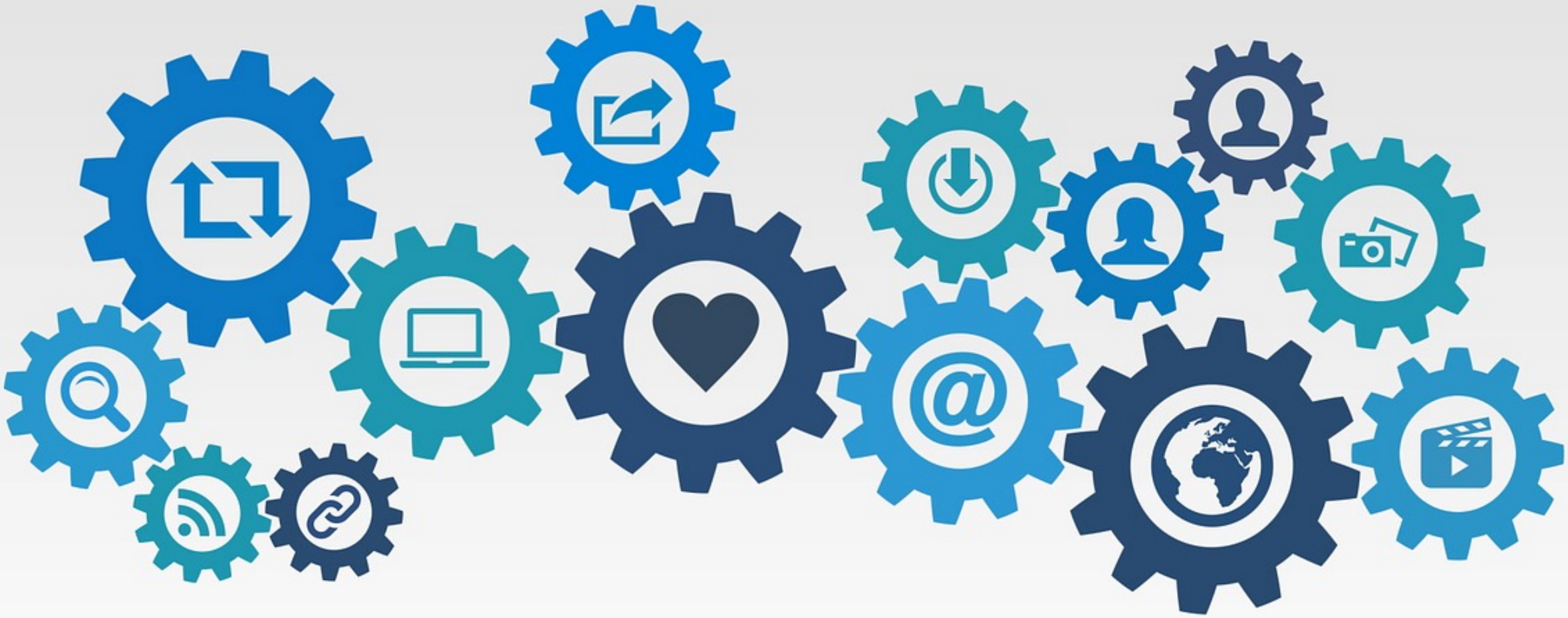
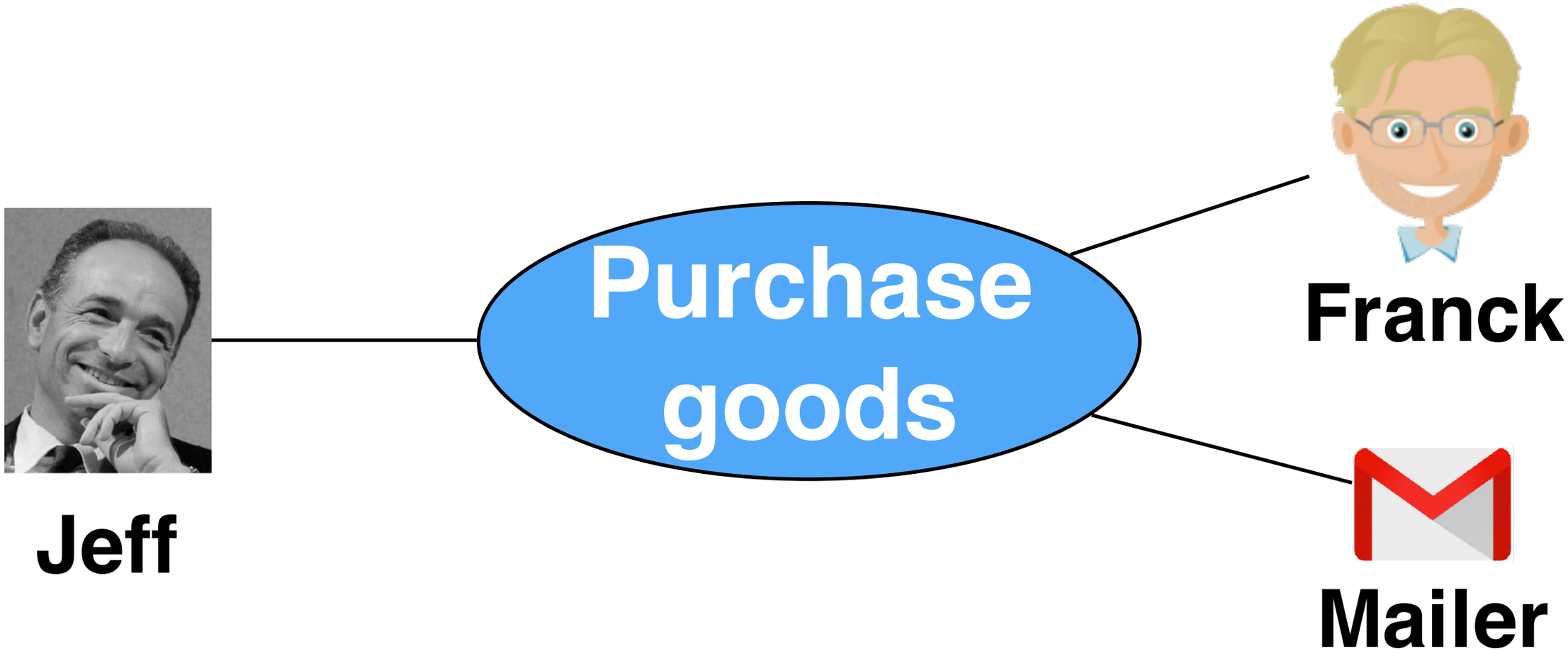


Interoperability with Web Services

AM Dery
Fortement inspirée des cours de S Mosser

Exemple
Loyalty System



As Jeff (Customer),
I want to log a purchase on my card
So that I know my loyalty credit increase

Scenario: Purchase Goods (MVP)

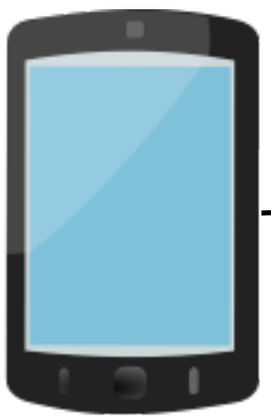
1. Jeff (a Customer) presents a loyalty card and the goods to be purchased;
2. Franck (a Dealer) scans the card, and logs the purchase information;
3. These data are sent to the Loyalty System;
4. The purchase amount is transformed into Loyalty credit points;
5. This amount is added to the balance of the customer (based on the card ID);
6. An email is sent to the user email with the new balance.

LogTransaction:

register(???)

Messaging:

sendMail(data: Message)



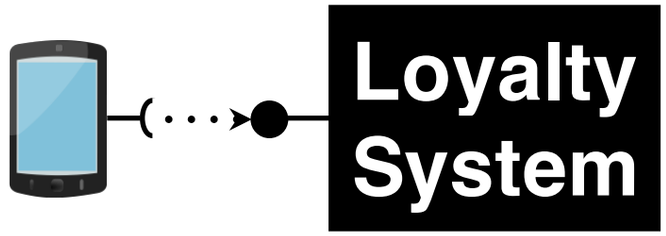
Loyalty System

Mailer

Our System

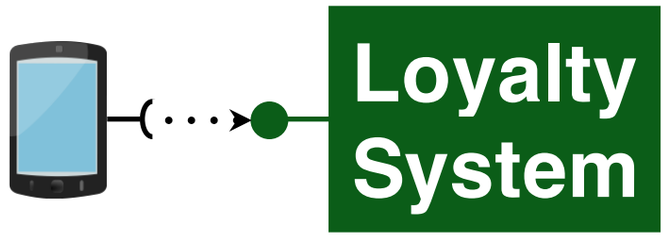
Client

External partner



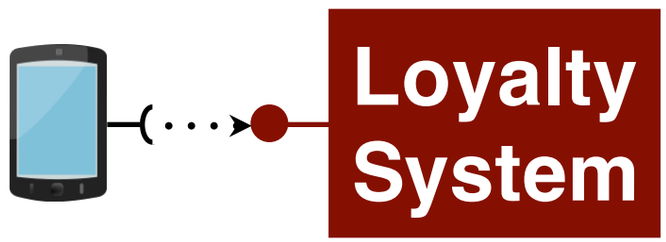
LogTransaction:

register(shop: Shop, card: Image, prod: Product, quantity: Int)



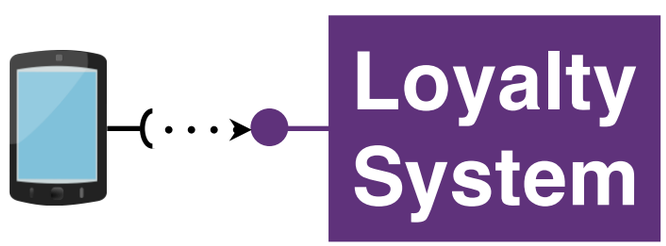
LogTransaction:

register(shop: ID, card:ID, product: Product, value: Float)



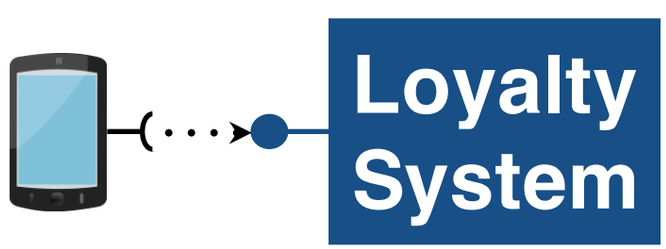
LogTransaction:

register(shop: ID, card:ID, product: ID, value: Float)



LogTransaction:

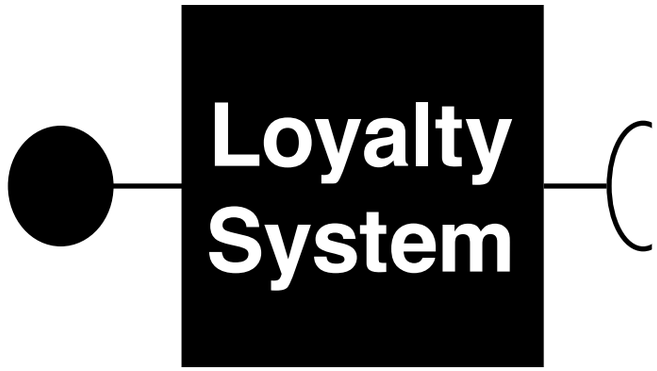
register(shop: ID, card:ID, value: Float)



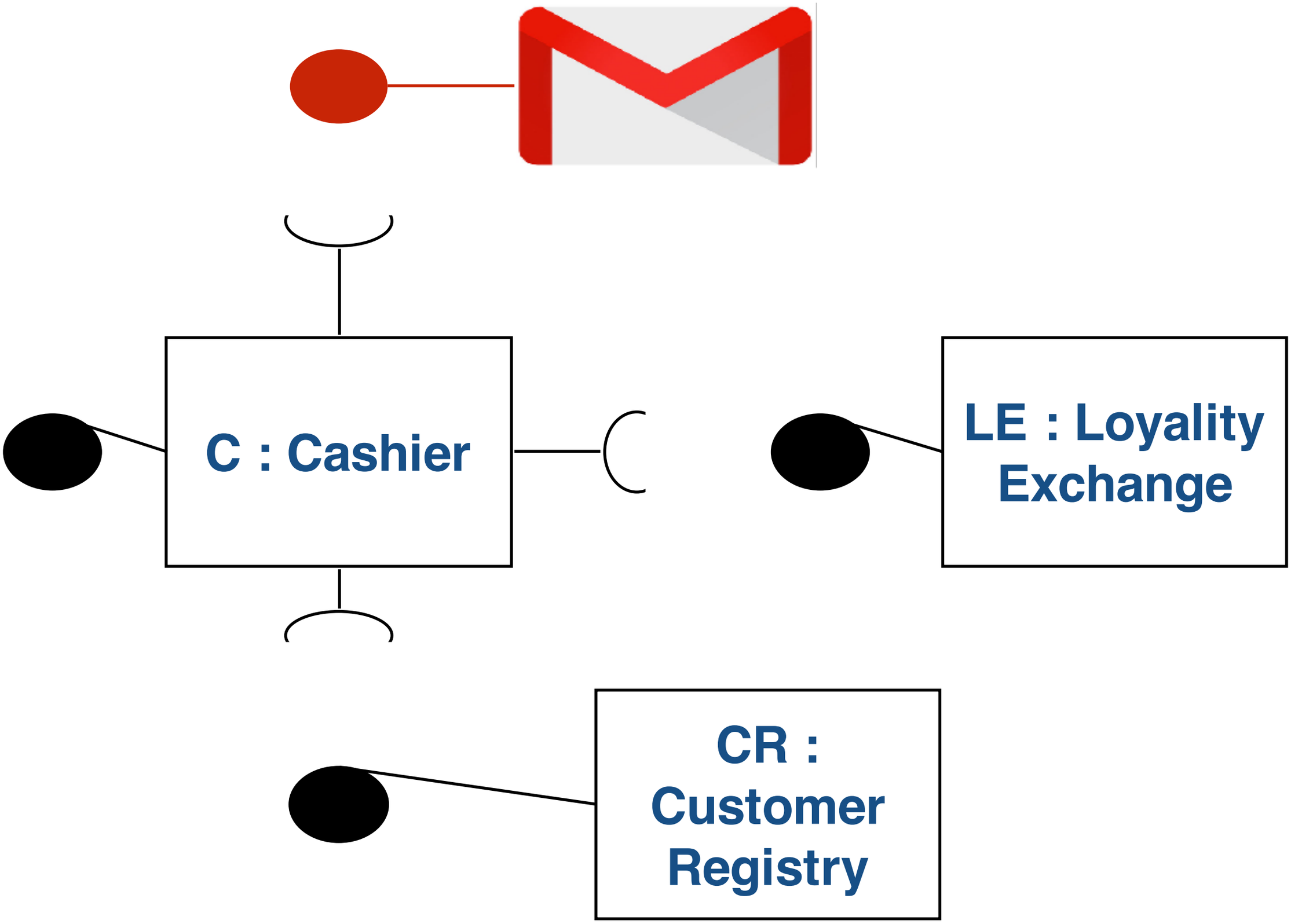
LogTransaction:

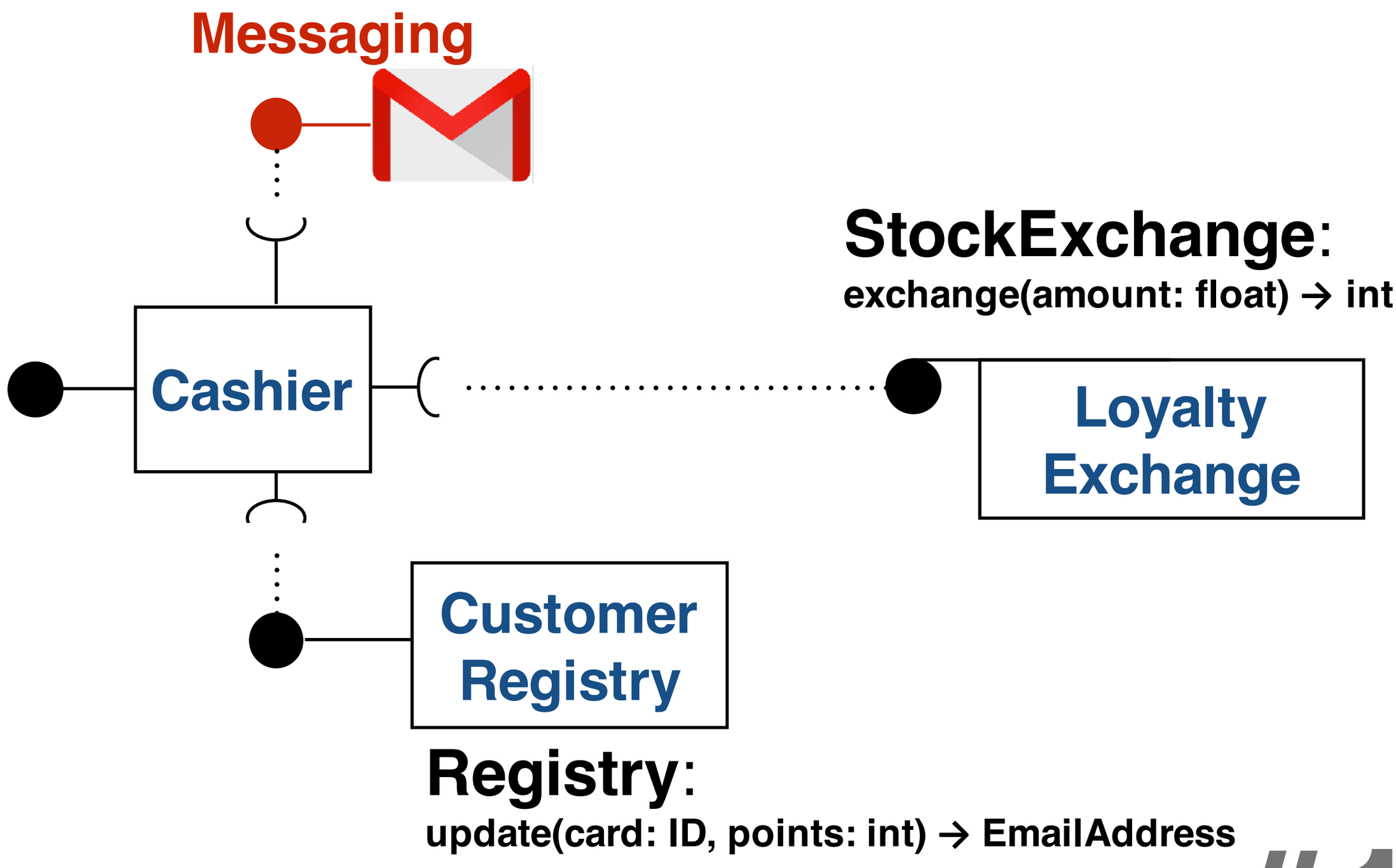
register(transaction: Transaction)

Componentizing the system

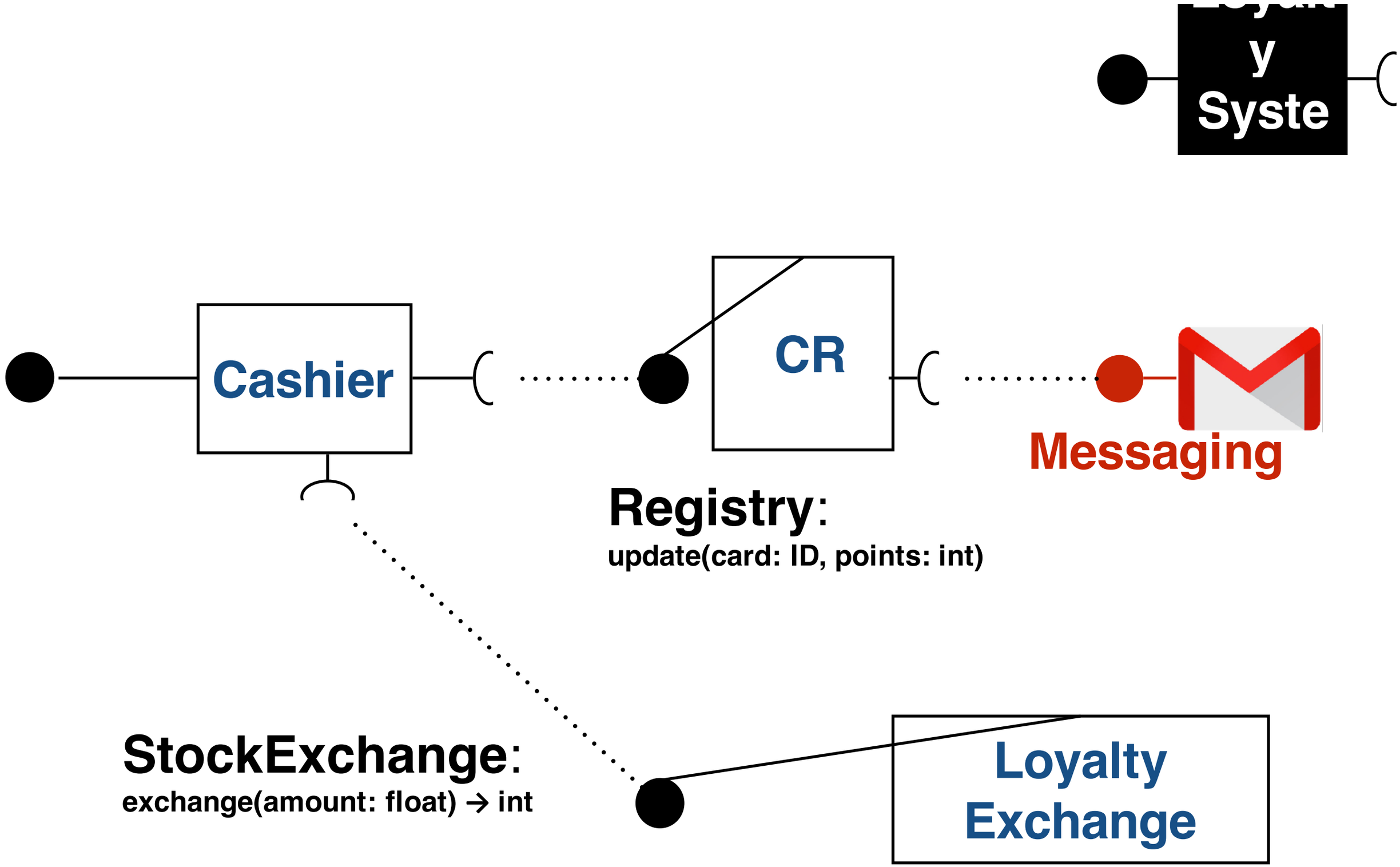


4. The purchase amount is transformed into Loyalty credit points;
5. This amount is added to the balance of the customer (based on the card ID);
6. An email is sent to the user email with the new balance.

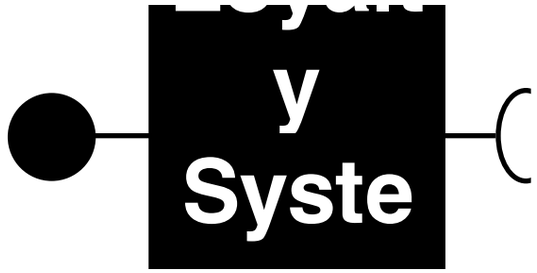

C : Cashier
LE : Loyality Exchange
CR : Customer Registry


Messaging
StockExchange:
exchange(amount: float) → int
Cashier
Loyalty Exchange
Customer Registry
Registry:
update(card: ID, points: int) → EmailAddress

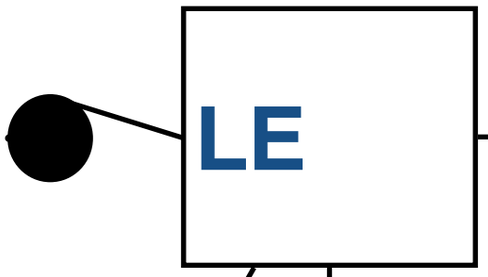
#1


y
Syste
Cashier
CR
Messaging
Registry:
update(card: ID, points: int)
StockExchange:
exchange(amount: float) → int
Loyalty
Exchange

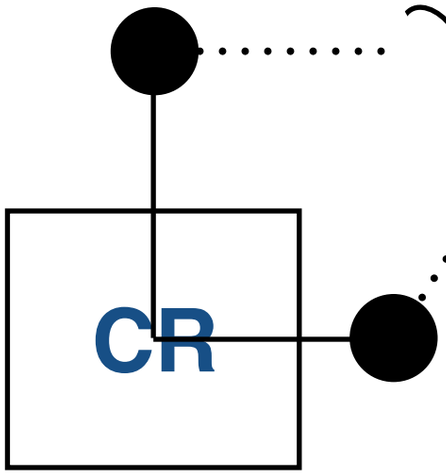
#2



StockExchange:
exchange(amount: float, card: ID)



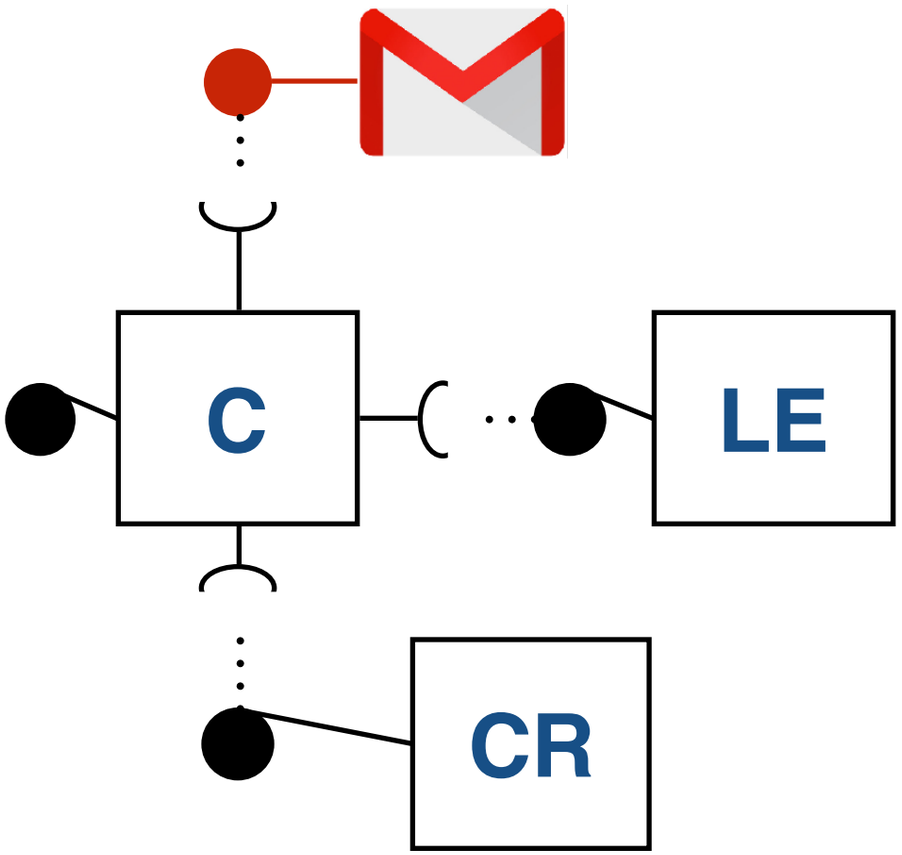
Messaging

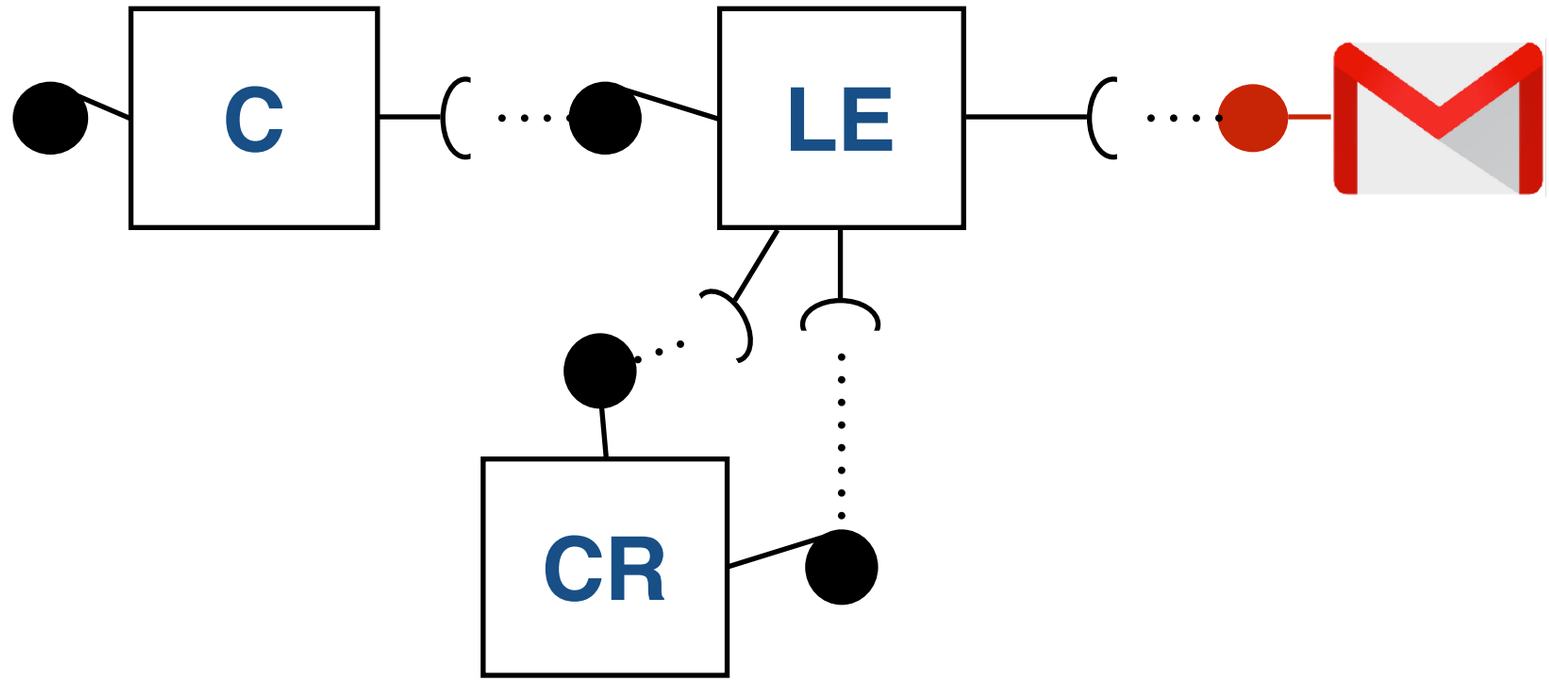


Finder:
get(card: ID) → Customer

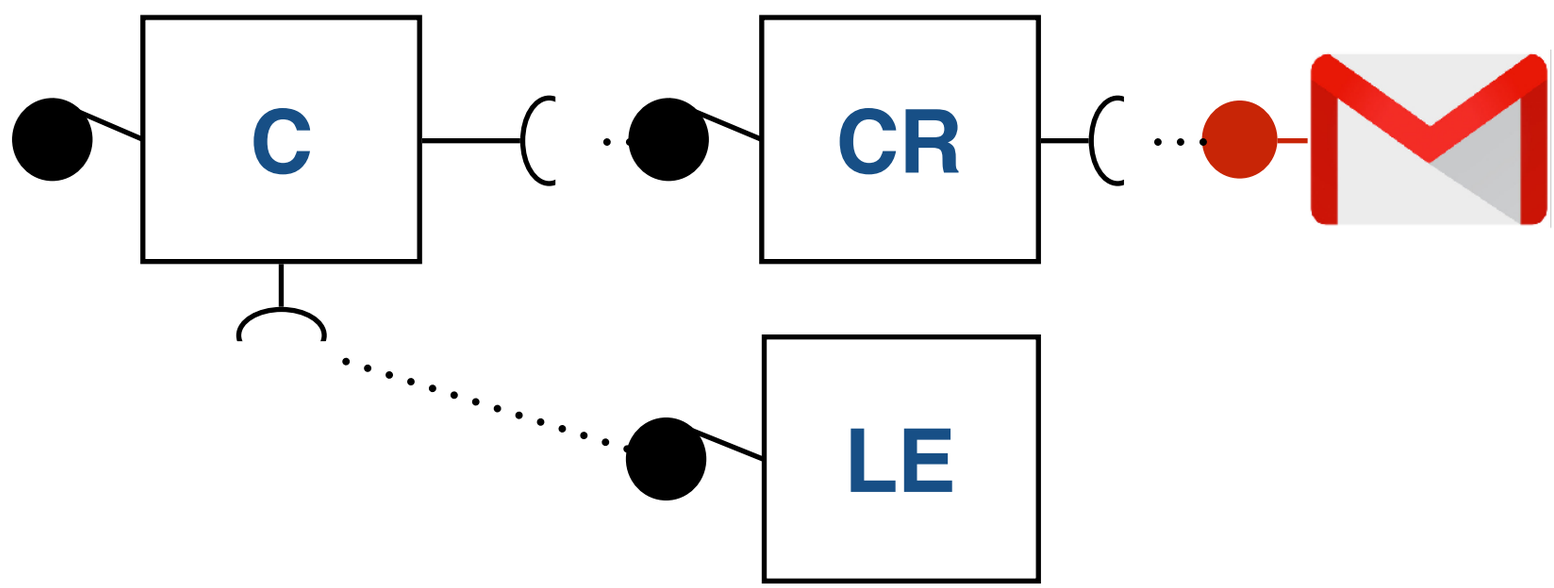
Registry:
update(card: ID, points: int)

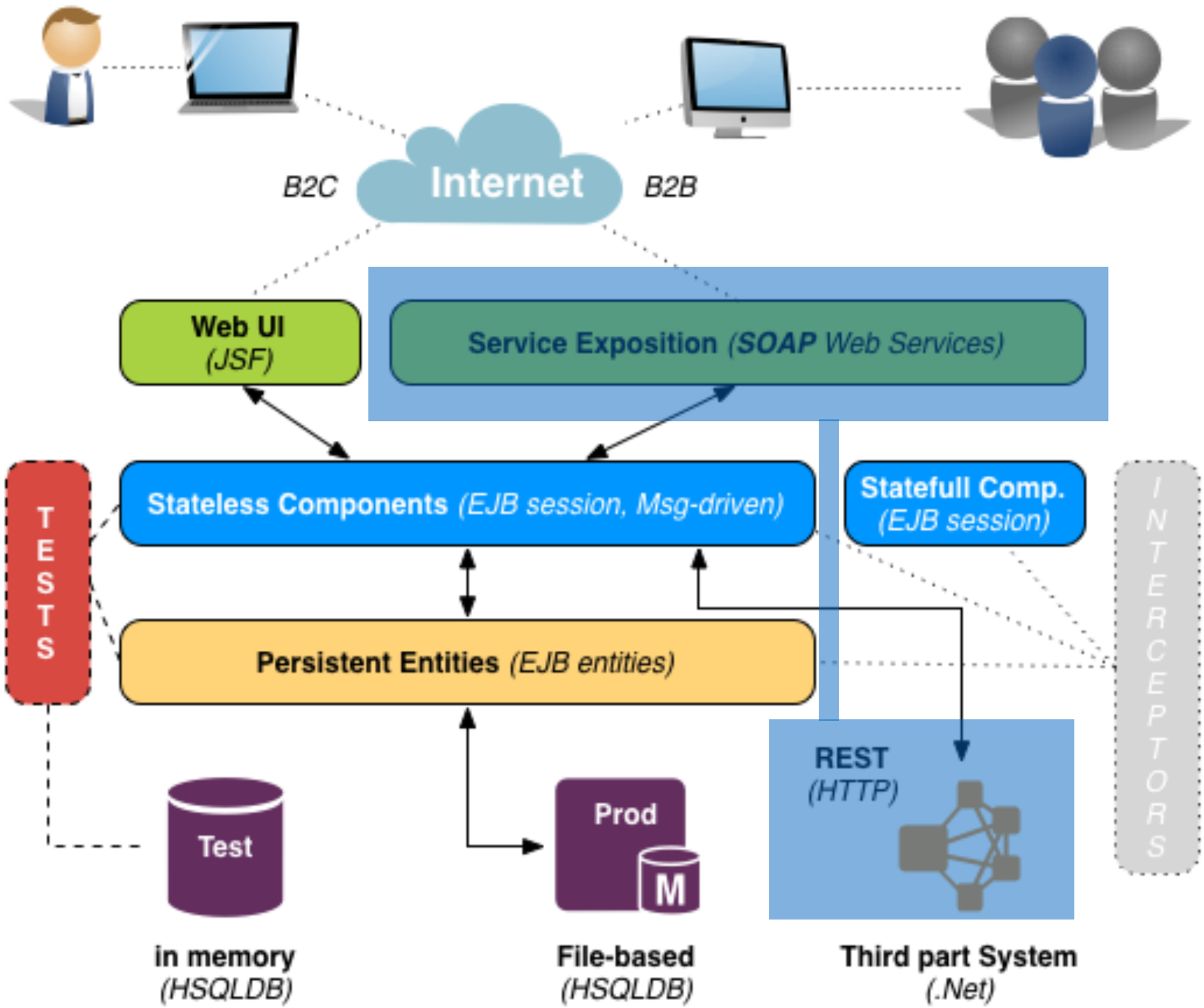
#3


C
LE
CR


C
LE
CR

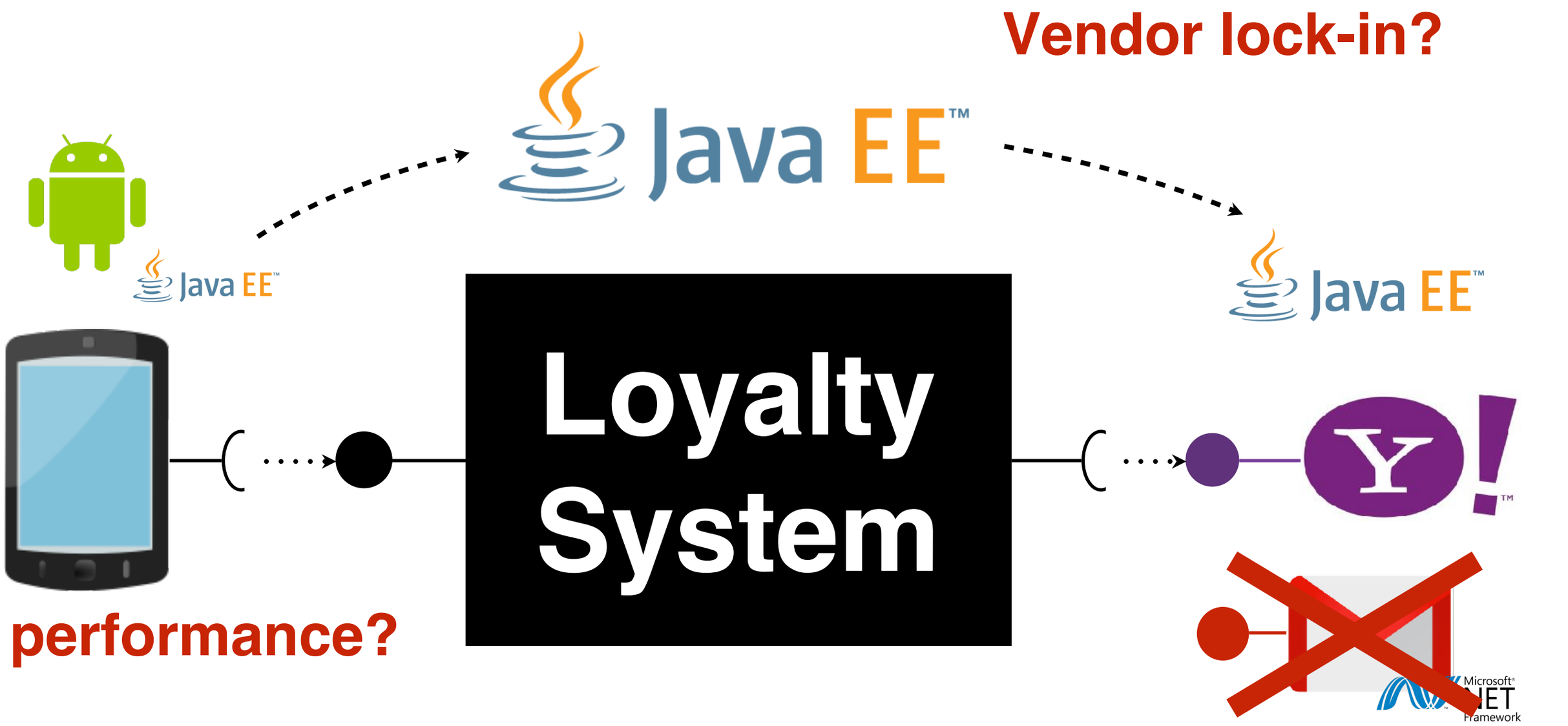

Deal With It


C
CR
LE


B2C
Internet
B2B
Web UI
(JSF)
Service Exposition (SOAP Web Services)
TESTS
Stateless Components (EJB session, Msg-driven)
Statefull Comp.
(EJB session)
INTERCEPTORS
Persistent Entities (EJB entities)
REST
(HTTP)
Test
Prod
M
in memory
(HSQLDB)
File-based
(HSQLDB)
Third part System
(.Net)

Public APIs support flexibility

Using J2E dependency injection



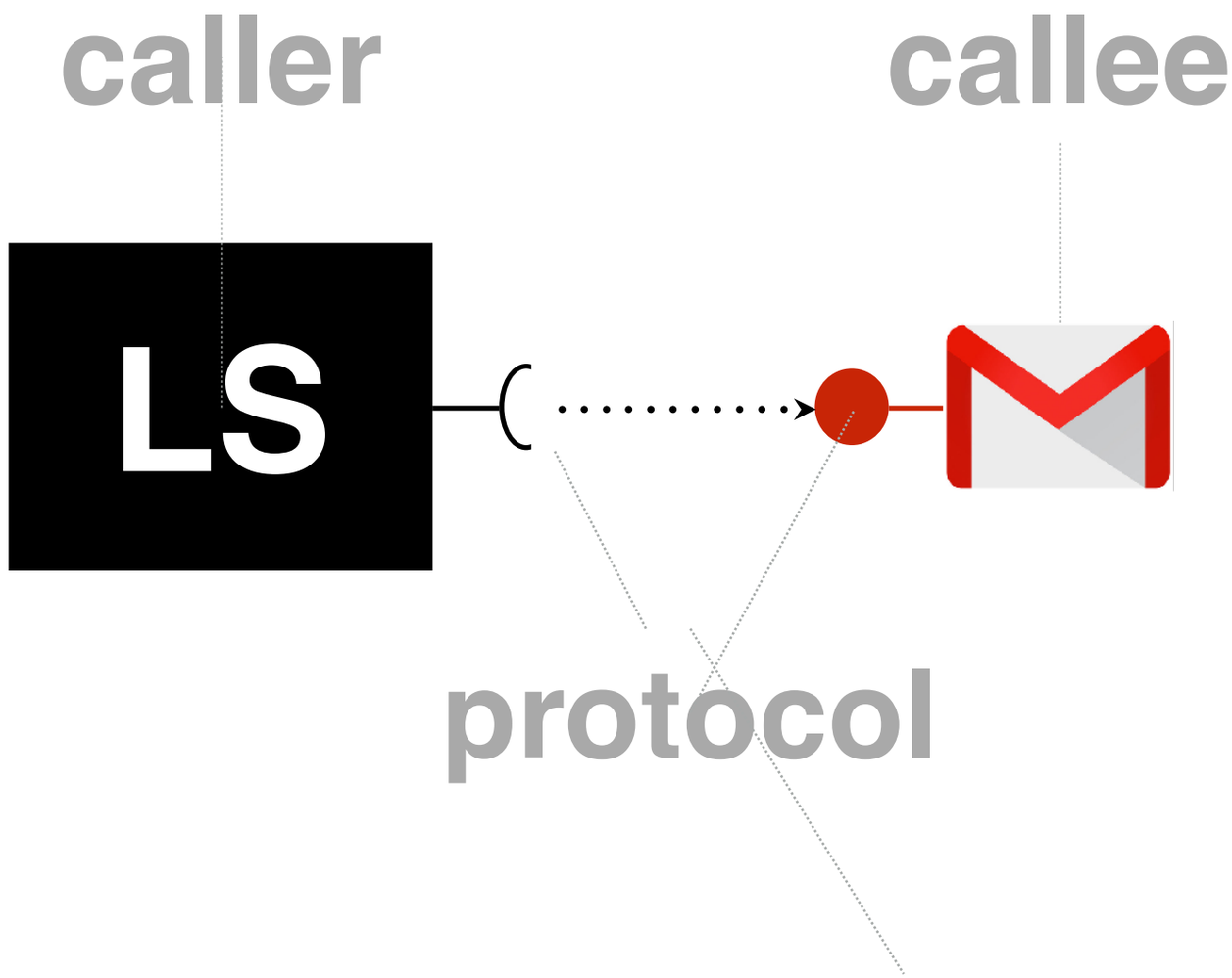
technology driving business?

Homogeneous System

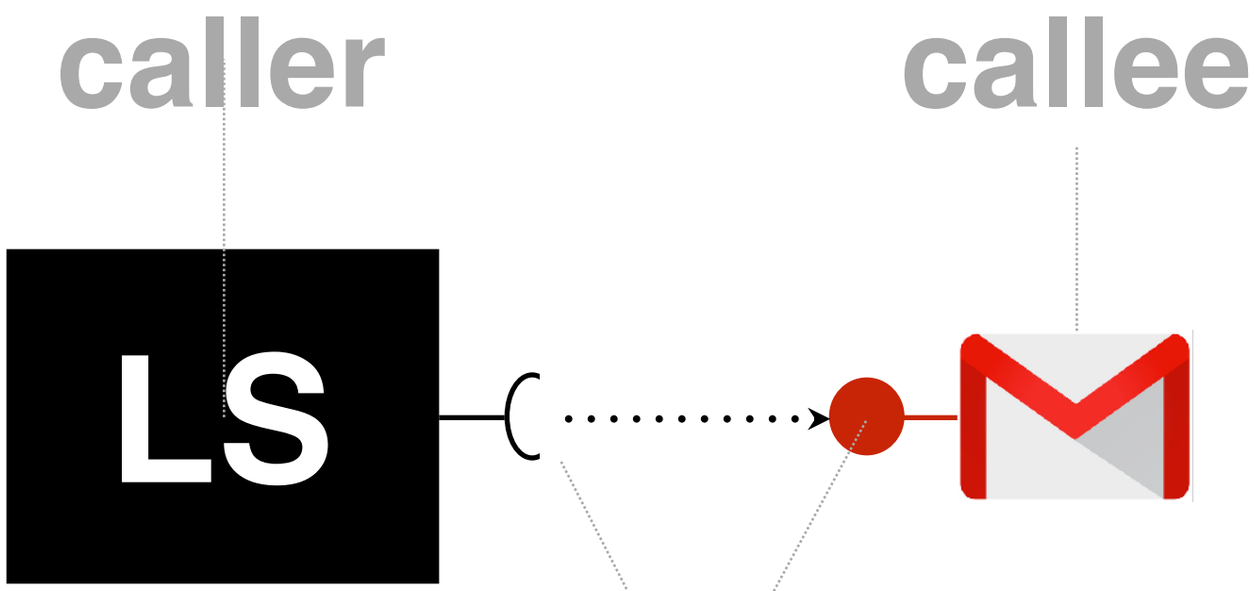
Interoperability ?

Heterogeneous System

Abstracting from Implementation



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



Messaging:

sendMail(data: Message)

Defined in the interface

protocol

- Endpoint: Where, How ?

- Operations: Why ?
- Business Object: What ?

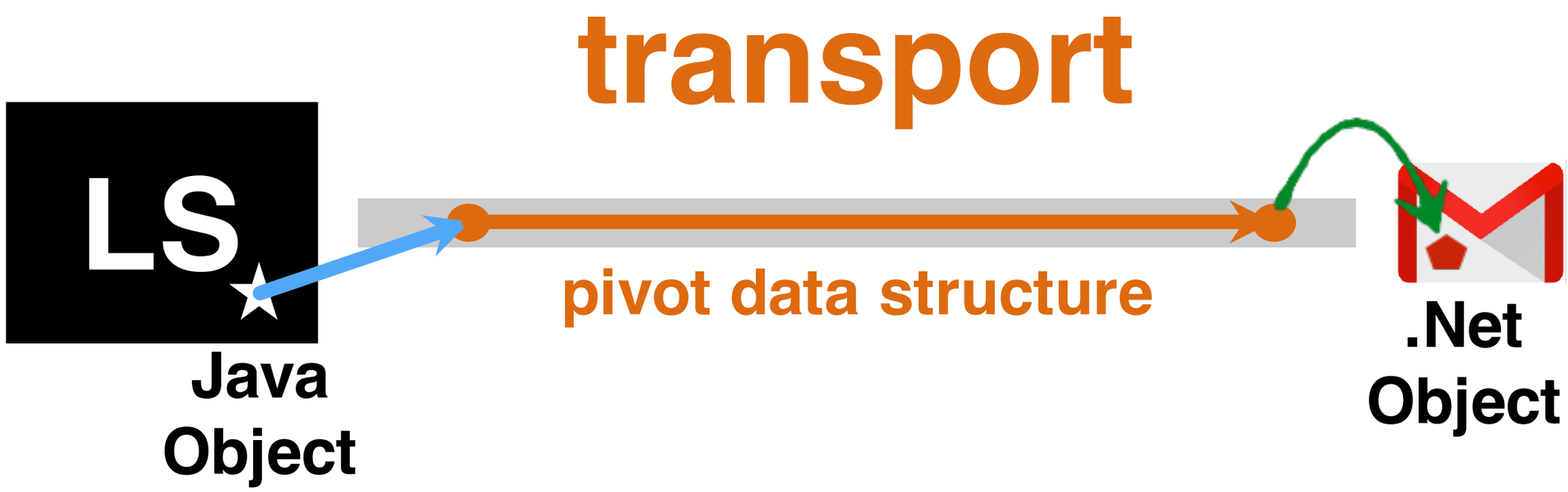
Endpoint

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

Platform Independent

marshalling:
Object → Pivot

unmarshalling:
Pivot → Object



REST vs SOAP

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

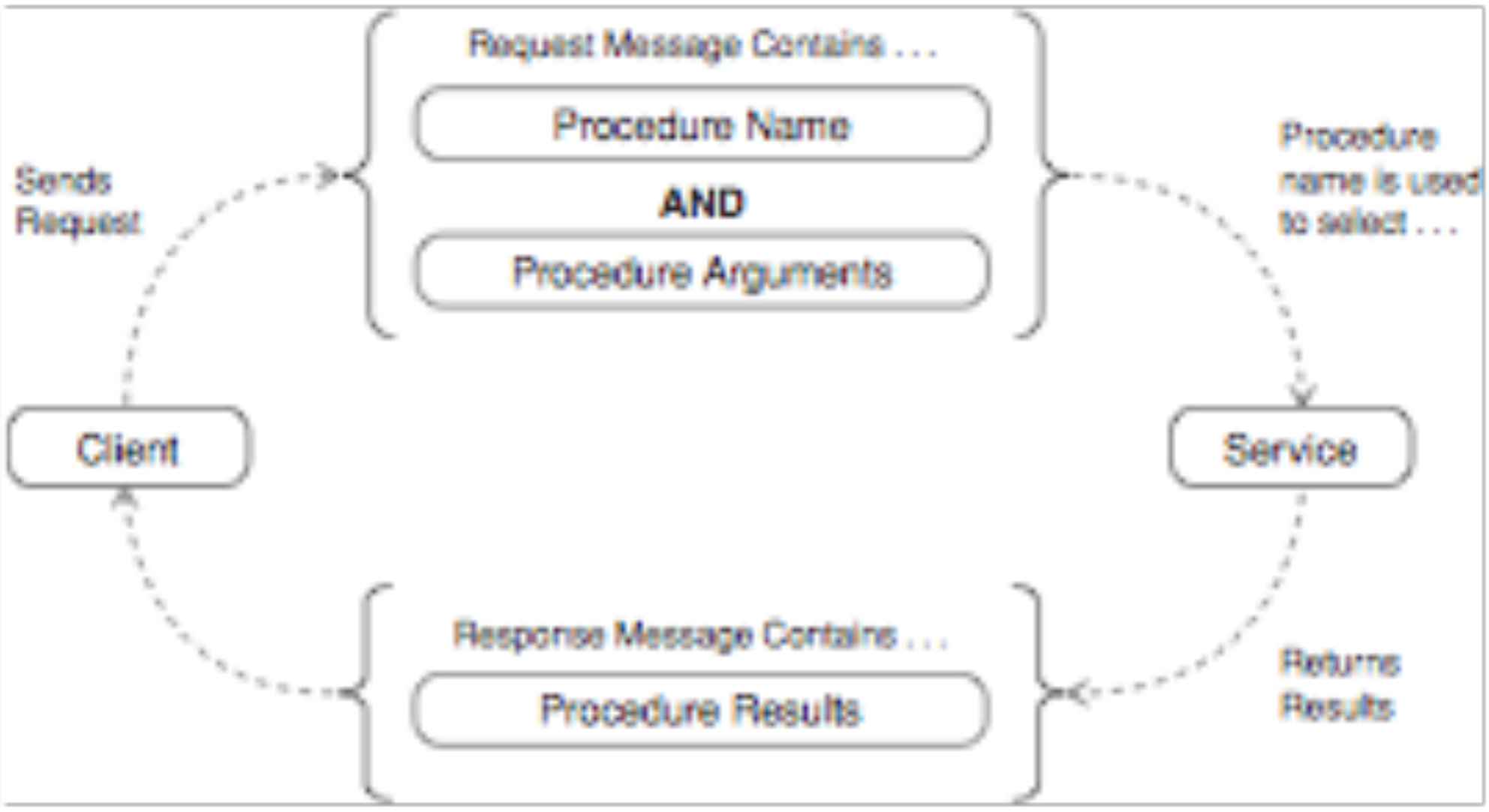
REST ➔
exposes **resources** (*i.e.*, nouns instead of verbs).

Contracts and style

Exposing Resources (Nouns)

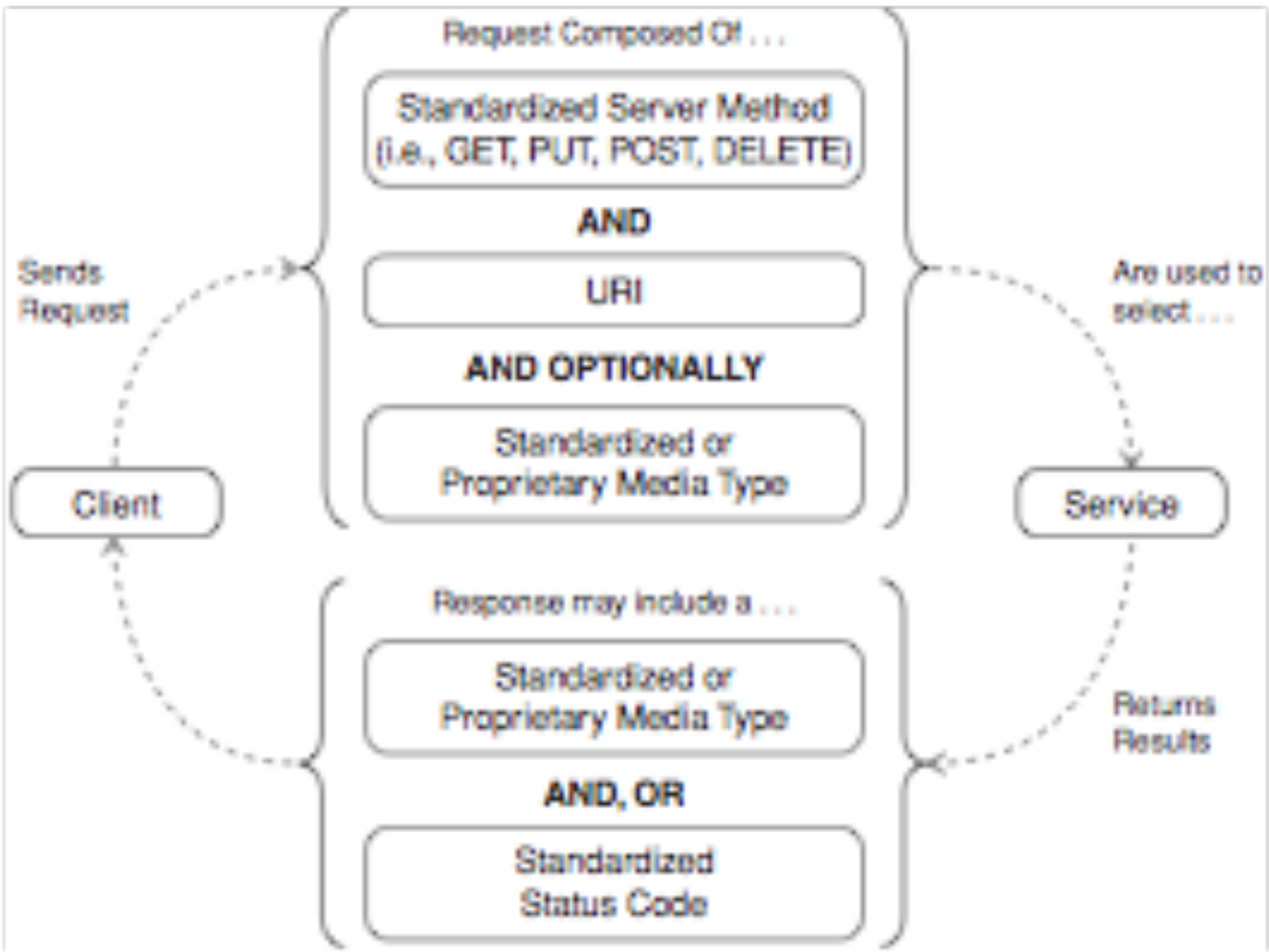
Exposing Operations (Verbs)

RPC Interaction Protocol



[Service Design Patterns]

Resource Interaction Protocol



[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

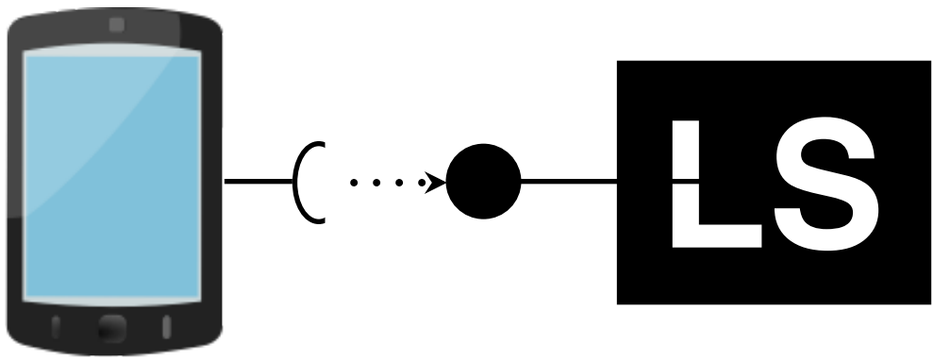
With a Contribution by
Ian Robinson

Forewords by
Martin Fowler *and* Ian Robinson

Public APIs support interoperability

Strong Contract

Expose Web services (SOAP)



The Cookie Factory open source example

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

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

Contracts ?

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

?

Web service contract

defined by an annotated interface.

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

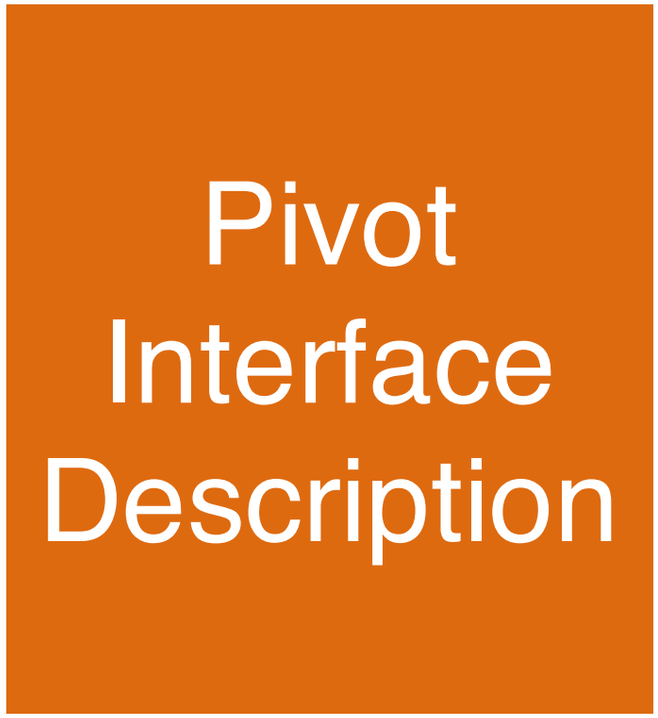
2. A WebParam annotation tags the parameters to change their name, or handle xml namespace manually

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

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

Contract

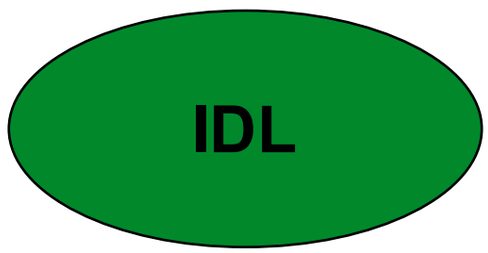
```java
@WebService
public interface CartWebService {

  @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;
}
```

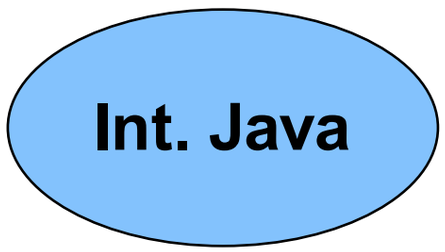


Compilation process
Interface → Pivot

Générateurs

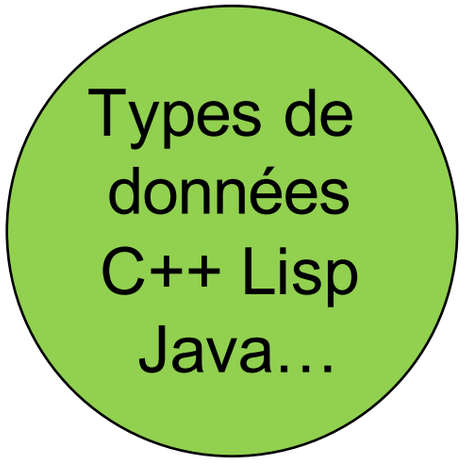
Spécifications des données





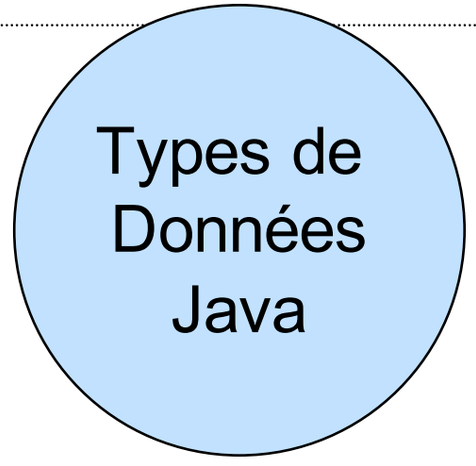
Générateurs

Fichiers générés



Stubs Skeletons Proxy

(mise en œuvre de la sérialisation et désérialisation…)



Points communs avec les middlewares objets

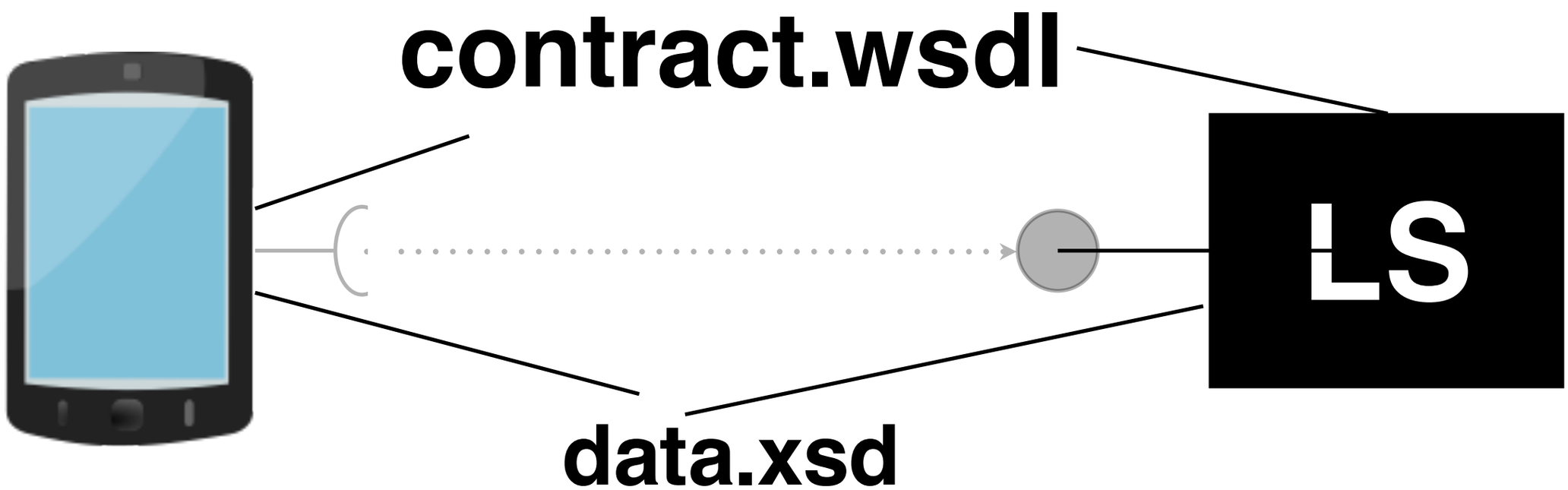
Un langage de description : WSDL

Une infrastructure : Le Web et http

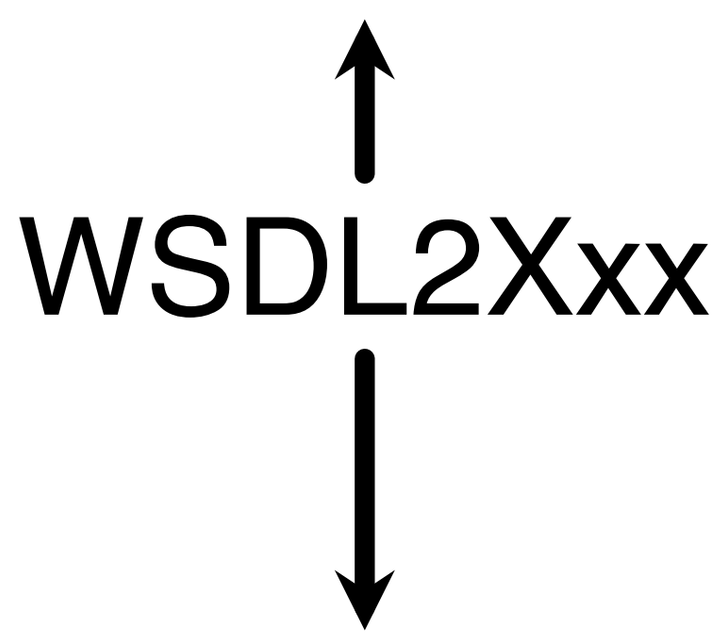
Une communication par envoi de messages : SOAP

Du marshalling : XML

Un service de nommage « dynamique » : UDDI



Data structures using the Xxx language



(Un)Marshalling using the Xxx language

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.

1. Générer le contrat .wsdl
2. Générer le code java à partir du .wsdl
3. Ecrire le code client

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

Automated Generation & Exposition

localhost:8080/tcf-backend/w

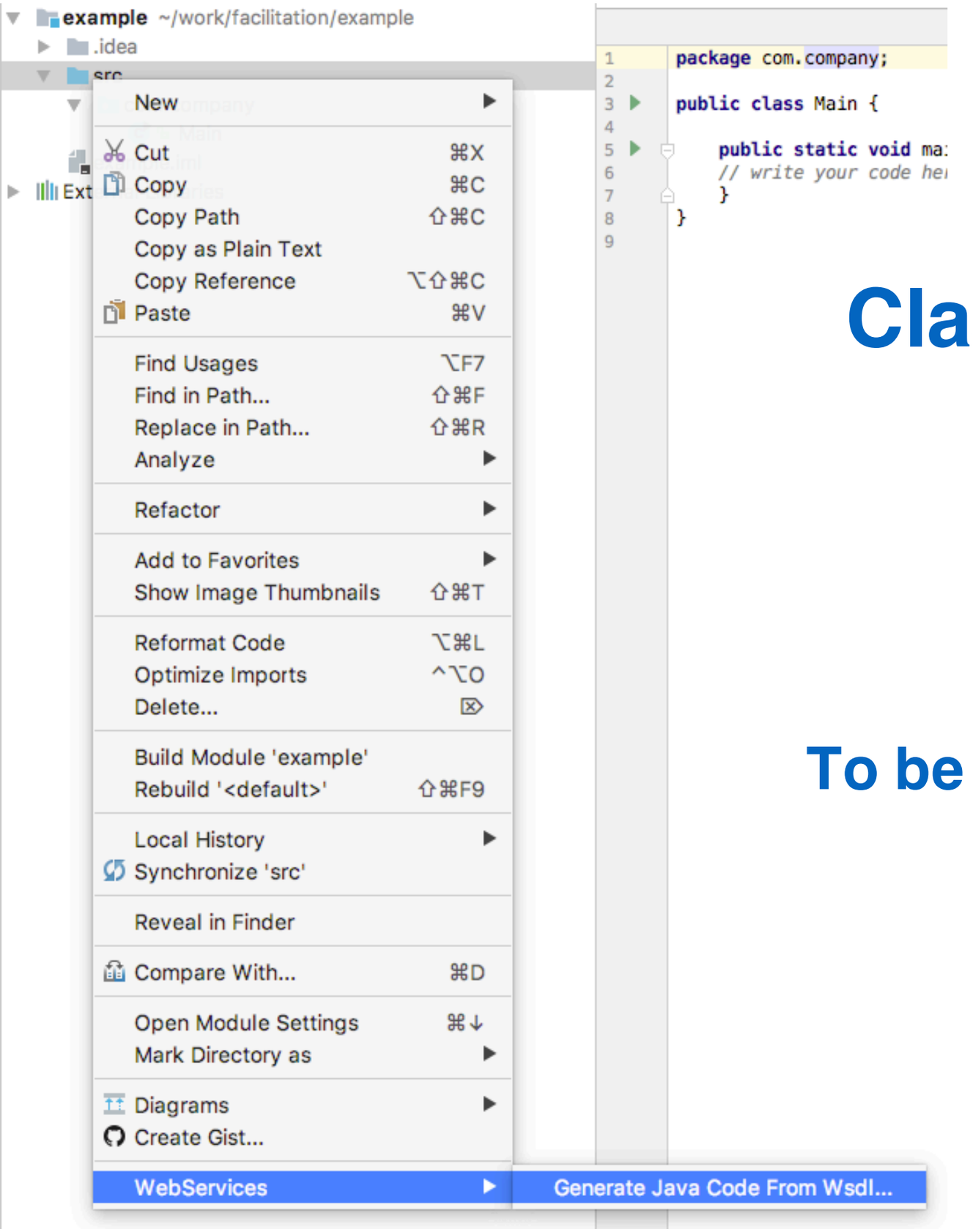
Sébastien

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



Classical (as in Standard) Code generator

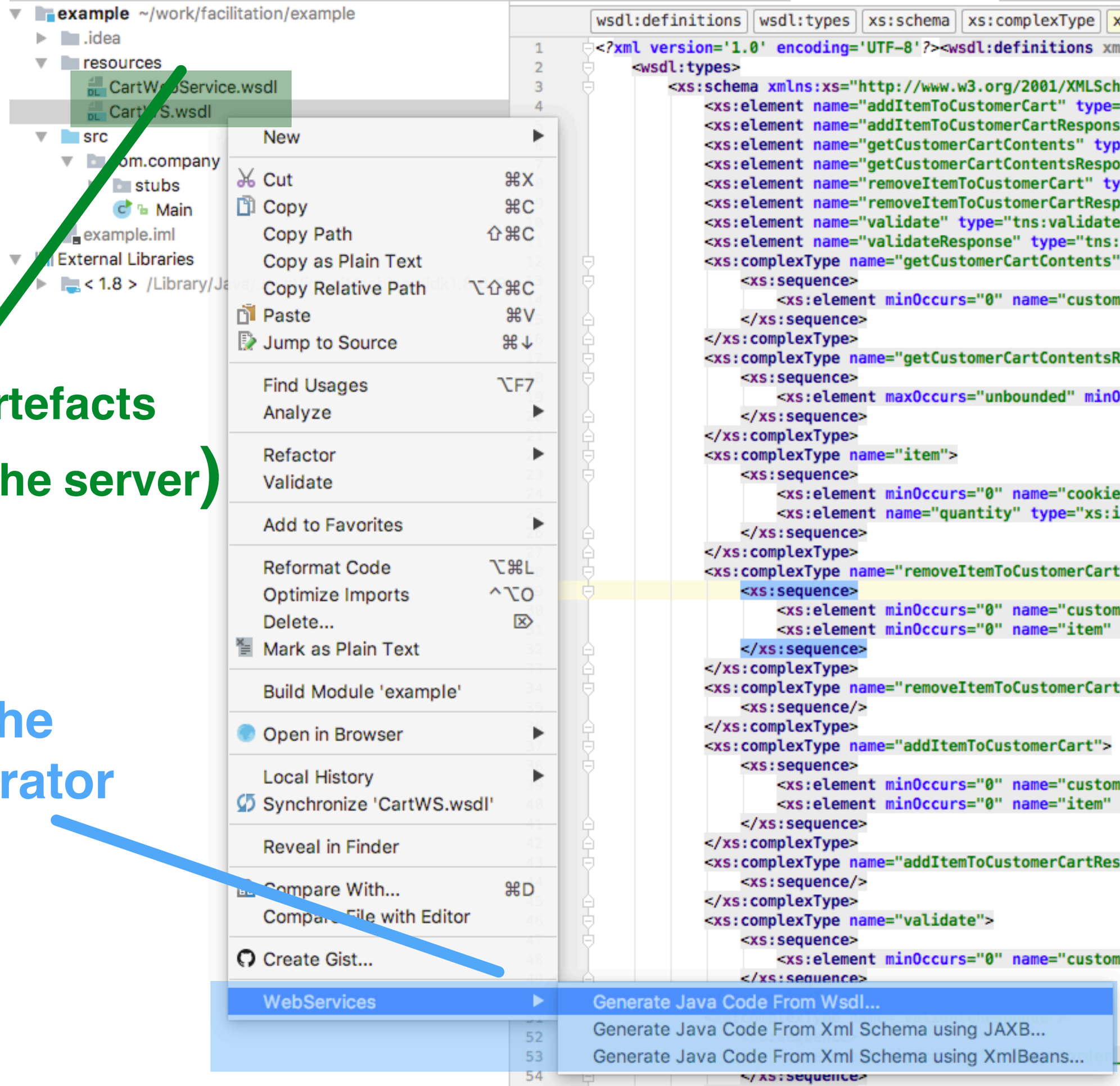
To be called on client side (obviously)

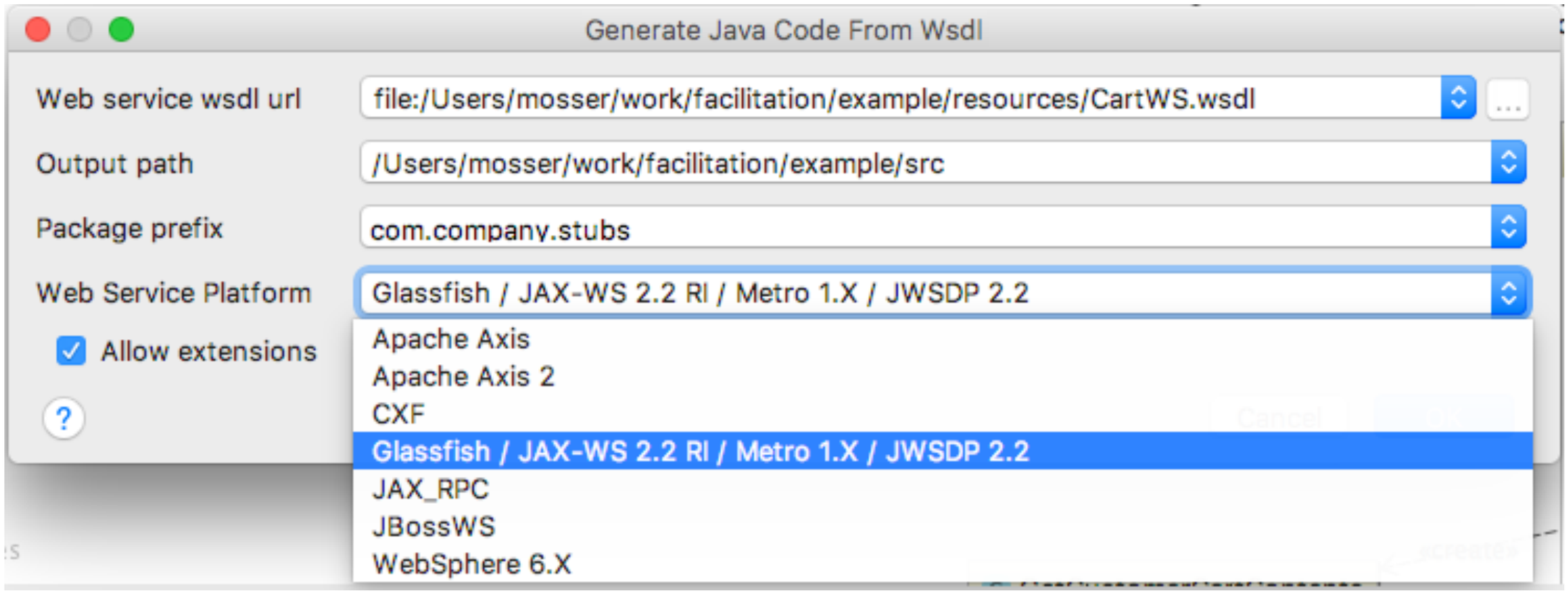
Here IntelliJ Ultimate

Contract artefacts

(shared with the server)

Calling the code generator

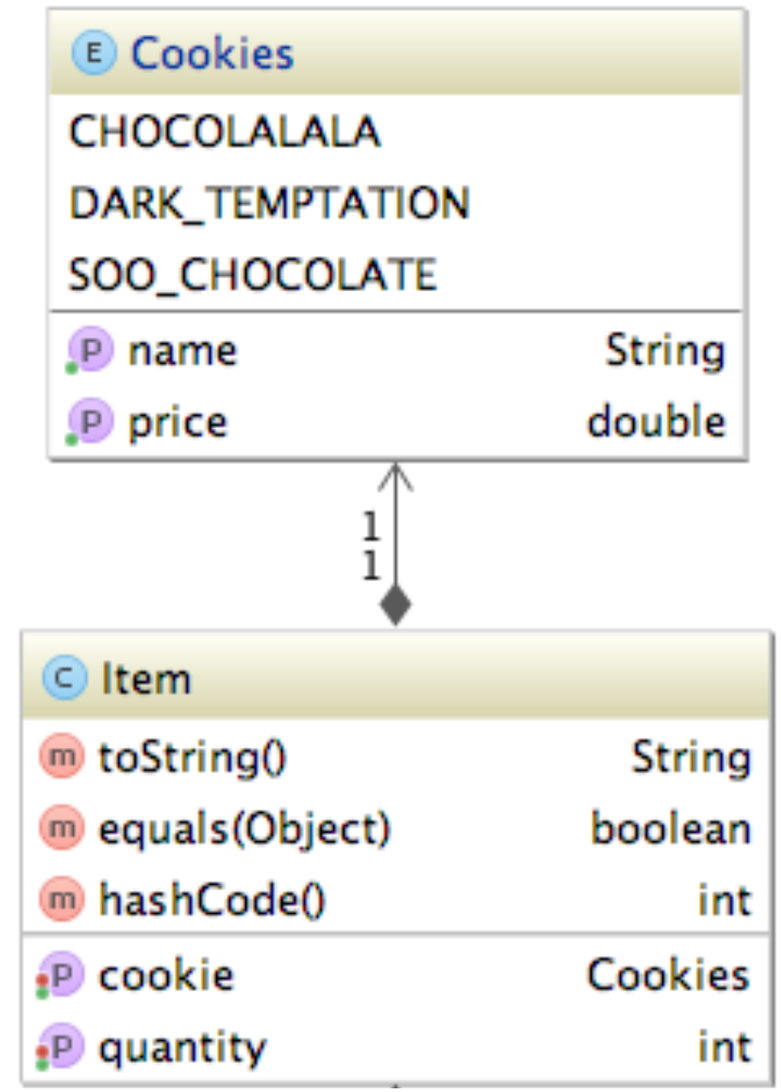




Standard ≠ single implementation

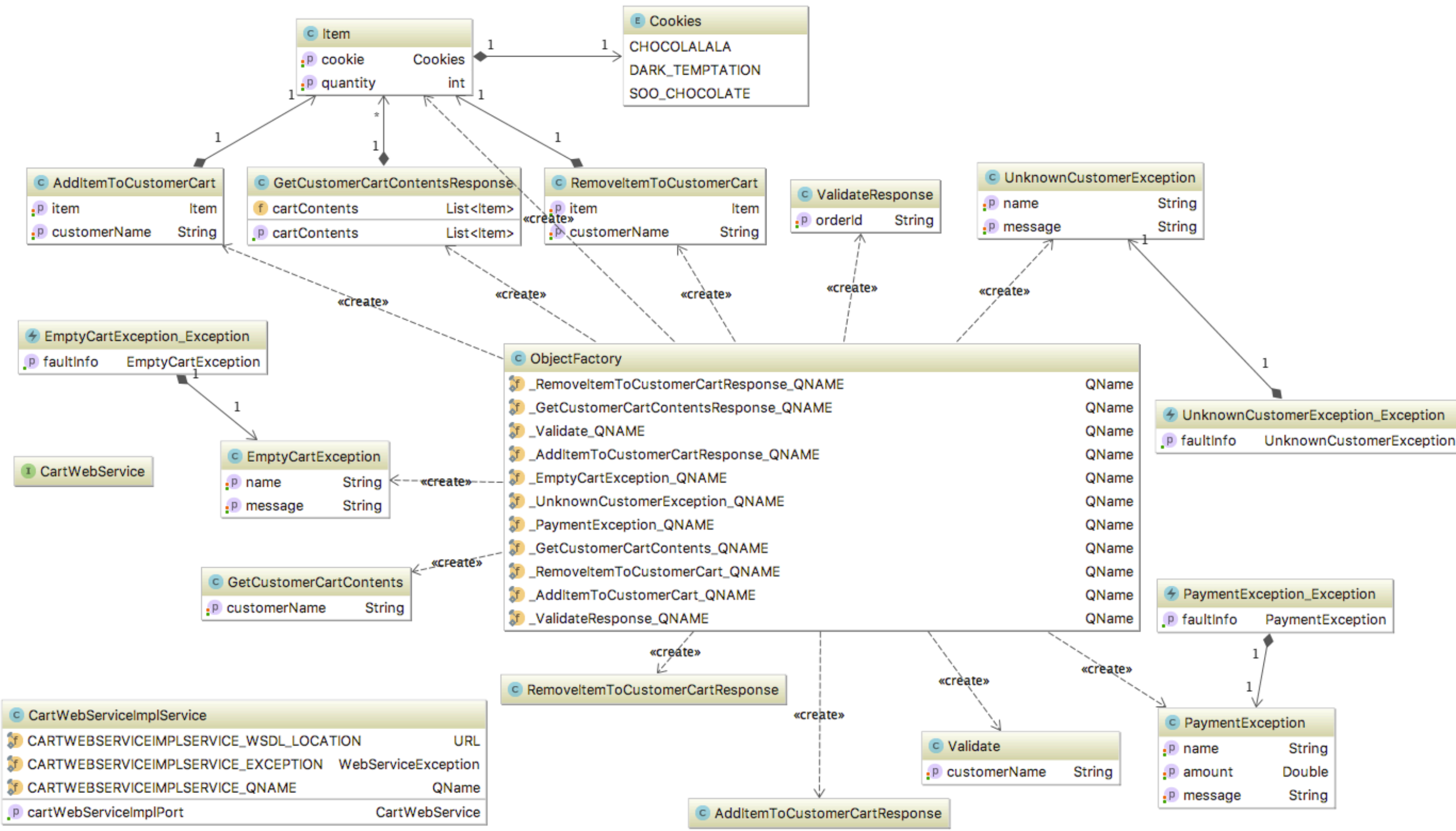
XSD for data structures

```
<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

Generated Code!



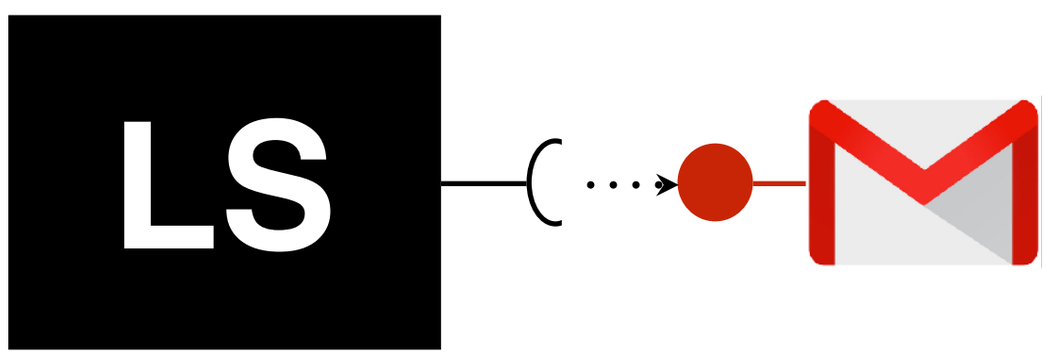
Consuming a service == sending messages to objects

```
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 I No Contract

Consume Web services (REST)



The Cookie Factory open source example

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

Invoking a REST service

The J2E system is client of the service.

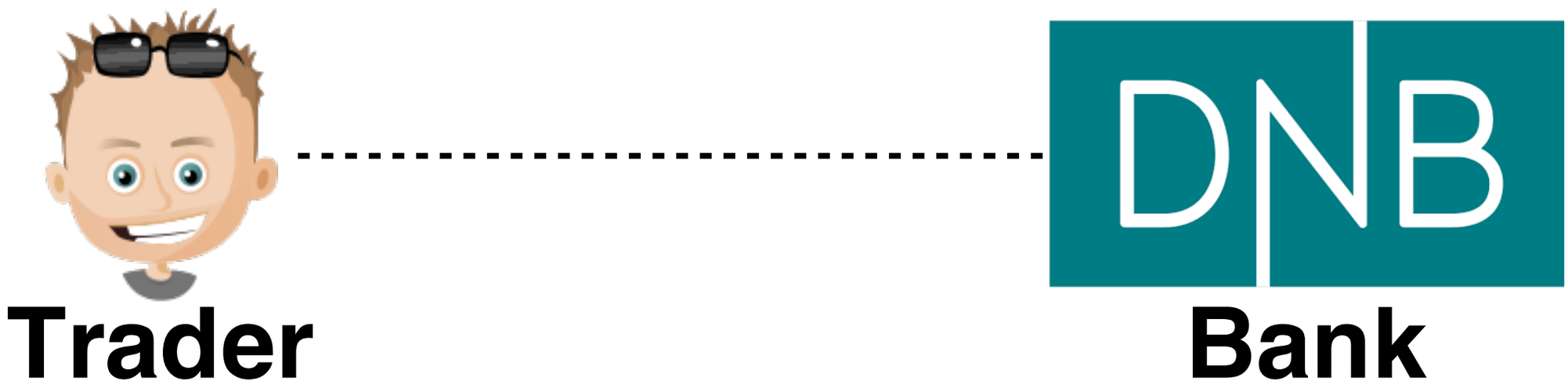
To consume REST web services :

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

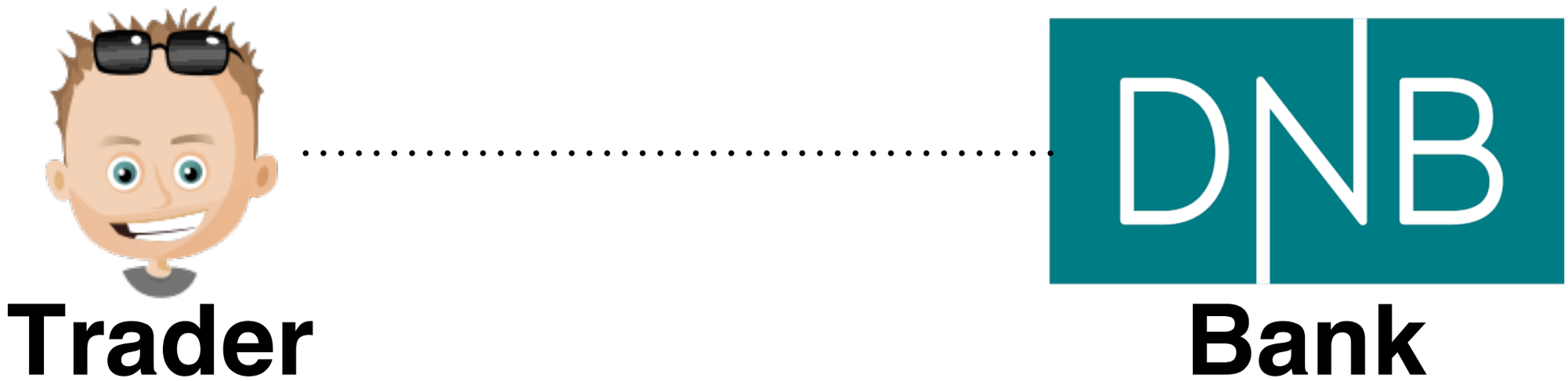
Locate the remote service

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

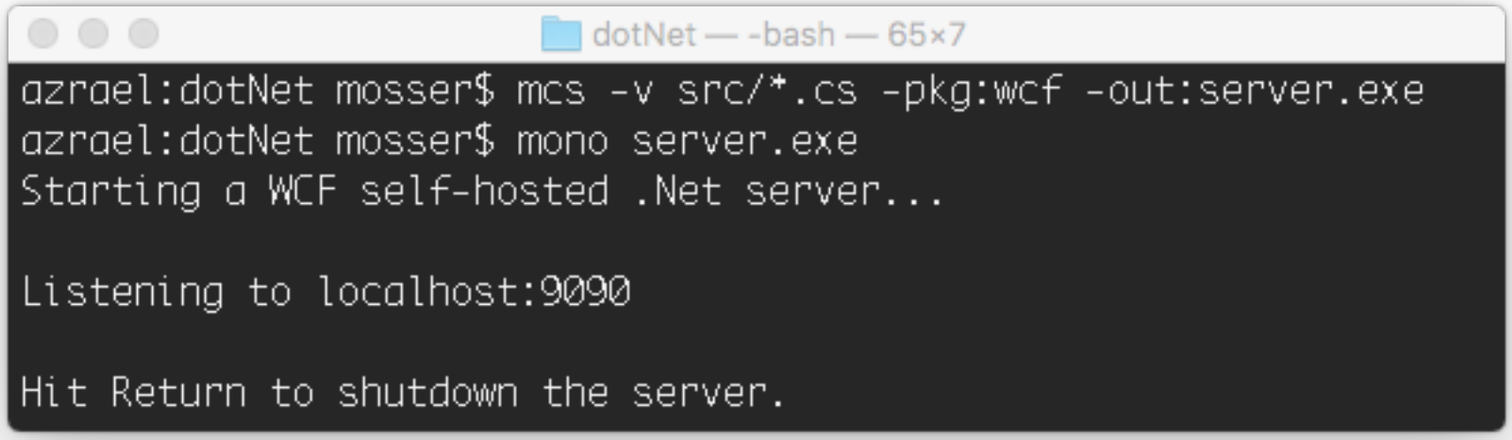

```
[ServiceContract]
public interface IPaymentService
{

  [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




```
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(); }

}
```

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);
```

marshalling

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

unmarshalling

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