Université Lille 1
Master mention Informatique – M1

Construction d'applications réparties

III. Web Services

Lionel.Seinturier@univ-lille1.fr

Introduction

Besoins

Communiquer en environnement réparti & hétérogène (OS, lang)

	Protocole	Format représentation données
• CORBA	IIOP	binaire (CDR)
• EJB	RMI-IIOP	binaire (Java Serialization)
• .NET	.NET Remoting	binaire

⇨ cela fonctionne
⇨ mais solution "propriétaire" (liée à l'environnement)
⇨ interopérabilité inter-environnements ?

Introduction

Solution Web Services

• quel est le protocole le + utilisé sur Internet ?
• quel est le format de données universel ?

• HTTP : simple, sans état, toutes les chances de traverser *firewalls*
• XML : ASCII (pas binaire), *parsing* facile, standard W3C

Différentes "incarnations"

- XML-RPC
- SOAP

- REST
  - basé sur HTTP
  - s'affranchit de la "contrainte" XML (HTML, XML, JSON, JPG, PDF, …)

Plan

1. HTTP – XML
2. REST
3. SOAP
4. WSDL

Introduction

Eléments de base : HTTP

- Protocole applicatif (niveau 7)
- Utilise TCP (niveau 4) ⇨ garantie d'un transport **fiable** (sans erreur)
- **Pas** de notion de **connexion** HTTP
- Tous les commandes HTTP sont émises en **mode texte** (ASCII)

⇨ Protocole simple, facilement implantable

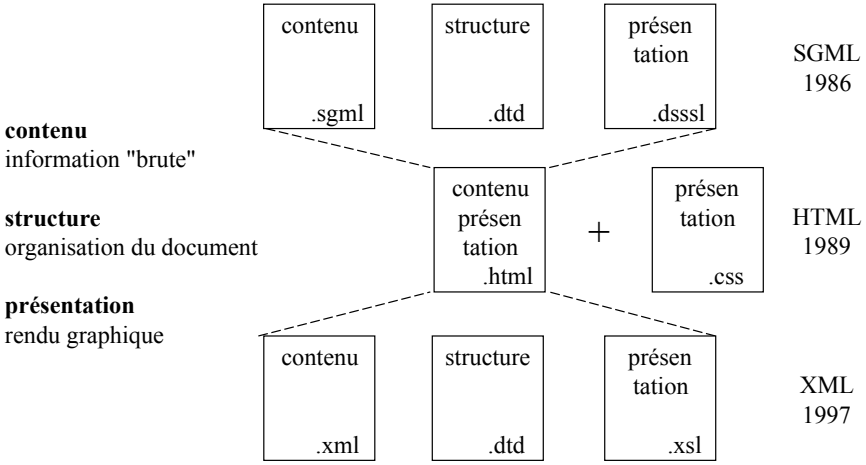
Version actuelle HTTP 1.1 (RFC 2067) depuis janvier 1997
Apport principal : **connexions TCP persistantes**

Raison : pour les "petits" fichiers (< 10 Ko, 80 % des documents Web)
le coût de l'ouverture de cx TCP est **non négligeable** / coût du transfert

⇨ gain de temps important

Introduction

Eléments de base : XML



Introduction

Eléments de base : XML

DTD grammaire (balises) du document

1. Définition des balises autorisées `<!ELEMENT ... >`
2. Définition de leurs attributs `<!ATTLIST ... >`

```
<balise attribut="type attr" ...> type balise </balise>

<!ELEMENT graphe (noeud|arc)* >
<!ELEMENT noeud EMPTY >
<!ELEMENT arc EMPTY >
<!ATTLIST noeud numero ID #REQUIRED >
<!ATTLIST arc source IDREF #REQUIRED >
<!ATTLIST arc destin IDREF #REQUIRED >

<graphe>
<noeud numero="1"/> <noeud numero="2"/>
<arc source="2" destin="1"></arc>
</graphe>
```


Introduction

Eléments de base : XML

XML Schema

```
<!ELEMENT promotion (individu)+ >
<!ELEMENT individu ( nom , prenom ) >
<!ELEMENT nom (#PCDATA)> <!ELEMENT prenom (#PCDATA)>
<!ATTLIST individu noSecuriteSociale ID #REQUIRED >
```

promotion.dtd

```
<?xml version="1.0" ?>
<element name="promotion" type="PromotionType" />
<complexType name="PromotionType">
  <element name="individu" type="IndividuType"
           minOccurs="1" maxOccurs="unbounded" />
  <attribute name="noSecuriteSociale"
              type="ID" use="required" />
</complexType>
<complexType name="IndividuType">
<sequence> <element name="nom" type="string">
           <element name="prenom" type="string">
</sequence> </complexType>
```

XML *schema* équivalent

Eléments de base : XML

XML Namespace

Utilisation des balises provenant de **+sieurs DTD** dans un doc. XML

- attribut **réservé** `xmlns` fournissant un nom et l'URL de sa DTD associée
- peut être ajouté à n'importe quelle balise (en général, la 1ère du document)

```
<balise xmlns:nomDEspace="URL associée" ... >
```

```
<html xmlns:m="http://www.w3.org/1998/Math/MathML"
      xmlns:s="http://www.w3.org/2000/svg" >
```

- l'espace de noms reste valide jusqu'à la **balise fermante** (ici `</html>`)
- les balises des ≠ DTD doivent être préxifées par *nomDEspace:*

```
<s:svg width="2cm" height="0.6cm">
```

REST

Lionel Seinturier

Université Lille 1

Lionel.Seinturier@univ-lille1.fr

REST

Representational State Transfer

- une "incarnation" des Web Services
- s'affranchit de la contrainte XML
- exploite les différentes commandes du protocole HTTP

- ce n'est pas un protocole
- pas de format de données prédéfinis

2 idées directrices

- basé sur l'idée que tout est ressource
  - accessible via une URL
- utilisation des commandes HTTP
  - pour agir sur ces ressources

➢ REST = style architectural pour les Web Services

REST

Representational State Transfer

Origine

- 2000
- Roy Fielding (un des concepteurs de HTTP 1.0 et 1.1)

Vocabulaire RESTful = application qui se conforme à REST

Principes des services REST (découlent de ceux de HTTP)

- client/serveur
  - éventuellement en "couches" (+sieurs c/s enchaînés)
- sans état
- dont les réponses peuvent être déclarées cacheables ou non
- interface d'accès uniforme quel que soit le serveur

REST

Ressources

Associées à une URI / URL

- http://example.com/resources
- http://example.com/users/bob
- …

- pas de règle
- toute URI est potentiellement une ressource REST
- entité logicielle sur laquelle on va vouloir agir

Pas de format de données imposé pour les échanges d'informations avec les ressources

- XML, JSON, GIF, JPEG, PDF, HTML, …
- identifé par un type MIME (ex. text/xml, image/jpeg, …)

Accès au ressources

Pour chaque ressource

• définition de l'action déclenchée sur la ressource par chaque commande HTTP

• pas de solution générale
• cas par cas

• souvent principe CRUD (Create, Read, Update, Delete)
• mais pas nécessairement

Accès au ressources

Resource	GET	PUT	POST	DELETE
Collection URI such as `http://example.com/resources/`	**List** the members of the collection complete with their member URIs for further navigation. For example list all the cars for sale.	Meaning defined as 'replace the entire collection with another collection'.	**Create** a new entry in the collection where the ID is assigned automatically by the collection. The ID created is usually included as part of the data returned by this operation.	Meaning defined as 'delete the entire collection'.
Member URI such as `http://example.com/resources/7HOU57Y`	**Retrieve** a representation the addressed member of the collection expressed in an appropriate MIME type	**Update** the addressed member of the collection or **create** it with the specified ID.	It would imply treating the addressed member as a collection in its own right and creating a new subordinate of it.	**Delete** the addressed member of the collection.

Frameworks de programmation

• Java : Restlet, Jboss RESTEasy, Jersey, Apache CXF, JAX-RS
• Python : RIP
• Ruby On Rails : Rails
• PHP : Symfony
• Perl : Catalyst REST
• …

Modèle de programmation Java

JAX-RS (JSR 311) projet Jersey

• définit un ensemble d'annotations Java 5 pour REST
• package `javax.ws.rs`

Un exemple de ressource

```
@Path("/helloworld")                // chemin d'accès à la ressource
public class HelloWorldResource {

  @GET                              // accès par commande HTTP GET
  @Produces("text/html")            // format de donnée produit
  public String sayHello() {
    return "<h1>Hello World</h1>";
} }
```

Accès client (NetBeans 6.7.1, projet HelloWorldREST)

http://localhost:8080/HelloWorldREST/resources/**helloworld**

Rq : @Produces peut être omis (charge au client d'interpréter le résultat)

Un deuxième exemple

```
@Path("/library")
public class LibraryResource {
  @GET
  @Path("/books")
  public String getBooks() { ... }

  @GET
  @Path("/book/{isbn}")
  public String getBook( @PathParam("isbn") String isbn ) { ... }

  @POST
  @Path("/book/{isbn}")
  public void addBook( @PathParam("isbn") String isbn,
                       @QueryParam("title") String title ) { ... }
}
```

Ex. accès client : http://…/**book/1234**

Annotations

- @Path : peut être spécifié au niveau classe + méthode
- @PathParam : attribut (entre accolades) de chemin
- @QueryParam : paramètre de la requête HTTP
- @HeaderParam : en-tête de la requête HTTP

Expressions régulières Java et @Path

```
@Path("/myresource")
public class MyResource {
  @GET
  @Path("{var: .*}/stuff")
  public String get( @PathParam("var") String var ) { ... } }
```

Ex. accès client : http://…/myresource/foo/stuff
Ex. accès client : http://…/myresource/foo/bar/stuff
Ex. accès client : http://…/myresource/stuff

Différents types de contenus

En fonction de l'en-tête HTTP Accept spécifié par le client

```
@Path("/library")
public class LibraryResource {

  @GET
  @Path("/books")
  @Produces("text/html")
  public String getBooksHTML() { ... }

  @GET
  @Path("/books")
  @Produces("application/json")
  public String getBooksJSON() { ... }
}
```


Conclusion

- principe simple
- mise en oeuvre légère

- de nombreux sites proposent un accès via REST
  - Amazon, Delicious, Facebook, Flickr, Google, Twitter, Yahoo

- pour plus d'informations
  - documentation en ligne : http://jersey.java.net

SOAP

Lionel Seinturier

Université Lille 1

Lionel.Seinturier@univ-lille1.fr

SOAP

(promoteurs : IBM + Microsoft)

But : invoquer un service distant

- invoquer un service
- sans se préoccuper de la façon dont le service est implanté

- indépendant des langages
- indépendant des OS
- nombreuses implantations dans ≠ langages
  - en Java : Axis, CXF, JAX-WS, …

Specifications SOAP

1. SOAP envelope specification — quelle méthode ? paramètre ? retour ? erreur ?
2. Data encoding rules — règles de représentation des types de données

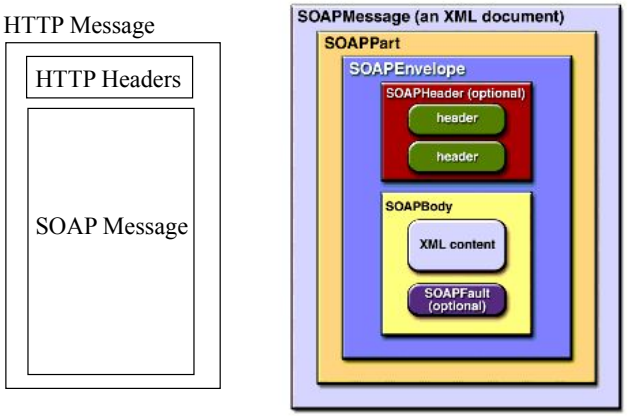
SOAP

- SOAP n'a **aucune** notion orientée objet
  - pas de référence d'objet
  - pas d'instanciation
  - pas de cycle de vie
  - pas de GC
  - pas de variables d'instances
  - pas d'état "conversationnel" i.e. session (+/- = EJB *stateless bean*)

- SOAP ne définit pas de protocole

SOAP : technologie centrée documents XML

Message SOAP



SOAP – Envelope

Enveloppe

• les informations du message (requête ou réponse)
• document XML
• schéma XML : `http://schemas.xmlsoap.org/soap/envelope`

• balise racine `<Envelope>`
• 2 balises principales : `<Header>` et `<Body>`

• balises concernant invocation (opération, paramètres, valeur retour)

Exemple : Invocation du service `euroToDollar` avec la valeur `12.34`

```
<Envelope>
  <Body>
        <euroToDollar>
          <value>12.34</value>
        </euroToDollar>
  </Body>
</Envelope>
```


SOAP – Envelope

Enveloppe

Risque conflit de noms entre balises SOAP & invocations

• namespace XML SOAP-ENV

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
  <SOAP-Env:Body>
        <euroToDollar>
          <value>12.34</value>
        </euroToDollar>
  </SOAP-Env:Body>
</SOAP-ENV:Envelope>
```


SOAP – Envelope

Enveloppe

Exemple HTTP

```
POST /convertisseur HTTP/1.1
Content-Type: text/xml
Content-Length: 999
SOAPAction: http://some.host/SOAPServer/convertisseur

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
  <SOAP-Env:Body>
        <euroToDollar>
          <value>12.34</value>
        </euroToDollar>
  </SOAP-Env:Body>
</SOAP-ENV:Envelope>
```

`SOAPAction` (URI du service web invoqué) doit être présent
mais peut-être vide ⇨ URL POST suffisante pour identifier le service

SOAP – Envelope

Enveloppe

Exemple HTTP

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 999

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
  <SOAP-ENV:Body>
        <euroToDollarResponse>
          <return>10.07</return>
        </euroToDollarResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP – Header

En-tête

• informations supplémentaires sur le message
• métadonnées pour l'exécution du service
• interprétées (ou non) par le serveur

Exemple

```
<SOAP-ENV:Header>
     <Transaction SOAP-ENV:mustUnderstand="1">
       5
     </Transaction>
</SOAP-ENV:Header>
```

Attribut facultatif

• `mustUnderstand="1"` le serveur doit être capable de traiter le message

SOAP – Fault

Erreur

• signale une erreur d'exécution (message de retour)
• balise `<Fault>`

4 sous-balises

`<faultcode>`	type d'erreur
`<faultstring>`	message d'erreur pour l'utilisateur
`<faultactor>`	émetteur de l'erreur en cas d'appels en cascade
`<detail>`	message détaillé pour l'application (ex. *stack trace*)

4 valeurs principales possibles pour `<faultcode>`

`Client`	erreur provenant de la requête du client
`Server`	erreur provenant du serveur
`MustUnderstand`	incapacité à traiter un header mustUnderstand
`VersionMismatch`	namespace de l'enveloppe incorrect

mais valeurs extensibles (même esprit que les code 4xx pour HTTP)
ex : `Client.Authentication`

SOAP – Fault

Erreur

Exemple

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
  <SOAP-ENV:Fault>
     <SOAP-ENV:faultcode>Client</SOAP-ENV:faultcode>
     <SOAP-ENV:faultstring>Méthode inexistante</SOAP-ENV:faultstring>
  </SOAP-ENV:Fault>
</SOAP-ENV:Envelope>
```


SOAP – Data encoding rules

Règles de représentation des types de données

But : typer les données échangées

• types simples : string, int, double, boolean, date, time, enum, tableaux d'octets
• types composés : structures, tableaux

• chaque donnée transmise est typée
  • soit directement dans le message
  • soit en faisant référence à un schéma XML défini de façon externe

SOAP – Data encoding rules

Types simples

Exemple de message avec données typées

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
  <euroToDollarResponse
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
         <return xsi:type="xsd:double">10.07</return>
  </euroToDollarResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


SOAP – Data encoding rules

Types simples

Exemple de message avec typage externe

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
  <SOAP-ENV:Body>
  <euroToDollarResponse
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
         <foo:return xmlns:foo="url schema">10.07</foo:return>
  </euroToDollarResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Schéma

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="return" type="xsd:double" />
</xsd:schema>
```


SOAP – Data encoding rules

Types composés

Structures

```
struct Personne { string nom; int age; }
```

```
<foo:Personne xmlns:foo="uri du schema"
  <foo:nom>Bob</foo:nom>
  <foo:age>45</foo:age>
</foo:Personne>
```

Schéma

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Personne" >
    <xsd:complexType base="xsd:string">
      <xsd:element name="nom" type="xsd:string" />
      <xsd:element name="age" type="xsd:int" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```


SOAP – Programmation Java

Modèle de programmation

Annotation Java 5

`@WebService` : annotation d'une la classe contenant des WS
`@WebMethod` : annotation d'une méthode accessible via un WS
`@WebResult` et `@WebParam` : annotations des paramètres d'un WS

Exemple

```
@WebService
public class MyWS {

  @WebMethod
  public long addUser(
    @WebParam(name="UserName") String name,
    @WebParam(name="UserAge") int age )
 { ... } }
```

SOAP – Programmation Java

Utilisation de WS

@WebServiceRef: référence un web service

```
public class MyClient {

  @WebServiceRef(wsdlLocation="http://localhost:8080/WS/MyWS?wsdl")
  private MyWS service;

  public void foo() {
    long id = service.addUser("Bob",15);
} }
```

Rq : fonctionne dès lors que la classe est prise en compte par une librairie ou un framework supportant l'injection de dépendances

SOAP – Conclusion

Conclusion

- mécanisme simple, facilement implantable (modulo XML)
- sécurité basée sur la sécurité du protocole sous-jacent (ex. HTTPS)
- indépendant langages, OS

mais

- pas de passage d'objets par référence
- pas d'activation à la demande
- pas de gestion de l'exécution des requêtes
- pas de ramasse-miettes
- pas de transaction entre +sieurs invocations (voir extensions)
- fiabilité essentiellement basée sur TCP

⇨ un mécanisme d'invocation de services

WSDL

Lionel Seinturier

Université Lille 1

Lionel.Seinturier@univ-lille1.fr

WSDL

WSDL (Web Service Description Language)

Description de services web

- contrat pour l'utilisation du service
- description XML

Concepts

- type de données
- message
- opération
- port
- liaison (*binding*)

WSDL

Type

- les types de données échangés

```
<wsdl:types>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="PersonneType">
      <xsd:complexType>
        <xsd:element name="nom" type="xsd:string" />
        <xsd:element name="age" type="xsd:int" />
      </xsd:complexType>
  </xsd:schema>
  ...
</wsdl:types>
```


WSDL

Message

- un message échangé
- comprend des paramètres (`part`)
  - élément de type simple
  - référence un `types` précédemment défini

```
<wsdl:message name="AddPersonneRequest">
  <wsdl:part name="nom" type="xsd:string" />
  <wsdl:part name="age" type="xsd:int" />
</wsdl:message>

<wsdl:message name="AddPersonneResponse">
</wsdl:message>

<wsdl:message name="RemovePersonneRequest">
  <wsdl:part name="nom" element="PersonneType" />
</wsdl:message>
```

WSDL

Opération

• un message + un mode d'interaction

• input
  - one-way 1 seul message en input
  - request-response 1 message en input + 1 en output
• output
  - solicit-response 1 seul message en output + 1 en input
  - notification 1 message en output

```
<wsdl:operation name="addPersonne">
  <wsdl:input message="AddPersonneRequest" />
  <wsdl:output message="AddPersonneResponse" />
  <wsdl:fault message="AddPersonneFault" />
</wsdl:operation>
```


WSDL

Port

• ensemble d'opérations

```
<wsdl:portType name="PersonPortType">
  <wsdl:operation name="addPersonne">
  ...
  </wsdl:operation>
  <wsdl:operation name="removePersonne">
  ...
  </wsdl:operation>
</wsdl:portType>
```


WSDL

Liaison (*Binding*)

• spécifie la liaison entre un port et un protocole

```
<wsdl:binding name="PersonneBinding" type="PersonPortType">
  <soap:location="http://localhost:8080/soap/servlet/rpcrouter" />
</wsdl:binding>
```


WSDL

Conclusion WSDL

• langage de définition d'interfaces de services web
• type ⊂ message ⊂ operation ⊂ port ⊂ liaison

• XML !!
• génération automatique de WSDL à partir de Java, EJB, ...