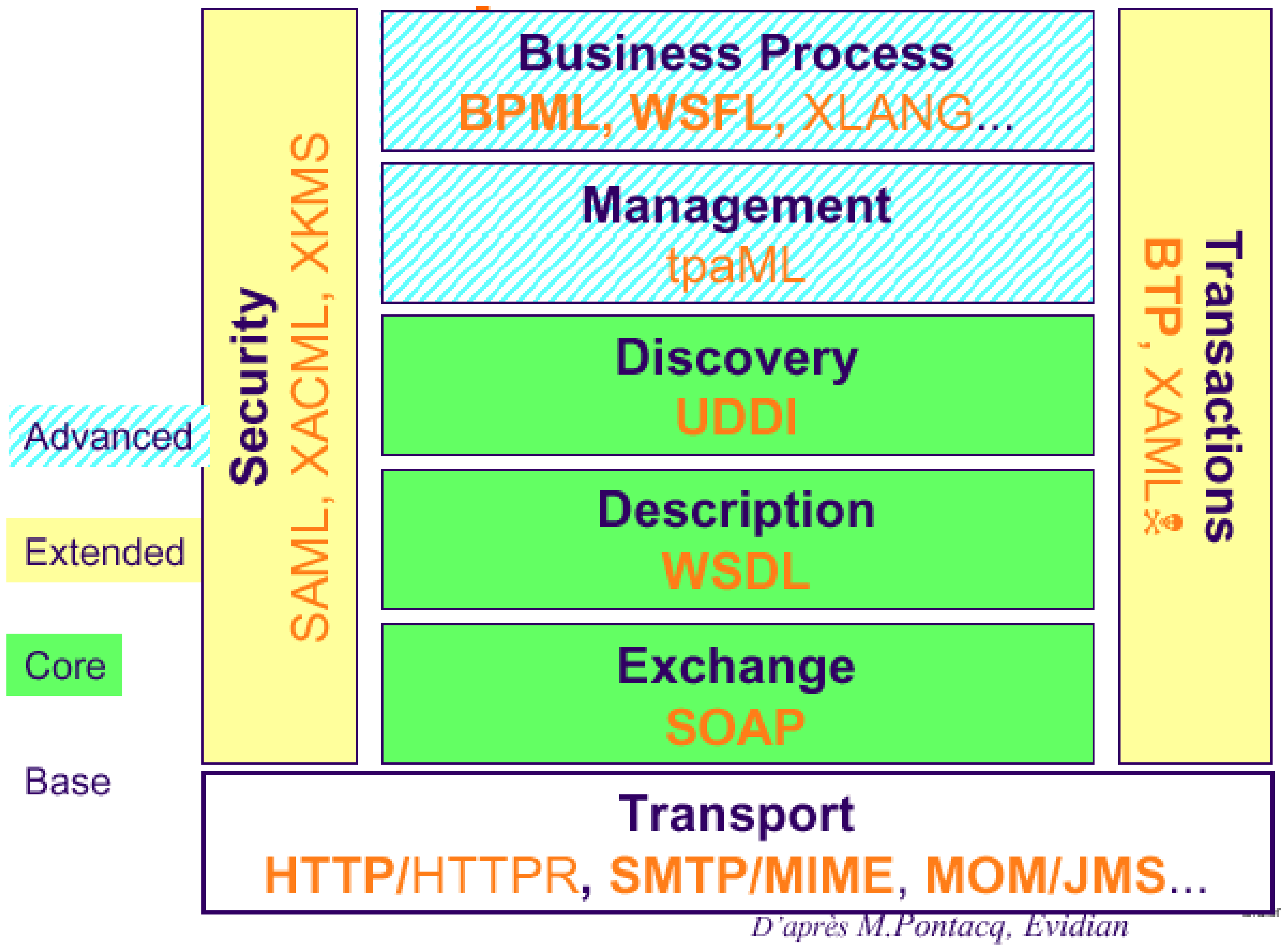
Interface : Héritage + Composition

Un Service Web, c’est quoi ?

- Une « unité logique applicative »
- Une «librairie» fournissant des données et des services à d’autres applications.
- Un objet métier déployé sur le web (vision objet)
- Un « module » ou « composant » ?

➢ Une sorte d'objet… plutôt qu'un composant

Architecture globale



Points communs avec les middlewares objets

Un langage de description : WSDL

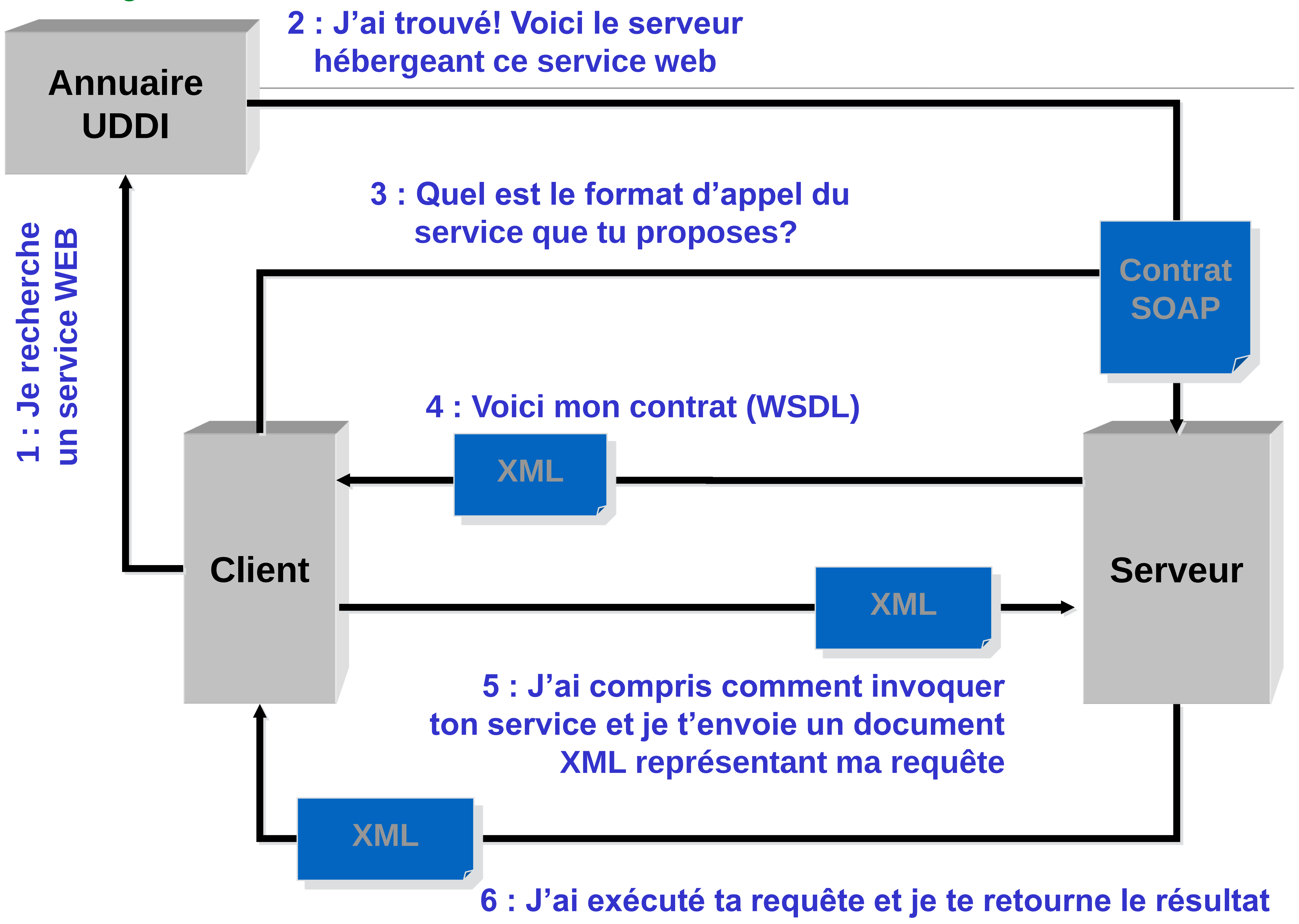
Une infrastructure : Le Web et http

Une communication par envoi de messages : SOAP

Du marshalling : XML

Un service de nommage « dynamique » : UDDI

Cycle de vie d'utilisation

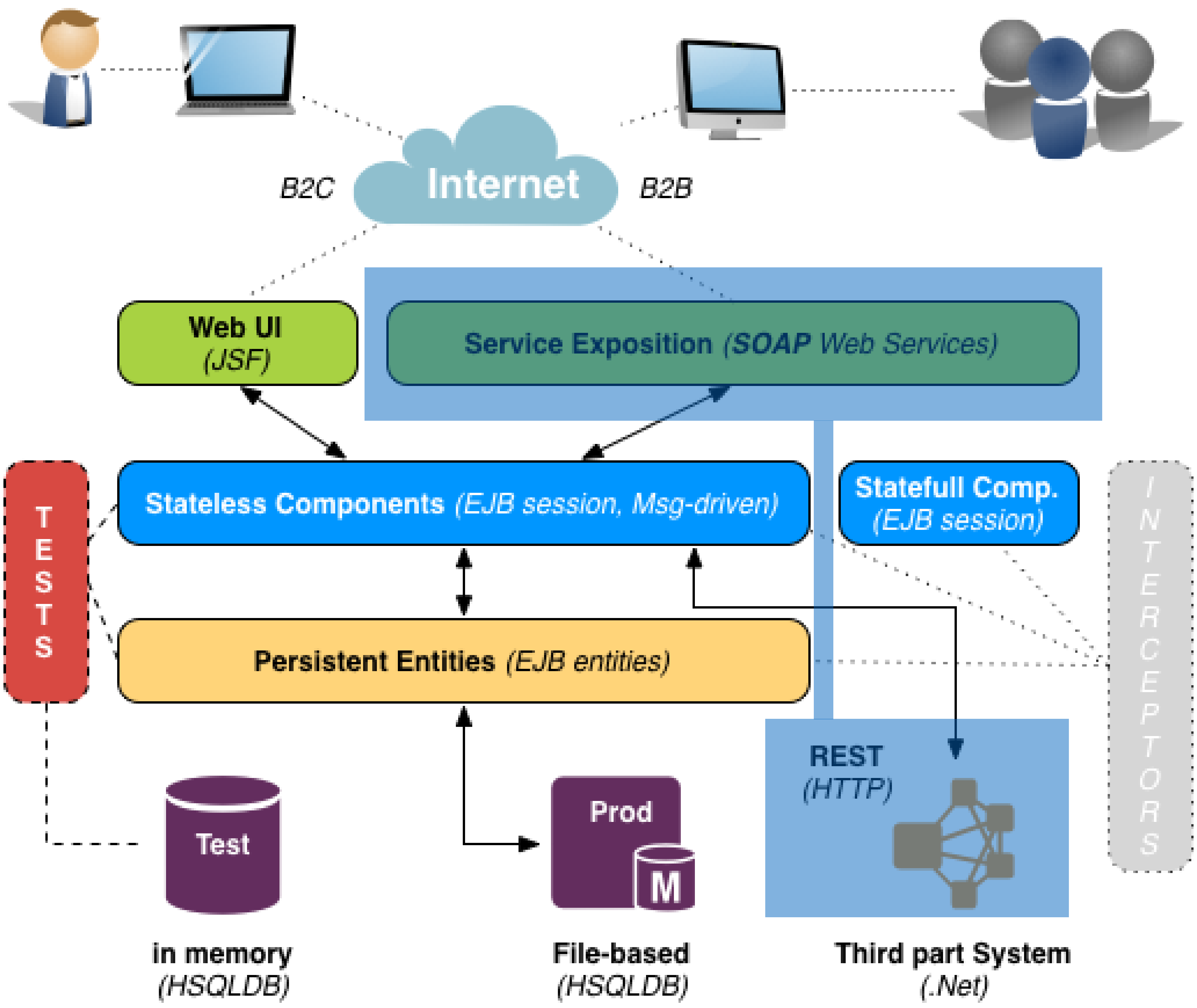

Annuaire UDDI
2 : J'ai trouvé! Voici le serveur hébergeant ce service web
1 : Je recherche un service WEB
3 : Quel est le format d'appel du service que tu proposes?
Contrat SOAP
4 : Voici mon contrat (WSDL)
XML
Client
Serveur
XML
5 : J'ai compris comment invoquer ton service et je t'envoie un document XML représentant ma requête
XML
6 : J'ai exécuté ta requête et je te retourne le résultat

Environnements intégrés .net

- Toute la mécanique est cachée
- On peut se concentrer sur la conception
- Aide à l'assemblage ?

Interoperability with Web Services

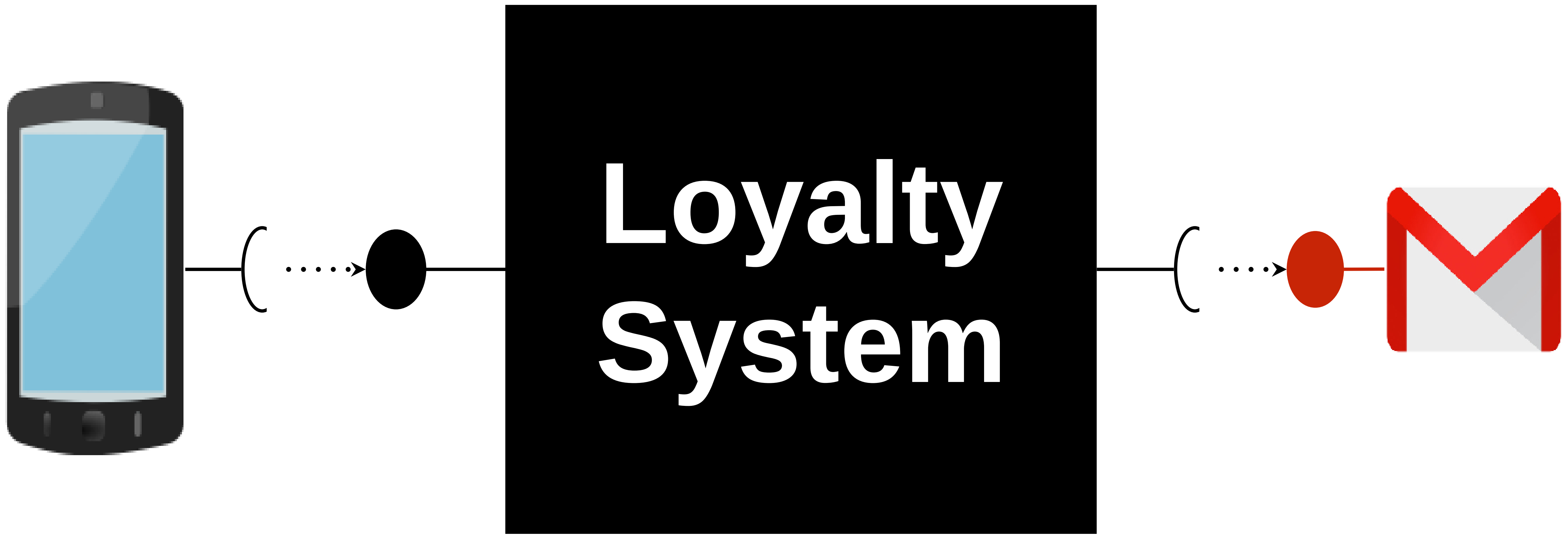
AM Dery
Fortement inspirée des cours de
S Mosser


B2C
Internet
B2B
Web UI
(JSF)
Service Exposition (SOAP Web Services)
T E S T S
Stateless Components (EJB session, Msg-driven)
Statefull Comp.
(EJB session)
I N T E R C E P T O R S
Persistent Entities (EJB entities)
REST
(HTTP)
Test
Prod
M
in memory
(HSQLDB)
File-based
(HSQLDB)
Third part System
(.Net)

The Cookie Factory open source example

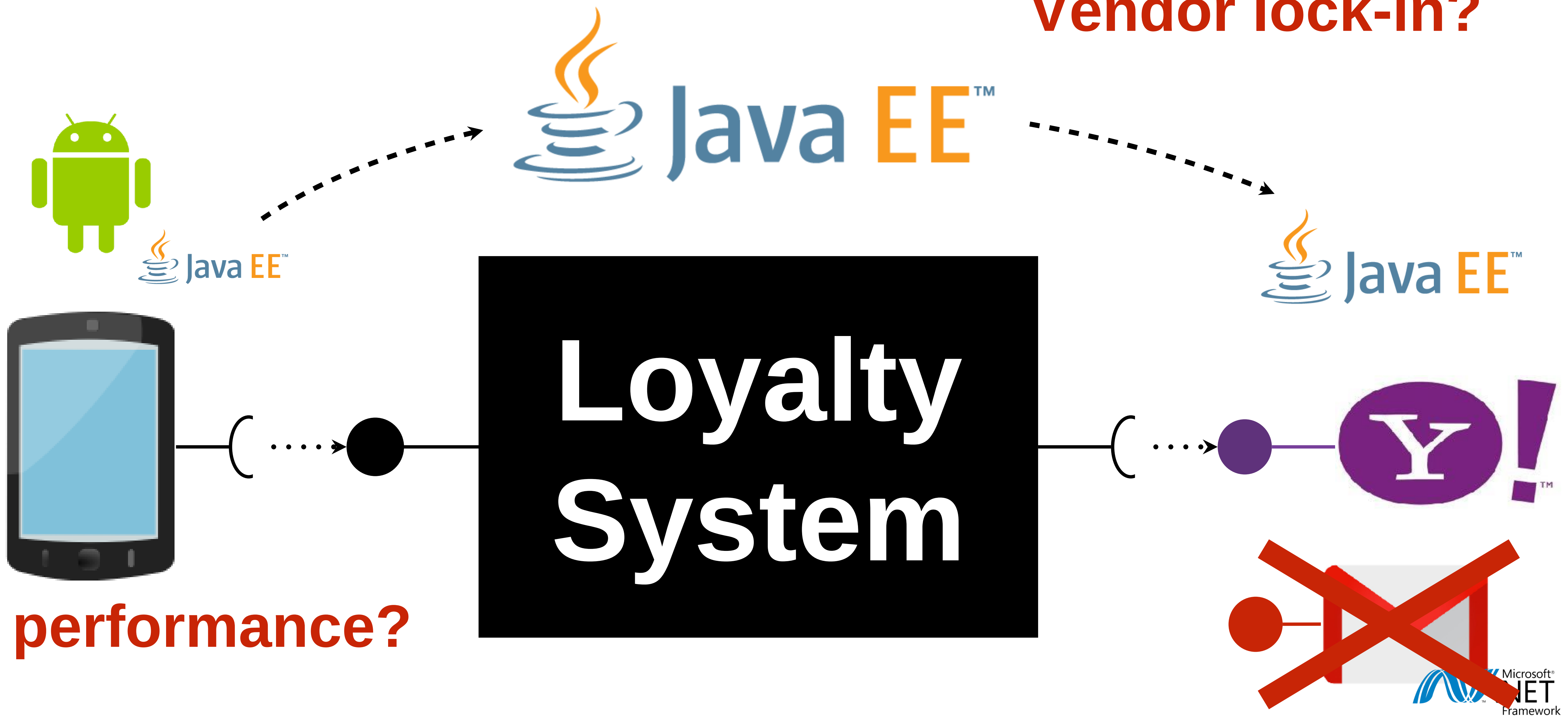
https://github.com/polytechnice-si/4A_ISA_TheCookieFactory/blob/develop/chapters/Exposing_SOAP.md

https://github.com/polytechnice-si/4A_ISA_TheCookieFactory/blob/develop/chapters/Consuming_REST.md



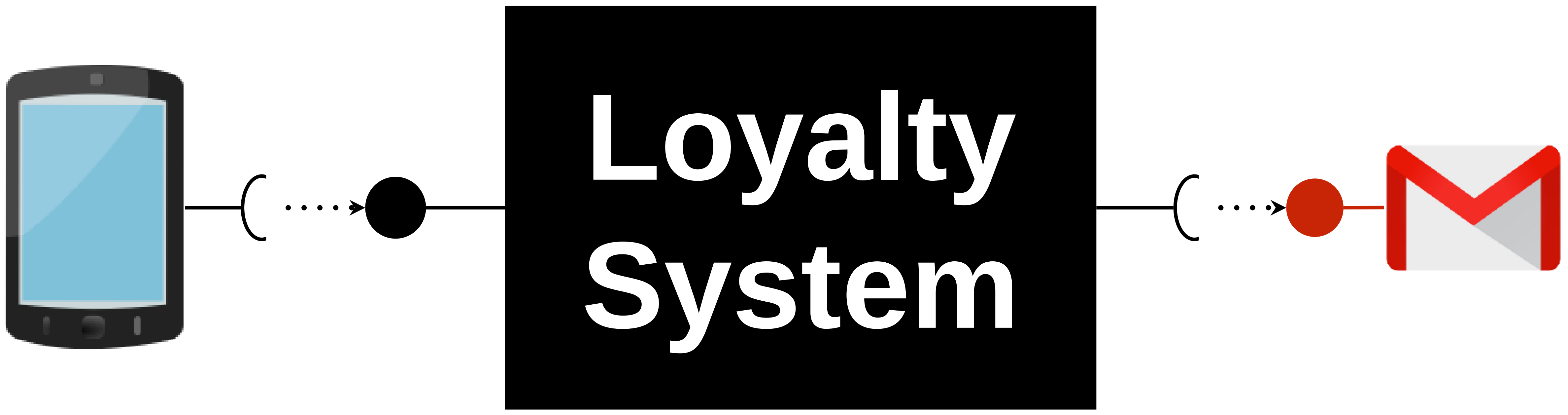
Public APIs support flexibility

Using J2E dependency injection



technology driving business?

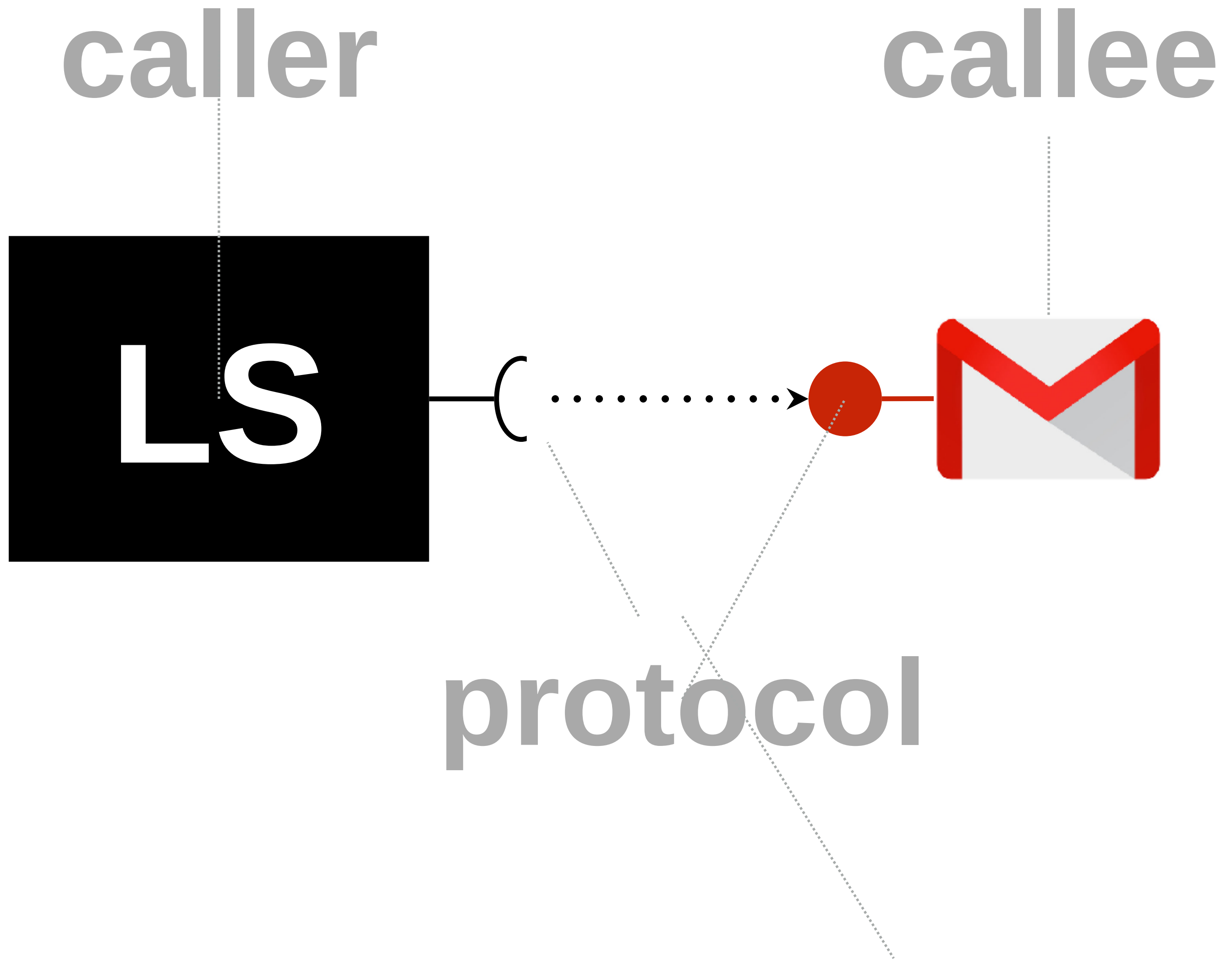
Homogeneous System



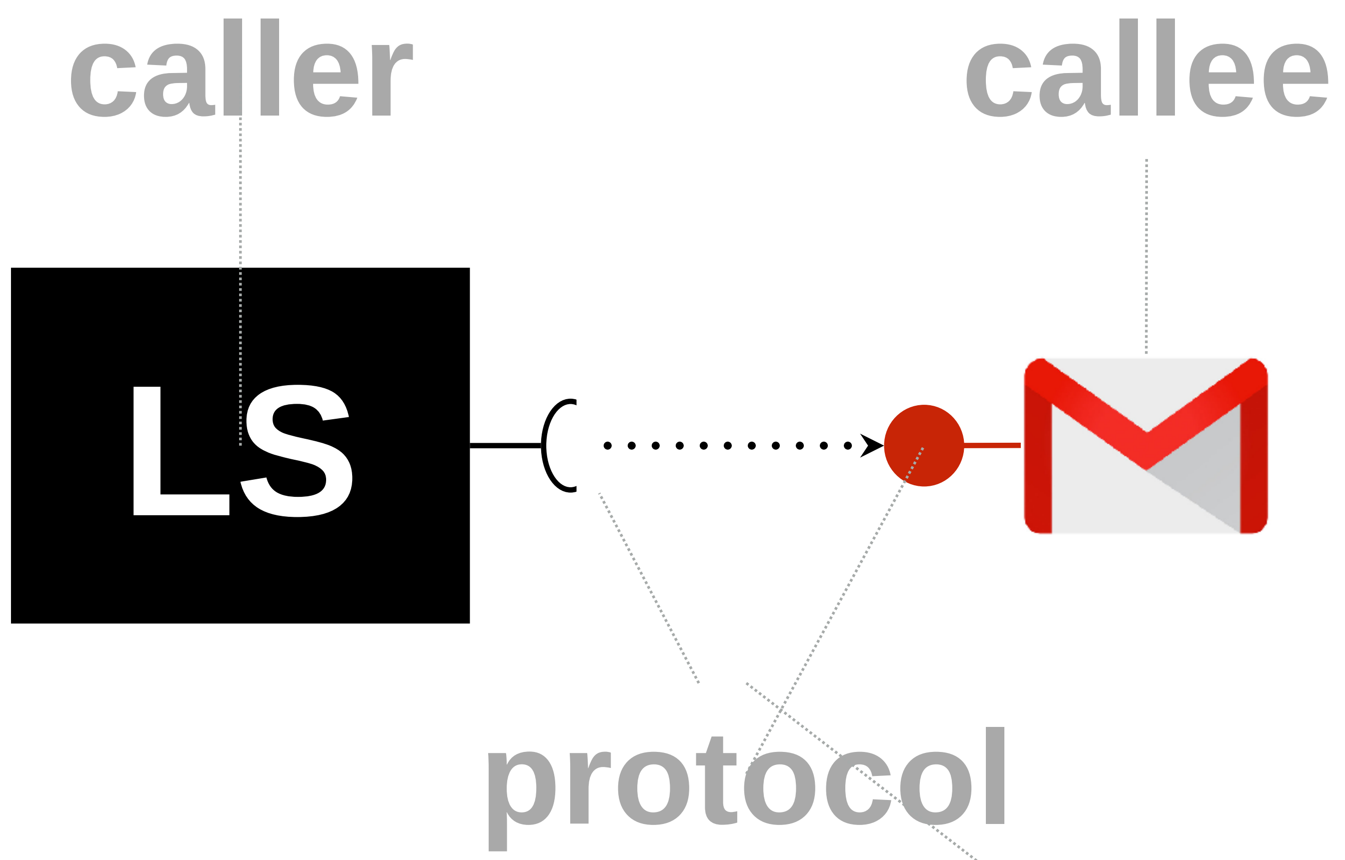
Interoperability ?

Heterogeneous System

Abstracting from Implementation



- Endpoint: Where, How ?
- Operations: Why ?
- Business Object: What ?



Messaging:

sendMail(data: Message)

Defined in the interface

- **Endpoint: Where, How ?**

- **Operations: Why ?**
- **Business Object: What ?**

Endpoint

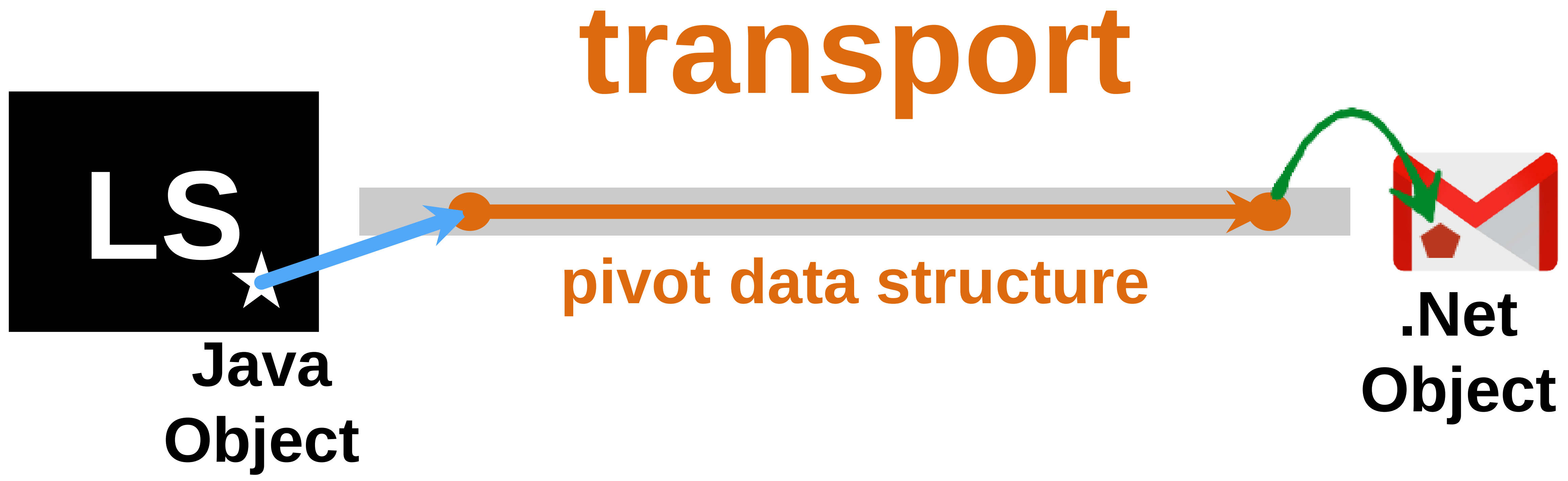
- **Where**:
  - IP Address
  - hostname (resolved to IP)
- **How**:
  - Communication protocol (e.g., HTTP)
  - Data Encoding (e.g., XML, JSON)

Platform Independent

APPROVED

marshalling:
Object → Pivot

unmarshalling:
Pivot → Object



REST vs SOAP

SOAP ➔
exposes **procedures** (*aka Remote Procedure Call*, RPC)

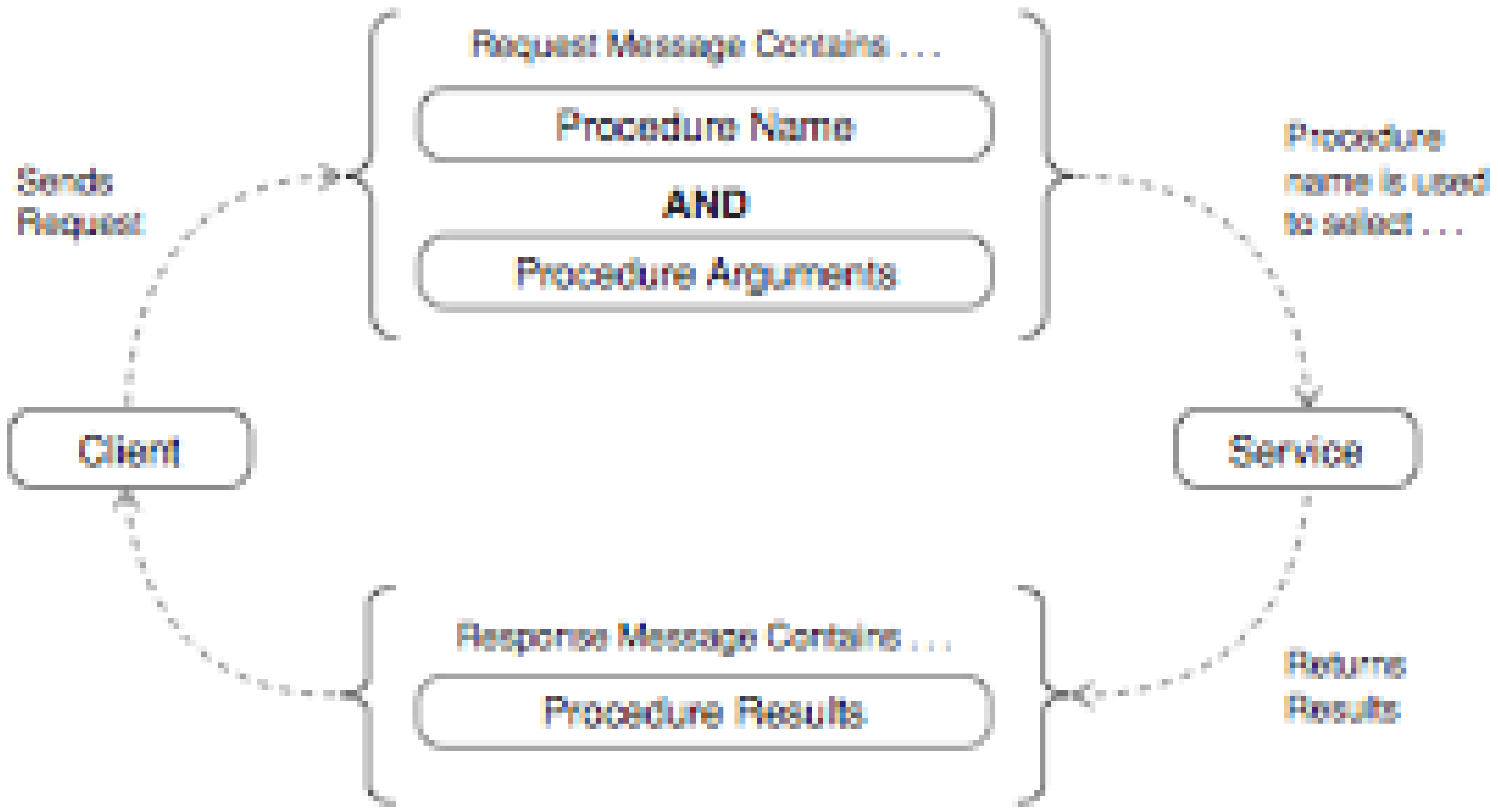
REST ➔
exposes **resources** (*i.e.*, nouns instead of verbs).

Contracts and style

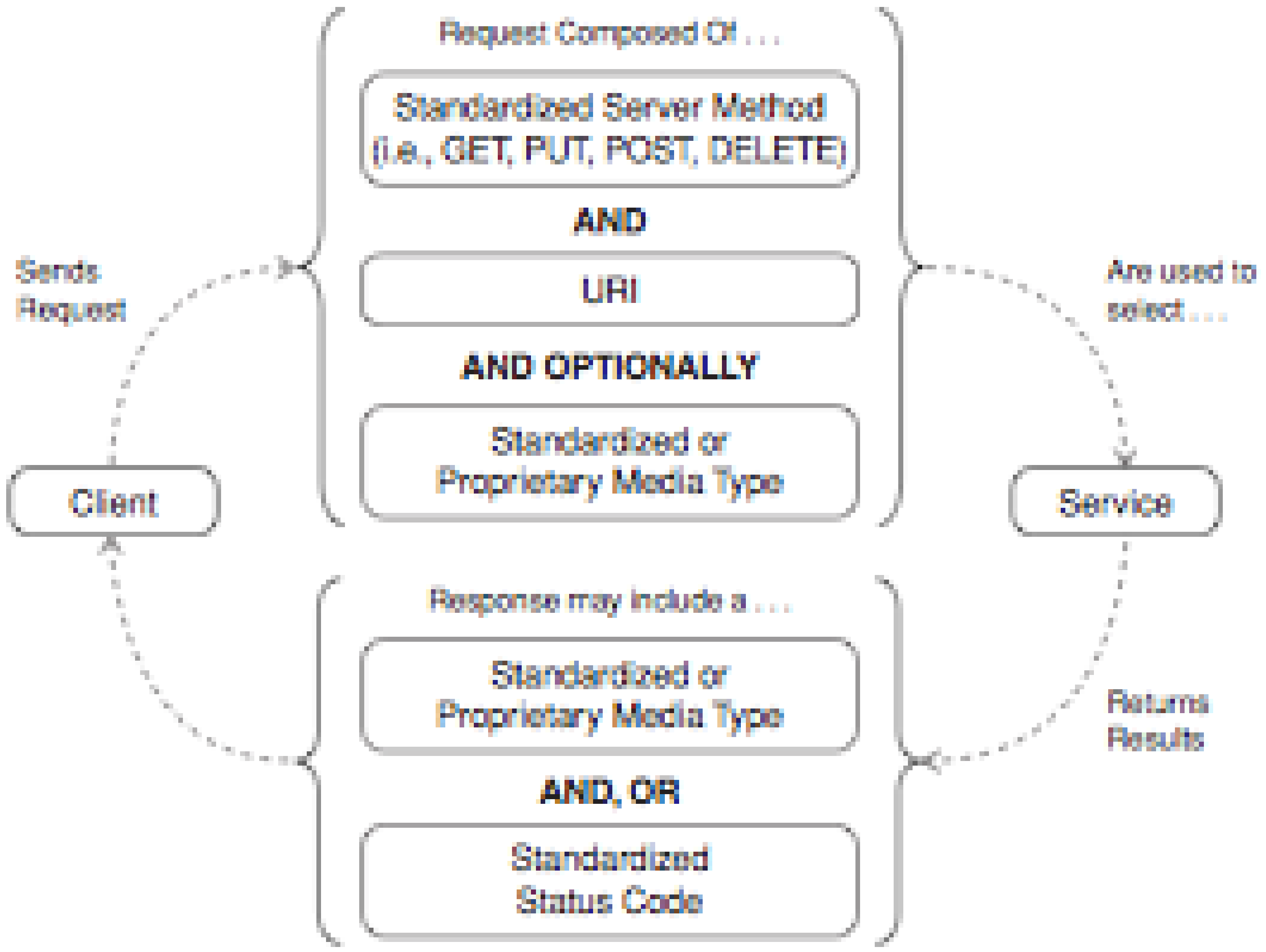
Exposing Resources (Nouns)

Exposing Operations (Verbs)

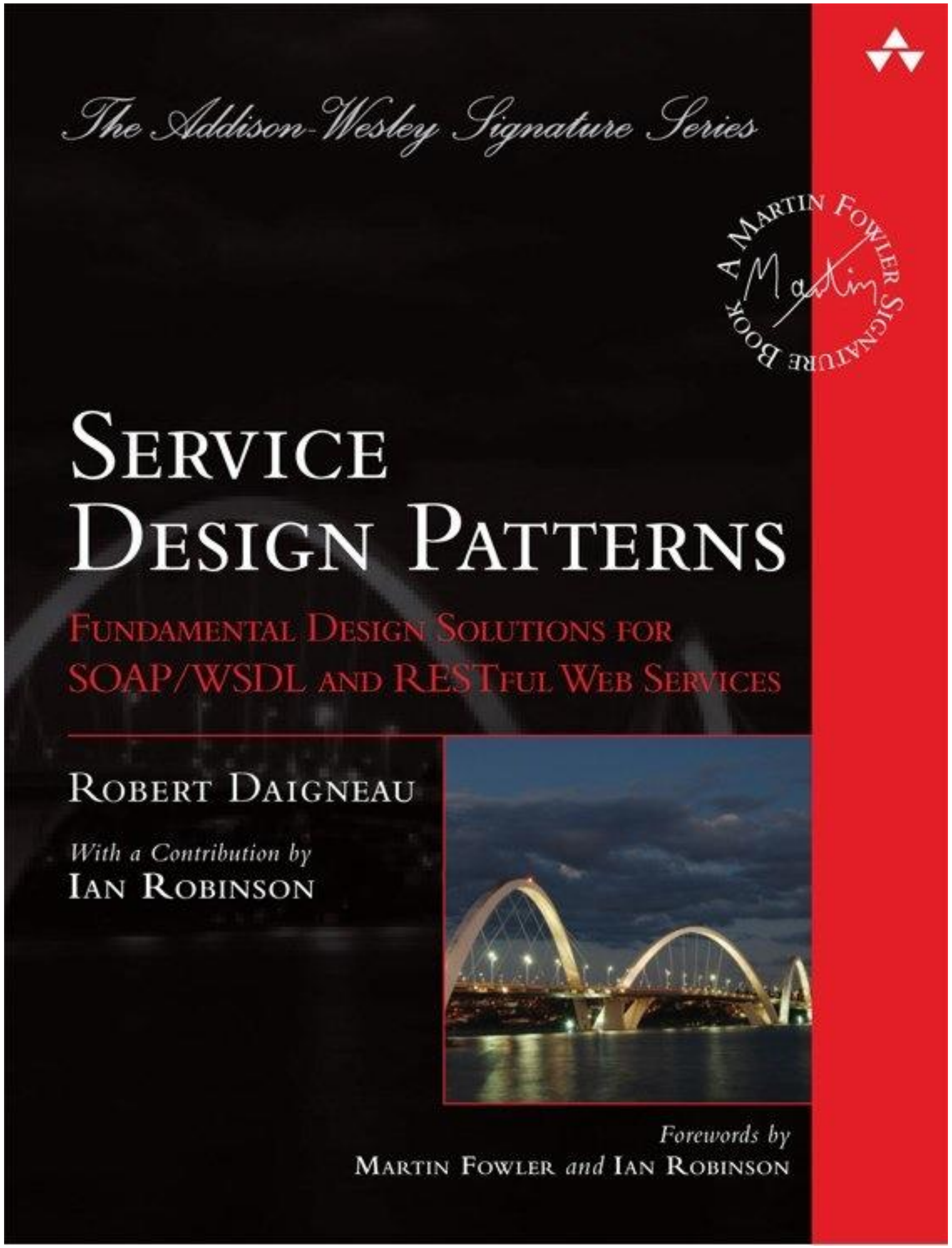
RPC Interaction Protocol


Request Message Contains . . .
Procedure Name
AND
Procedure Arguments
Sends Request
Procedure name is used to select . . .
Client
Service
Response Message Contains . . .
Procedure Results
Returns Results

Resource Interaction Protocol


Request Composed Of . . .
Standardized Server Method (i.e., GET, PUT, POST, DELETE)
AND
URI
AND OPTIONALLY
Standardized or Proprietary Media Type
Sends Request
Are used to select . . .
Client
Service
Response may include a . . .
Standardized or Proprietary Media Type
AND, OR
Standardized Status Code
Returns Results

[Service Design Patterns]



The Addison-Wesley Signature Series

A Martin Fowler Signature Book

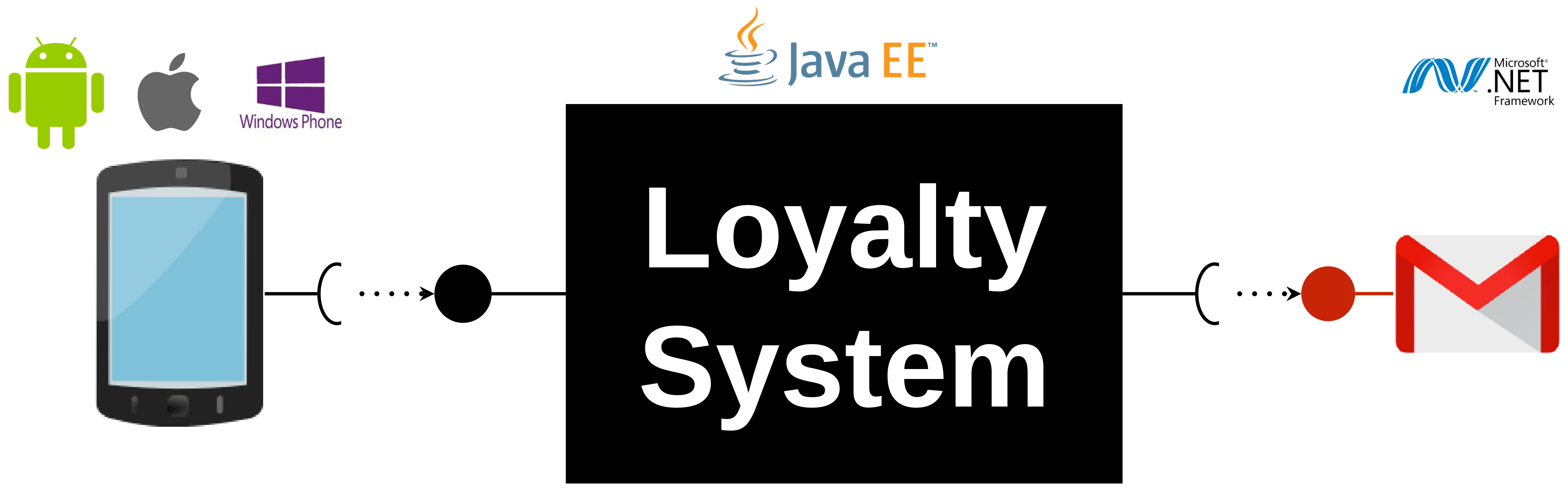
Service Design Patterns

Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services

Robert Daigneau

With a Contribution by
Ian Robinson

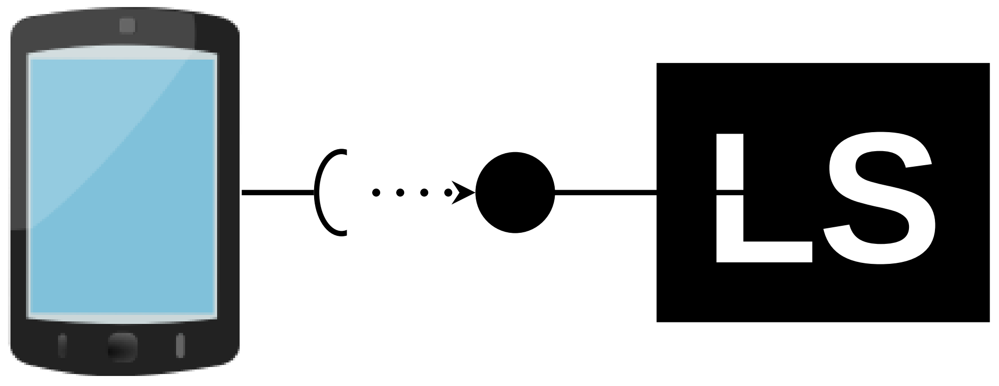
Forewords by
Martin Fowler *and* Ian Robinson



Public APIs support interoperability

Strong Contract

Expose Web services (SOAP)



Comment exposer les services aux clients distants ?

EJBs : remote EJBs,
➔implies for the clients to be J2E-compliant,

EJBs as a Web Service.

➔clients to be developed in any language.

Operations exposed by the web service
⇎ the associated bean,

can combine multiple beans

The Web Service layer is the *public API* of our architecture.

Web services constraints

Web services are stateless (WS standard)

➔ any beans exposed must be stateless.

Business objects exposed must :
be serializable, define an empty constructor and get/set methods

The TomEE container must include the software stack that implement Web Service (the TomEE+ version of the server)

Web service contract

defined by an annotated interface.

1. A WebMethod annotation tags the methods to expose as service operations

2. A WebParam annotation tags the parameters to change their name, or handle xml namespace manually

3. A WebResult annotation tags the returned value, like @WebParam

4. A WebService annotation is used to specify the *namespace* associated to the service

Remote client for the service

1. Load the Contract of the service, exposed using the *Web Service Description Language* (WSDL)

2. Generate the Java code that will support the interactions with the service.

Keystone

Contracts are reified into shared artefacts, and used by tools instead of humans

Standard ⇒ No freedom

Standard ⇒ Automation

Why should we **write piece of codes** instead of **being lazy** and write **pieces of code that will write pieces of code** on our behalf

- Jean-Marc Jézéquel **?**

Contract

```
@WebService
public interface CartWebService {

  @WebMethod
  void addItemToCustomerCart(
         @WebParam(name = "customer_name") String customerName,
         @WebParam(name = "item") Item it
       ) throws UnknownCustomerException;

   @WebMethod
   void removeItemToCustomerCart(
           @WebParam(name = "customer_name") String customerName,
           @WebParam(name = "item") Item it
         ) throws UnknownCustomerException;

   @WebMethod
   @WebResult(name = "cart_contents")
   Set<Item> getCustomerCartContents(
                @WebParam(name = "customer_name") String customerName
              ) throws UnknownCustomerException;

   @WebMethod
   @WebResult(name = "order_id")
   String validate(@WebParam(name = "customer_name") String customerName)
          throws PaymentException, UnknownCustomerException,
                  EmptyCartException;
}
```

Interface

Pivot Interface Description

Compilation process
Interface → Pivot

SOAP standard

```
@WebMethod
void addItemToCustomerCart(
       @WebParam(name = "customer_name") String customerName,
       @WebParam(name = "item") Item it
     ) throws UnknownCustomerException;
```

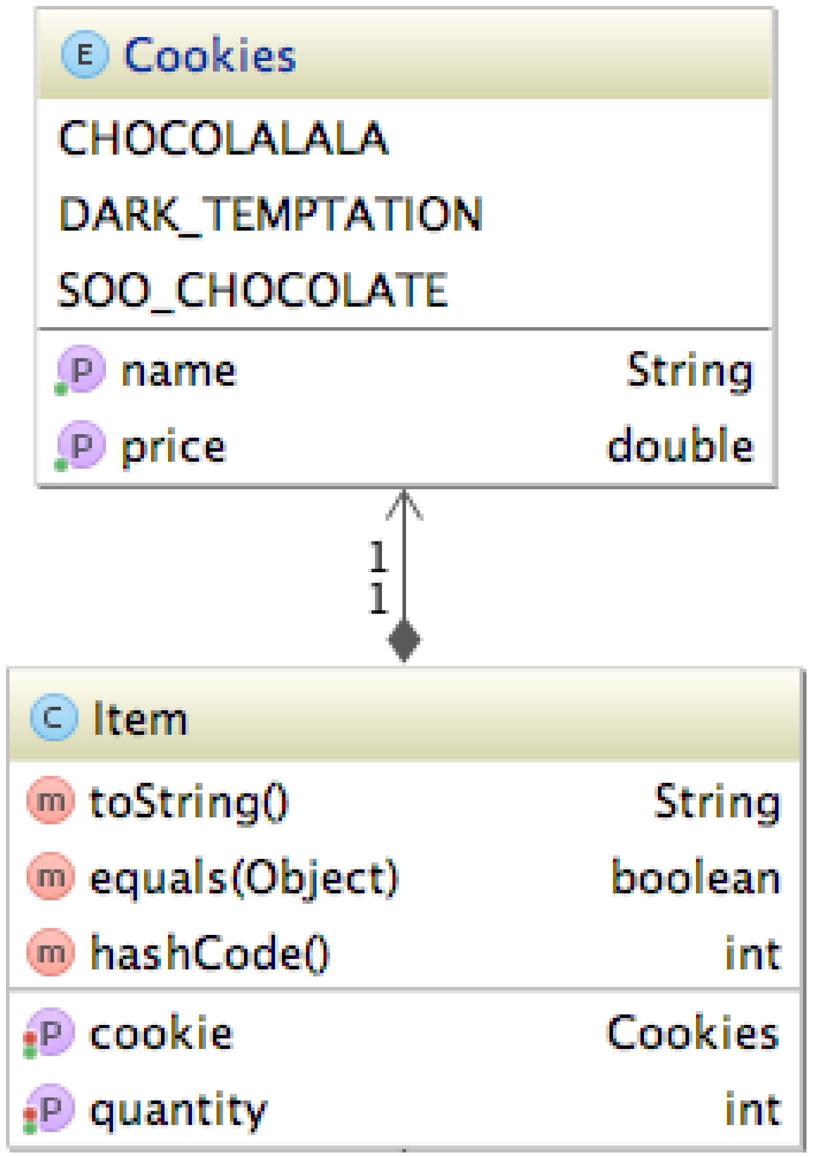
↓ **Java2WSDL**

```
<wsdl:portType name="CartWebService"><wsdl:operation name="addItemToCustomerCart">

    <wsdl:input message="ns1:addItemToCustomerCart"

                name="addItemToCustomerCart" />

    <wsdl:output message="ns1:addItemToCustomerCartResponse"

                 name="addItemToCustomerCartResponse"/>

    <wsdl:fault message="ns1:UnknownCustomerException"

                name="UnknownCustomerException" />

  </wsdl:operation>

  ...

</wsdl:portType>
```

Web Service Description Language

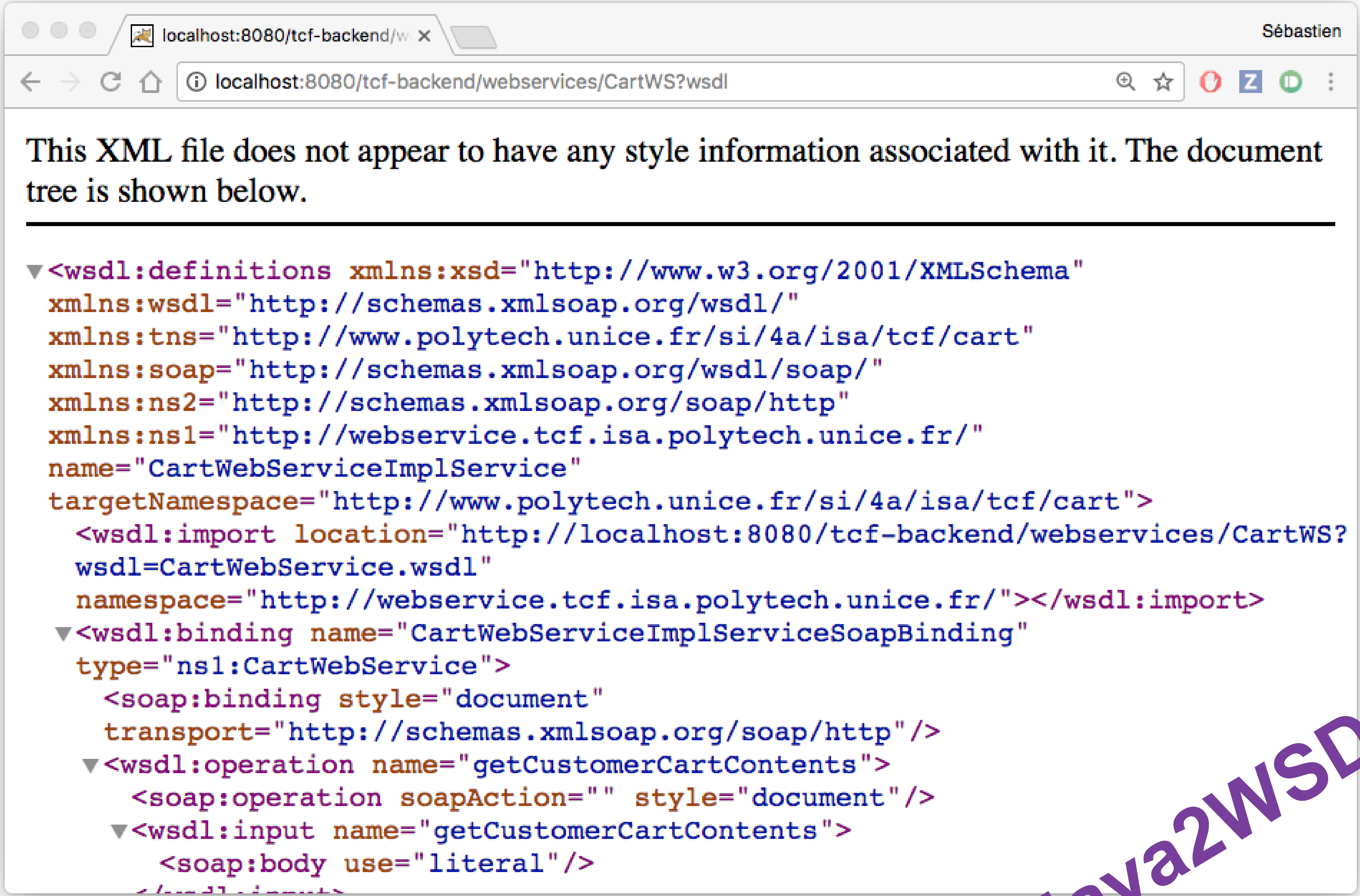
XSD for data structures

```
<xs:complexType name="item">

  <xs:sequence>

    <xs:element minOccurs="0"

                name="cookie"

                type="tns:cookies"/>

    <xs:element name="quantity"

                type="xs:int"/>

  </xs:sequence>

</xs:complexType>
<xs:simpleType name="cookies">

  <xs:restriction base="xs:string">

    <xs:enumeration value="CHOCOLALALA"/>

    <xs:enumeration value="DARK_TEMPTATION"/>

    <xs:enumeration value="SOO_CHOCOLATE"/>

  </xs:restriction>

</xs:simpleType>
```



e.g., Business objects, Messages

Automated Generation & Exposition

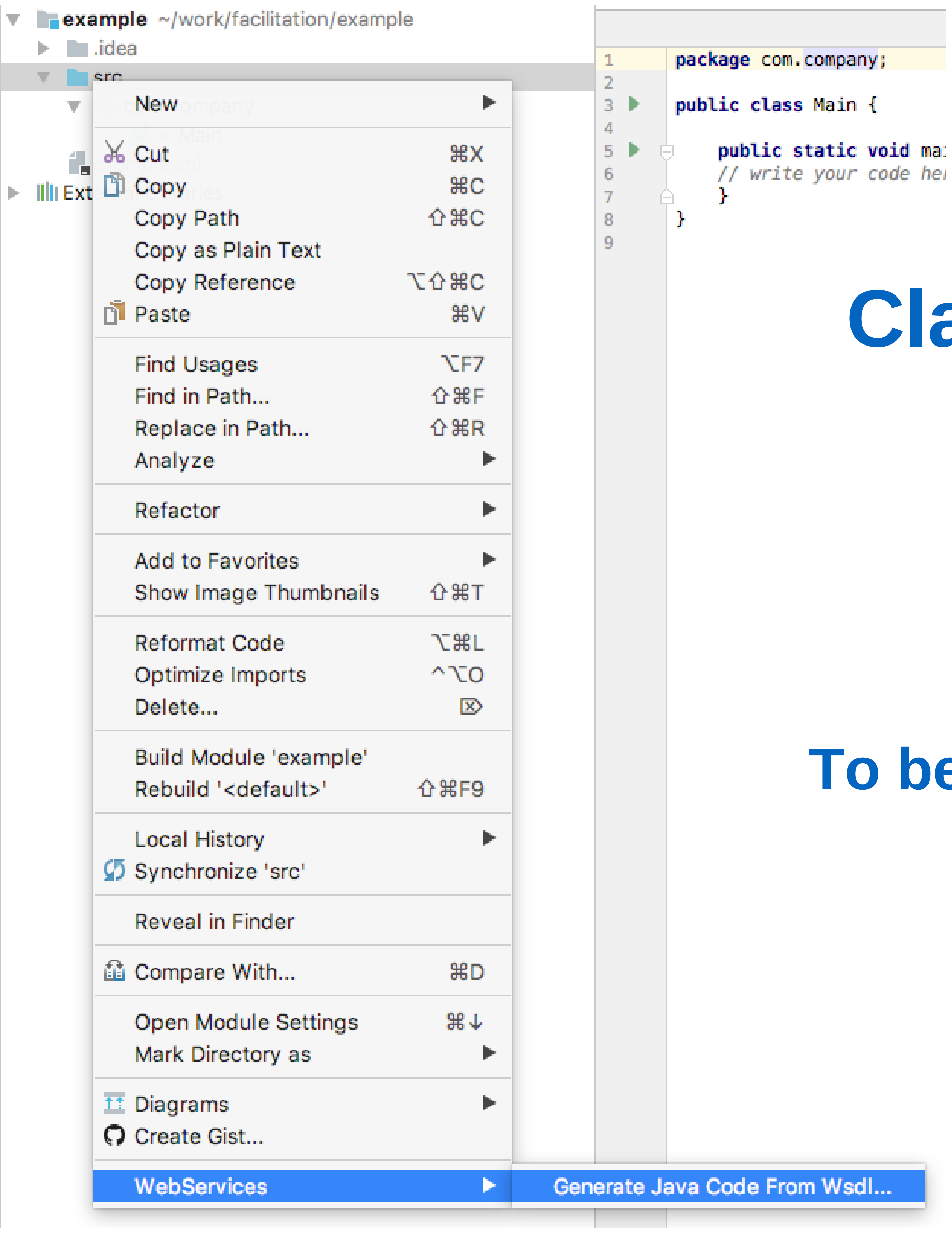


localhost:8080/tcf-backend/w
Sébastien
localhost:8080/tcf-backend/webservices/CartWS?wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:tns="http://www.polytech.unice.fr/si/4a/isa/tcf/cart"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:ns2="http://schemas.xmlsoap.org/soap/http"
 xmlns:ns1="http://webservice.tcf.isa.polytech.unice.fr/"
 name="CartWebServiceImplService"
 targetNamespace="http://www.polytech.unice.fr/si/4a/isa/tcf/cart">
   <wsdl:import location="http://localhost:8080/tcf-backend/webservices/CartWS?
   wsdl=CartWebService.wsdl"
   namespace="http://webservice.tcf.isa.polytech.unice.fr/"></wsdl:import>
 ▼<wsdl:binding name="CartWebServiceImplServiceSoapBinding"
   type="ns1:CartWebService">
     <soap:binding style="document"
     transport="http://schemas.xmlsoap.org/soap/http"/>
   ▼<wsdl:operation name="getCustomerCartContents">
       <soap:operation soapAction="" style="document"/>
     ▼<wsdl:input name="getCustomerCartContents">
         <soap:body use="literal"/>
```

Java2WSDL



```
package com.company;

public class Main {

    public static void ma
    // write your code he
    }
}
```

Classical (as in Standard) Code generator

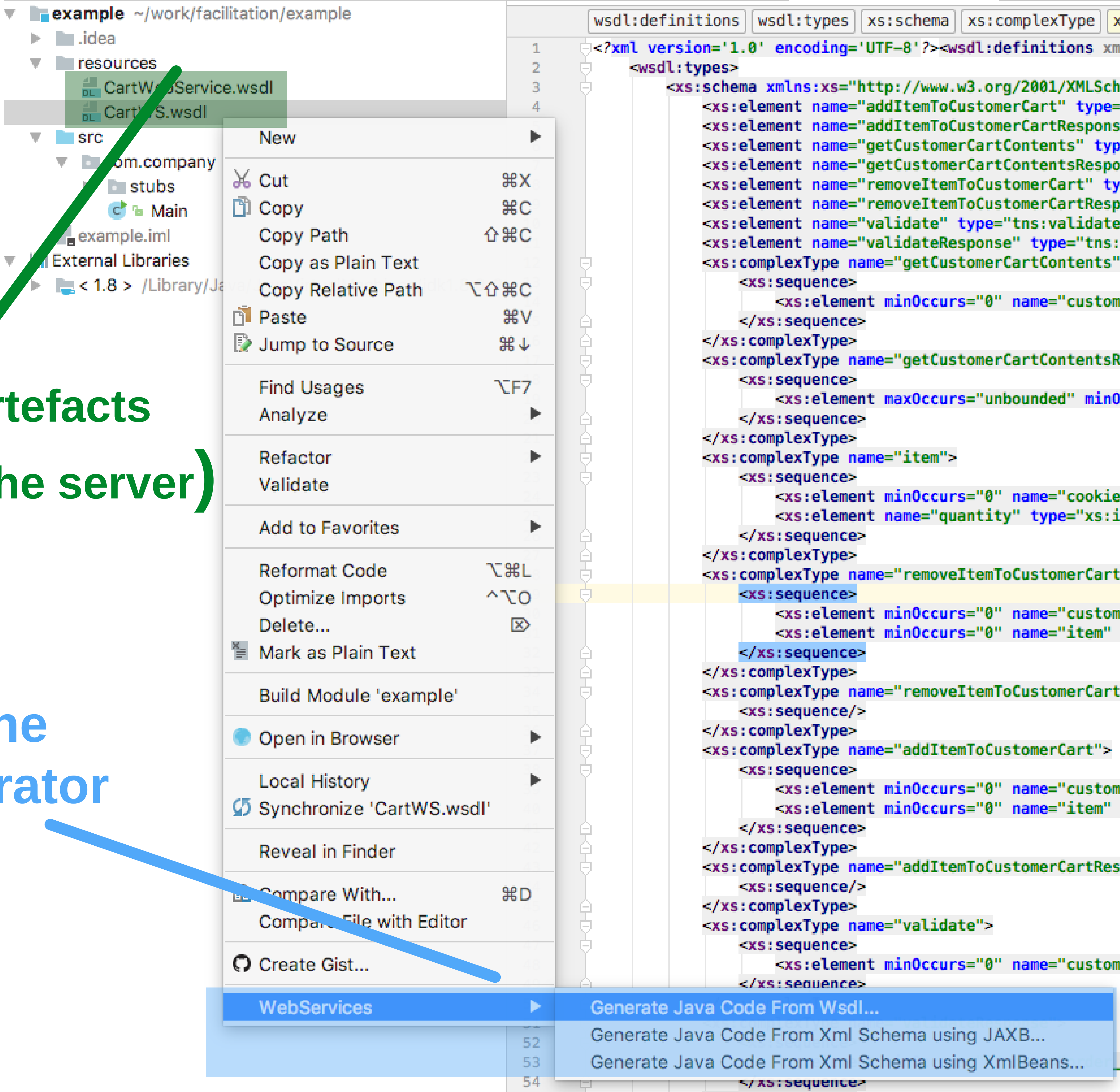
To be called on client side (obviously)

Here IntelliJ Ultimate

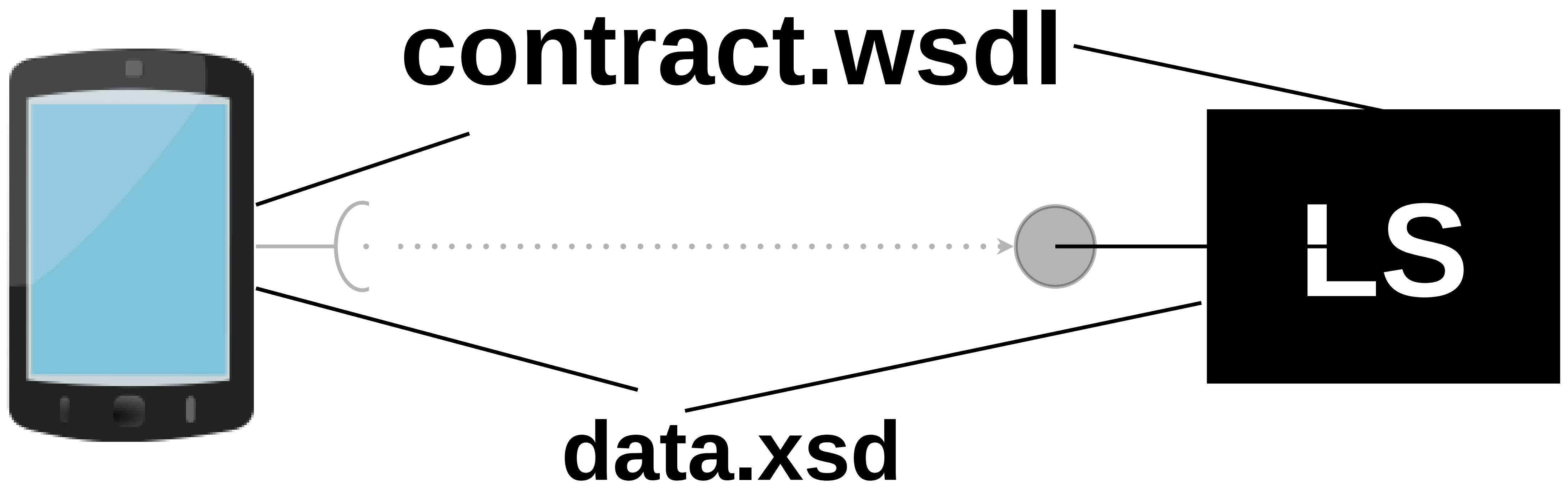
Contract artefacts

(shared with the server)

Calling the code generator



```
<?xml version='1.0' encoding='UTF-8'?><wsdl:definitions xm
    <wsdl:types>
        <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSch
            <xs:element name="addItemToCustomerCart" type=
            <xs:element name="addItemToCustomerCartRespons
            <xs:element name="getCustomerCartContents" typ
            <xs:element name="getCustomerCartContentsRespo
            <xs:element name="removeItemToCustomerCart" ty
            <xs:element name="removeItemToCustomerCartResp
            <xs:element name="validate" type="tns:validate
            <xs:element name="validateResponse" type="tns:
            <xs:complexType name="getCustomerCartContents"
                <xs:sequence>
                    <xs:element minOccurs="0" name="custom
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="getCustomerCartContentsR
                <xs:sequence>
                    <xs:element maxOccurs="unbounded" minO
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="item">
                <xs:sequence>
                    <xs:element minOccurs="0" name="cookie
                    <xs:element name="quantity" type="xs:i
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="removeItemToCustomerCart
                <xs:sequence>
                    <xs:element minOccurs="0" name="custom
                    <xs:element minOccurs="0" name="item"
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="removeItemToCustomerCart
                <xs:sequence/>
            </xs:complexType>
            <xs:complexType name="addItemToCustomerCart">
                <xs:sequence>
                    <xs:element minOccurs="0" name="custom
                    <xs:element minOccurs="0" name="item"
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="addItemToCustomerCartRes
                <xs:sequence/>
            </xs:complexType>
            <xs:complexType name="validate">
                <xs:sequence>
                    <xs:element minOccurs="0" name="custom
                </xs:sequence>
```



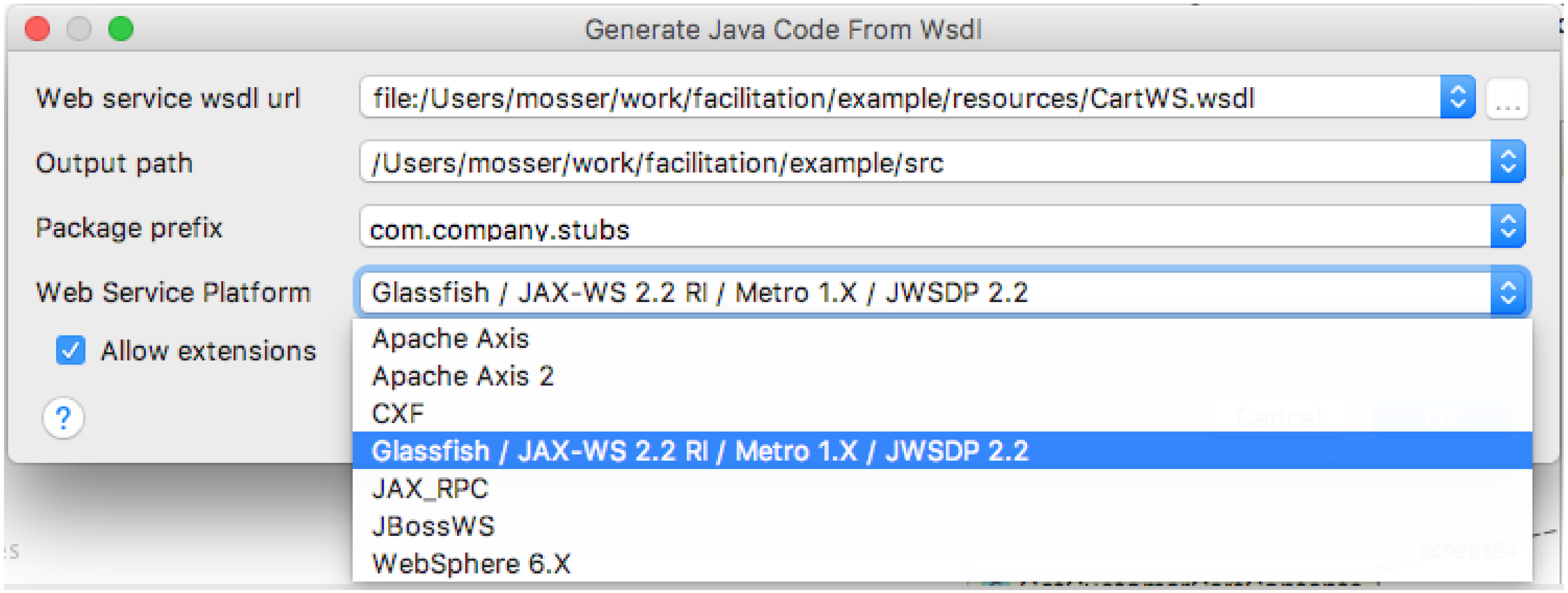
Data structures using the Xxx language

↑

WSDL2Xxx

↓

(Un)Marshalling using the Xxx language



Generate Java Code From Wsdl

Web service wsdl url: file:/Users/mosser/work/facilitation/example/resources/CartWS.wsdl

Output path: /Users/mosser/work/facilitation/example/src

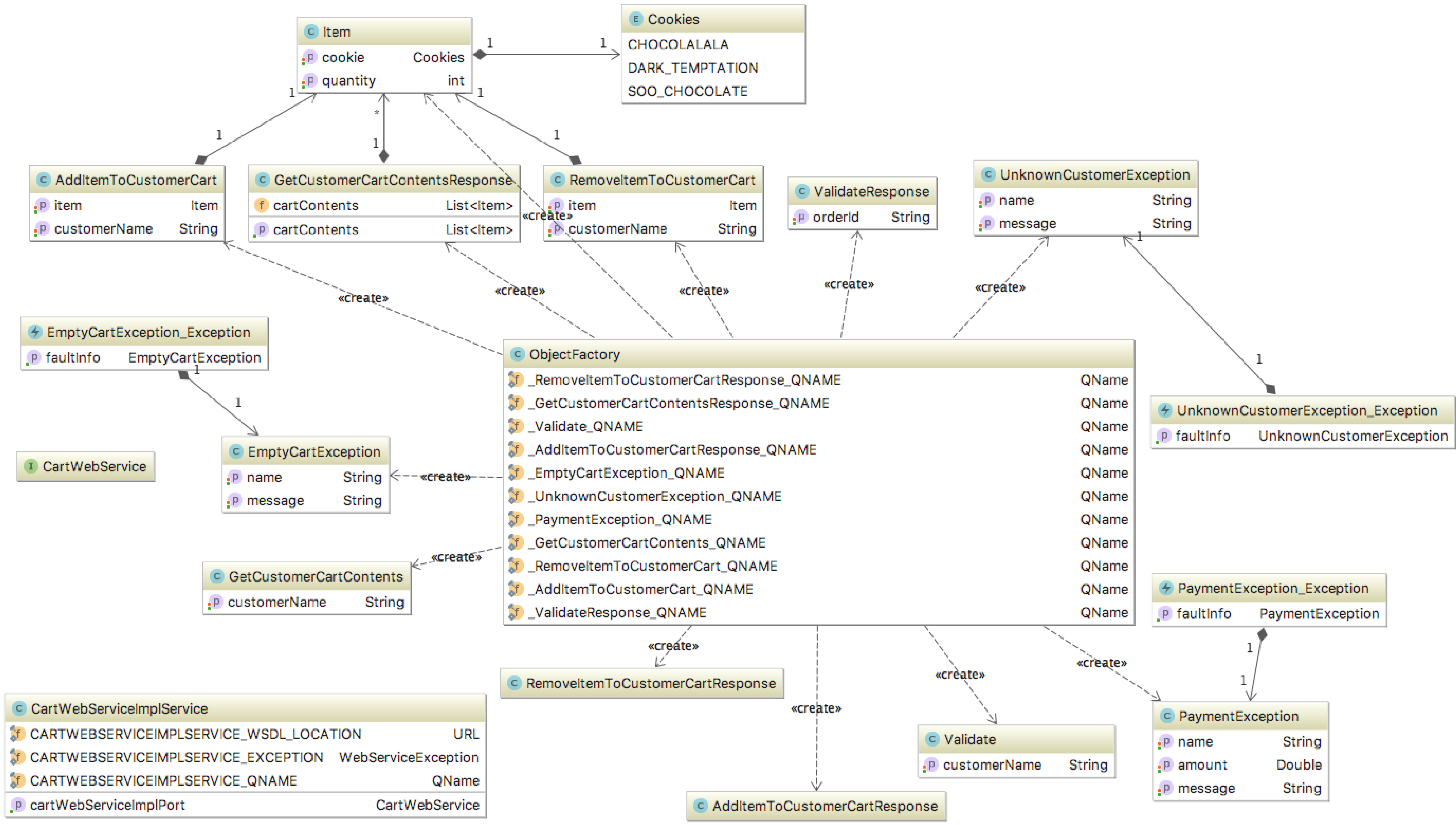
Package prefix: com.company.stubs

Web Service Platform: Glassfish / JAX-WS 2.2 RI / Metro 1.X / JWSDP 2.2

[✓] Allow extensions

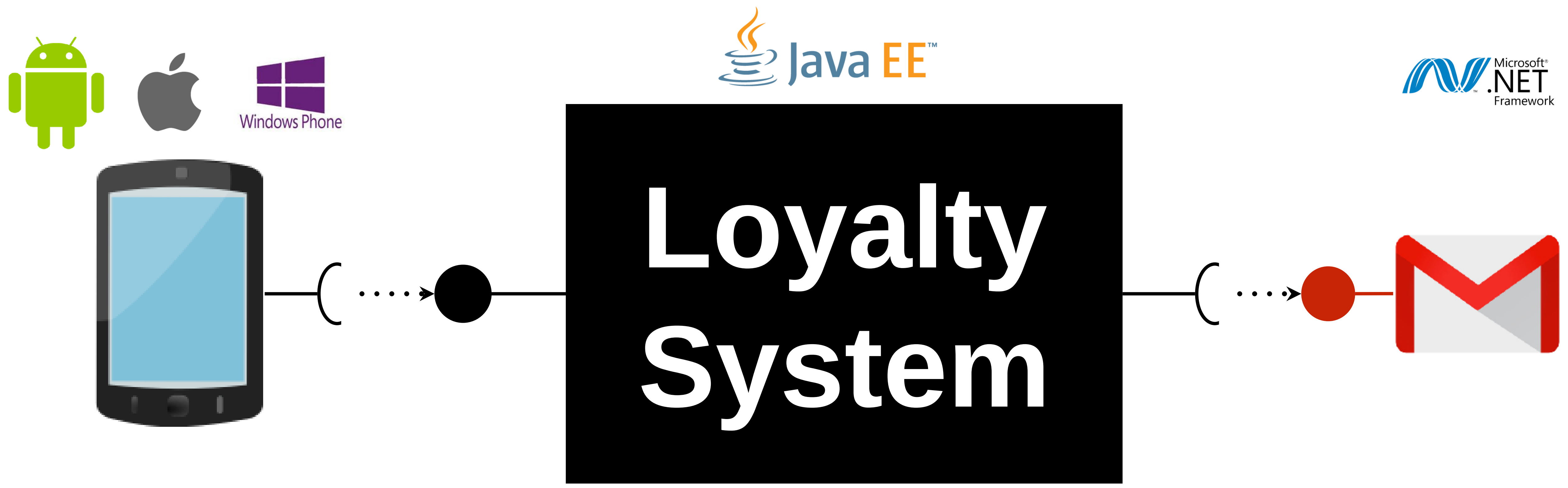
Standard $\nRightarrow$ single implementation

Generated Code!


Item
cookie Cookies
quantity int
Cookies
CHOCOLALALA
DARK_TEMPTATION
SOO_CHOCOLATE
AddItemToCustomerCart
item Item
customerName String
GetCustomerCartContentsResponse
cartContents List<Item>
cartContents List<Item>
RemoveItemToCustomerCart
item Item
customerName String
ValidateResponse
orderId String
UnknownCustomerException
name String
message String
EmptyCartException_Exception
faultInfo EmptyCartException
ObjectFactory
_RemoveItemToCustomerCartResponse_QNAME QName
_GetCustomerCartContentsResponse_QNAME QName
_Validate_QNAME QName
_AddItemToCustomerCartResponse_QNAME QName
_EmptyCartException_QNAME QName
_UnknownCustomerException_QNAME QName
_PaymentException_QNAME QName
_GetCustomerCartContents_QNAME QName
_RemoveItemToCustomerCart_QNAME QName
_AddItemToCustomerCart_QNAME QName
_ValidateResponse_QNAME QName
UnknownCustomerException_Exception
faultInfo UnknownCustomerException
CartWebService
EmptyCartException
name String
message String
GetCustomerCartContents
customerName String
PaymentException_Exception
faultInfo PaymentException
RemoveItemToCustomerCartResponse
CartWebServiceImplService
CARTWEBSERVICEIMPLSERVICE_WSDL_LOCATION URL
CARTWEBSERVICEIMPLSERVICE_EXCEPTION WebServiceException
CARTWEBSERVICEIMPLSERVICE_QNAME QName
cartWebServiceImplPort CartWebService
Validate
customerName String
PaymentException
name String
amount Double
message String
AddItemToCustomerCartResponse
«create»
1
*

Consuming a service == sending messages to objects

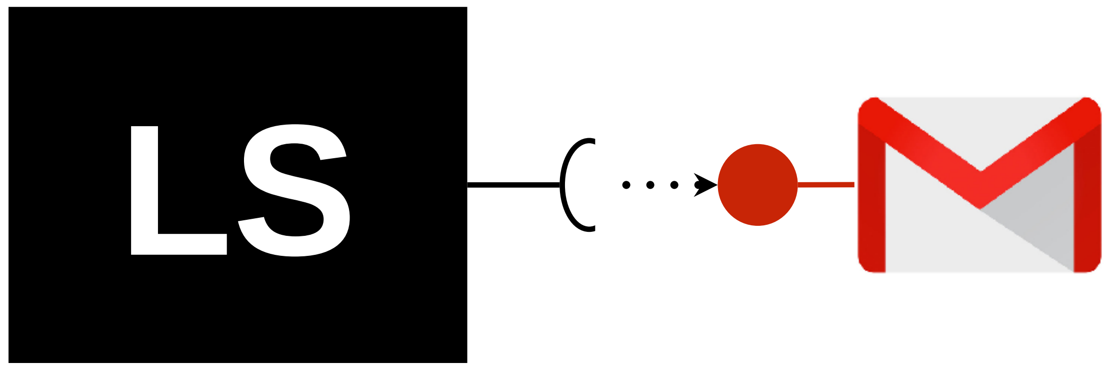
```java
public static void main(String[] args) throws Exception {
  System.out.println("#### Instantiating the WS Proxy");
  CartWebServiceImplService factory = new CartWebServiceImplService();
  CartWebService ws = factory.getCartWebServiceImplPort();

  List<Item> cart = ws.getCustomerCartContents("john");
  System.out.println("Cart is empty: " + cart.isEmpty());

  Item i = new Item();
  i.setCookie(Cookies.CHOCOLALALA); i.setQuantity(3);
  ws.addItemToCustomerCart("john", i);
  i.setCookie(Cookies.DARK_TEMPTATION); i.setQuantity(2);
  ws.addItemToCustomerCart("john", i);
  i.setCookie(Cookies.CHOCOLALALA); i.setQuantity(4);
  ws.addItemToCustomerCart("john", i);

  cart = ws.getCustomerCartContents("john");
  System.out.println("John's cart: " +cart);
}
```



Public APIs support interoperability

Light | No Contract

Consume Web services (REST)



Invoking a REST service

The J2E system is client of the service.

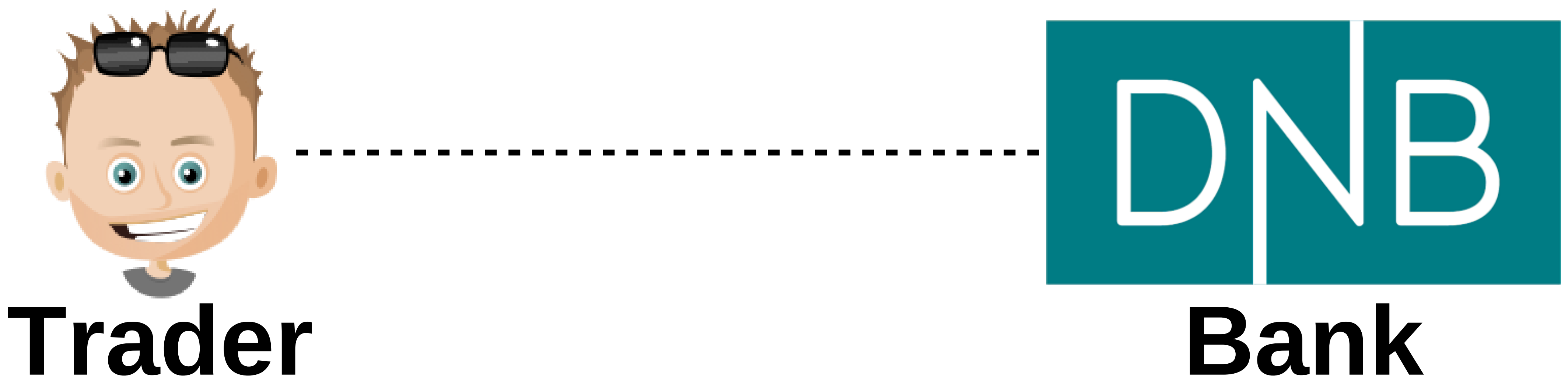
To consume REST web services :

- implement the methods that support the communication with the service in a utility class
- interact with the remote service.

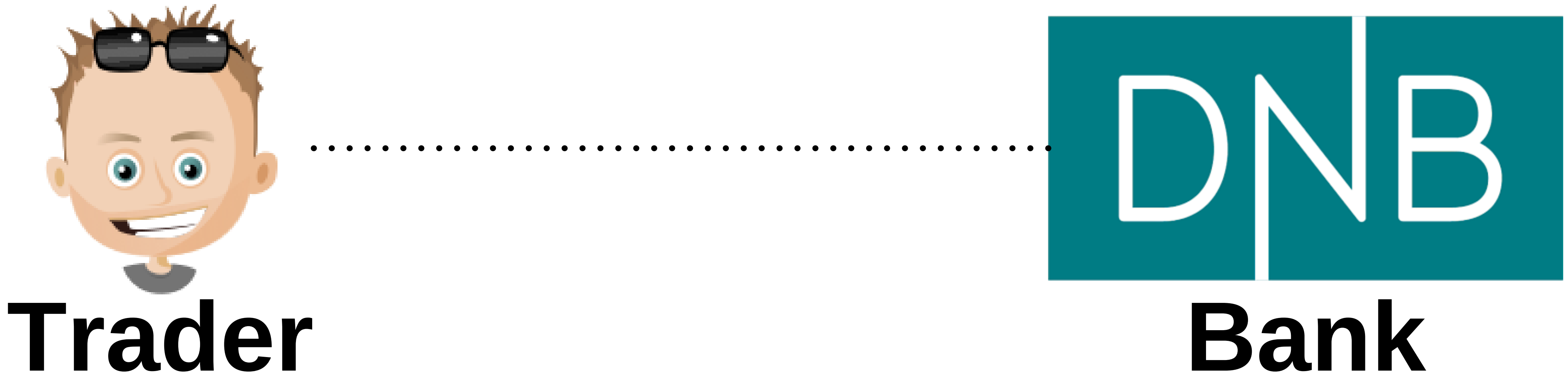
Locate the remote service

define a XXX.properties file in the resources directory, which will defined the endpoint : hostname and port number to be used when interacting with the XXX service

@PostConstruct annotation to load these properties from the resource file after the bean initialization



- Operations :
  - Send a payment request to be processed by the bank
  - Describe a given payment (status, …)
  - List all received payments from the calling trader
- Protocol:
  - Plain HTTP (GET, POST, …)
  - Data encoding using JSON



- Send a payment request to be processed by the bank
    - POST /mailbox { “card”: “123456780”, “amount”: 2.85 } ↦ 42
- Describe a given payment (status, …)
    - GET /payments/42 ↦ {“card”: “…”, “amount”: 2.85, “status”: “OK” , …}
- List all received payments from the calling trader
    - GET /payments ↦ [“42”, “24”, …]

REST et Create / Read / Update / Delete

Customers:
  POST /customers
  GET /customers/{id}
  GET /customers/{id}/orders
  PUT /customers/{id}
  DELETE /customers/{id}

Orders:
  POST /orders
  GET /orders/{id}
  PUT /orders/{id}
  DELETE /orders/{id}

Items …

CRUD services oriented database as a service kind of thinking

1. "Services" - business logic.
2. thought-out contract.
3. Do not going straight to their data.
4. A service is not equivalent to data source.
5. minuscule services instead of business distributed services.

http://www.vinaysahni.com/best-practices-for-a-prag

Describing the Business Objects

```
[DataContract(Namespace = "http://partner/external/payment/data/",
              Name = "PaymentRequest")]
public class PaymentRequest
{
  [DataMember]
  public string CreditCard { get; set; }

  [DataMember]
  public double Amount { get; set; }
}
```

No methods. Structure only.

Describing the Interface

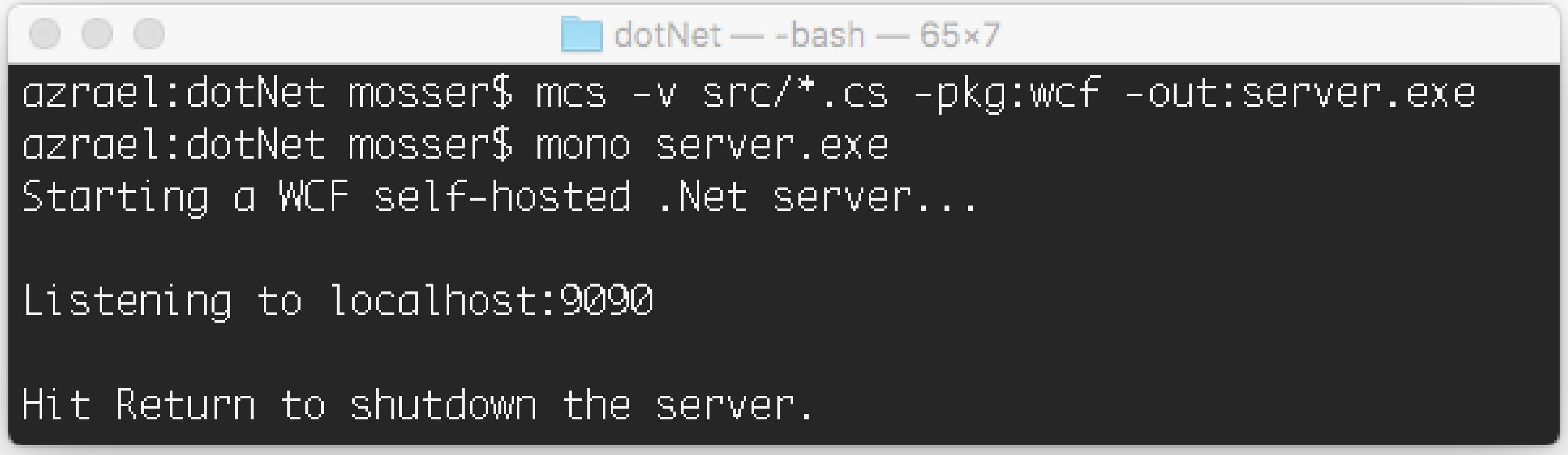
```
[ServiceContract]
public interface IPaymentService
{

  [OperationContract]
  [WebInvoke( Method = "POST", UriTemplate = "mailbox",
              RequestFormat = WebMessageFormat.Json,
              ResponseFormat = WebMessageFormat.Json)]
  int ReceiveRequest(PaymentRequest request);

  [OperationContract]
  [WebInvoke( Method = "GET", UriTemplate = "payments/{identifier}",
              ResponseFormat = WebMessageFormat.Json)]
  Payment FindPaymentById(int identifier);

  [OperationContract]
  [WebInvoke( Method = "GET", UriTemplate = "payments",
              ResponseFormat = WebMessageFormat.Json)]
  List<int> GetAllPaymentIds();
```

Implementing the service

```
public class PaymentService : IPaymentService
{

   public int ReceiveRequest(PaymentRequest request)
   {
      Console.WriteLine("ReceiveRequest: " + request);
      var payment = BuildPayment(request);
      accounts.Add(counter, payment);
      return counter;
   }

   // ...
}
```

Mocked Implementation

Starting a self-hosted server



```
azrael:dotNet mosser$ mcs -v src/*.cs -pkg:wcf -out:server.exe
azrael:dotNet mosser$ mono server.exe
Starting a WCF self-hosted .Net server...

Listening to localhost:9090

Hit Return to shutdown the server.
```

```
public void start()
{
  Console.WriteLine("Starting a WCF self-hosted .Net server... ");
  string url = "http://" + "localhost" + ":" + Port;

  WebHttpBinding b = new WebHttpBinding();
  Host = new WebServiceHost(typeof(PaymentService), new Uri (url));

  // Adding the service to the host
  Host.AddServiceEndpoint(typeof(IPaymentService), b, "");

  // Staring the Host server
  Host.Open();
  Console.WriteLine("\nListening to " + "localhost" + ":" + Port + "\n");

  if ( Standalone ) { lockServer(); } else { interactive(); }

}
```

Plain HTTP communication

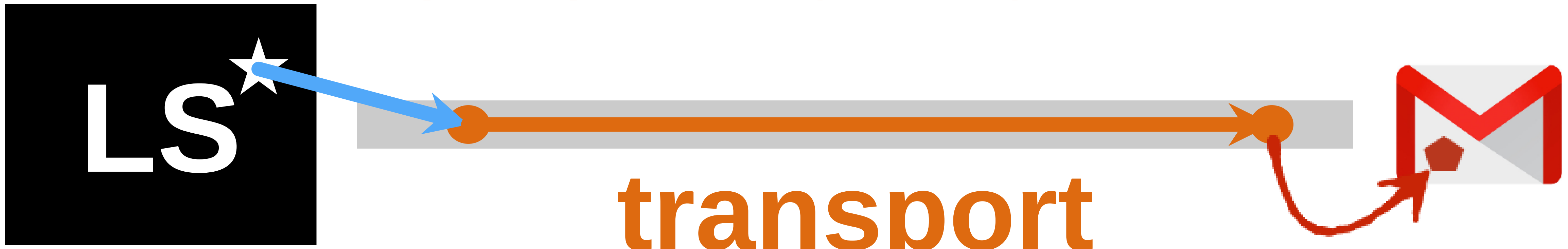
```
azrael:~ mosser$ REQUEST='{ "CreditCard": "1234-896983", "Amount": 12.09 }'
azrael:~ mosser$ BASE_URL="http://localhost:9090"
azrael:~ mosser$ HEADERS='Content-Type: application/json'
azrael:~ mosser$ curl -i -w "\n" -H "$HEADERS" \
                      -X POST -d “$REQUEST" \
                      $BASE_URL/mailbox
HTTP/1.1 200
Content-Type: application/json; charset=utf-8
Server: Mono-HTTPAPI/1.0
Date: Thu, 25 Feb 2016 09:24:41 GMT
Content-Length: 1
Keep-Alive: timeout=15,max=100

1
azrael:~ mosser$
```

Consuming from Java

```java
public class BankAPI {

  private String url;

  public BankAPI(String host, String port) {
    this.url = "http://" + host + ":" + port;
  }

  public BankAPI() { this("localhost", "9090"); }

  public boolean performPayment(Customer customer, double value) throws ExternalPartnerException {
    // Building payment request
    JSONObject request = new JSONObject().put("CreditCard", customer.getCreditCard())
                                         .put("Amount", value);
    // Sending a Payment request to the mailbox
    Integer id;
    try {
      String str = WebClient.create(url).path("/mailbox")
                            .accept(MediaType.APPLICATION_JSON_TYPE)
                            .header("Content-Type", MediaType.APPLICATION_JSON)
                            .post(request.toString(), String.class);
      id = Integer.parseInt(str);
    } catch (Exception e) {
      throw new ExternalPartnerException(url+"/mailbox", e);
    }

    // Retrieving the payment status
    JSONObject payment;
    try {
      String response = WebClient.create(url).path("/payments/" + id).get(String.class);
      payment = new JSONObject(response);
    } catch (Exception e) {
      throw new ExternalPartnerException(url + "payments/" + id, e);
    }
    // Assessing the payment status
    return (payment.getInt("Status") == 0);
  }
}
```

The All Together

```
WebClient.create(url).path("/mailbox")
  .accept(MediaType.APPLICATION_JSON_TYPE)
  .header("Content-Type", MediaType.APPLICATION_JSON)
  .post(request.toString(), String.class);
```



transport

marshalling

```
JSONObject request =
  new JSONObject()
    .put("CreditCard",customer.getCreditCard())
    .put("Amount", value);
```

unmarshalling

```
[WebInvoke( Method = "POST", UriTemplate = "mailbox",
            RequestFormat = WebMessageFormat.Json,
            ResponseFormat = WebMessageFormat.Json)]
```