XML et Databases
XML Data binding

INRIA SOPHIA ANTIPOLIS
Philippe.Poulard@inria.fr

XML et Databases

Mapping

- SQL → XML
- XML → SGBDR

Bases de données XML natives

- XML:DB API
- XQuery
- XUpdate

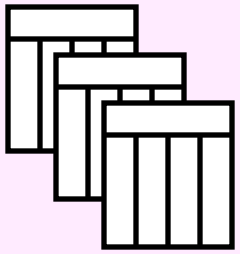
XML Databinding

Unmarshalling

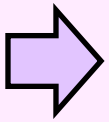
- XML → objet
- Mapping manuel
- JAXB

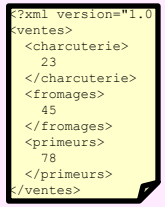
Marshalling

- objet → XML



tables





document XML

Extraire une structure XML à partir d'une requête SQL

Intérêt

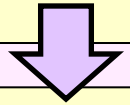
Une fois le modèle XML extrait de la source de données, il est facile de le manipuler, de l'agréger, et de le transformer

Techniques de *mapping*

- Extensions des SGBDR
  - "XMLisation" du SQL
  - Propriétaire
  - Rapidement limité

  → Exemple : Microsoft SQLXML
- Programmation explicite (SQL + SAX)
  - Programmation simple et répétitive
  - Peu évolutif
- Active Tags (expérimental)
  - Le must ! Permet d'obtenir sans limitation n'importe quelle structure XML
  - Assez simple à utiliser : mixe SQL avec des tags fonctionnels et des tags structurels

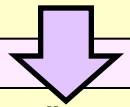
```
SELECT Customers.CustomerID, Orders.OrderID, Orders.OrderDate
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerID
FOR XML RAW
```



```
<row CustomerID="ALFKI" OrderID="10643" OrderDate="1997-08-25T00:00:00"/>
<row CustomerID="ANATR" OrderID="10308" OrderDate="1996-09-18T00:00:00"/>
<row CustomerID="ANATR" OrderID="10625" OrderDate="1997-08-08T00:00:00"/>
<row CustomerID="AROUT" OrderID="10355" OrderDate="1996-11-15T00:00:00"/>
```

Les colonnes des tables deviennent des attributs qui ont le même nom que les colonnes
Les noms des éléments sont arbitraires

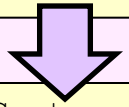
```
SELECT C.CustomerID, O.OrderID, O.OrderDate
FROM Customers C LEFT OUTER JOIN Orders O ON C.CustomerID = O.CustomerID
ORDER BY C.CustomerID
FOR XML RAW
```



```
<row CustomerID="BONAP" OrderID="11076" OrderDate="1998-05-06T00:00:00"/>
<row CustomerID="FISSA"/>
<row CustomerID="PARIS"/>
<row CustomerID="RICSU" OrderID="11075" OrderDate="1998-05-06T00:00:00"/>
```

Les valeurs `null` ne produisent pas d'attributs

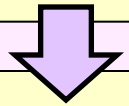
```
SELECT Customers.CustomerID, Orders.OrderID, Customers.ContactName
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
FOR XML AUTO
```



```
<Customers CustomerID="ALFKI"
      ContactName="Maria Anders">
  <Orders OrderID="10643"/>
  <Orders OrderID="10692"/>
  <Orders OrderID="10702"/>
  <Orders OrderID="10835"/>
  <Orders OrderID="10952"/>
  <Orders OrderID="11011"/>
</Customers>
```

Les noms des éléments sont les noms des tables
Les noms des attributs sont les noms des colonnes

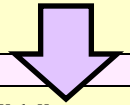
```
SELECT Customers.CustomerID, Orders.OrderID, Customers.ContactName
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
FOR XML AUTO, ELEMENTS
```



```
<Customers>
  <CustomerID>ALFKI</CustomerID>
  <ContactName>Maria Anders</ContactName>
        <Orders><OrderID>10643</OrderID></Orders>
        <Orders><OrderID>10692</OrderID></Orders>
        <Orders><OrderID>10702</OrderID></Orders>
        <Orders><OrderID>10835</OrderID></Orders>
        <Orders><OrderID>10952</OrderID></Orders>
        <Orders><OrderID>11011</OrderID></Orders>
</Customers>
```

Les noms des éléments sont les noms des tables et des colonnes

```
SELECT Employees.EmployeeID, LastName, FirstName,
       OrderID, OrderDate, Orders.EmployeeID
FROM Orders, Employees
WHERE Orders.EmployeeID = Employees.EmployeeID
ORDER BY Employees.EmployeeID
FOR XML AUTO
```



```
<Employees EmployeeID="1" LastName="Davolio" FirstName="Nancy">
  <Orders OrderID="10258" OrderDate="1996-07-17T00:00:00" EmployeeID="1"/>
  <Orders OrderID="10270" OrderDate="1996-08-01T00:00:00" EmployeeID="1"/>
</Employees>
<Employees EmployeeID="2" LastName="Fuller" FirstName="Andrew">
  <Orders OrderID="10248" OrderDate="1996-07-04T00:00:00" EmployeeID="5"/>
  <Orders OrderID="10249" OrderDate="1996-07-05T00:00:00" EmployeeID="6"/>
</Employees>
```

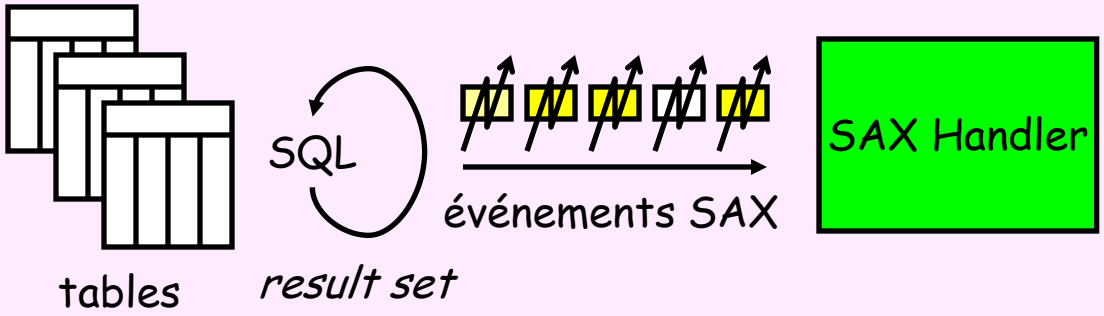
Tout se fait automatiquement ; on ne peut pas :

☹
- Choisir avec précision les niveaux d'imbrication
- Ajouter des éléments conteneurs
- Décider arbitrairement de ce qui doit être élément et de ce qui doit être attribut
- Nommer soit-même les noms d'éléments ou d'attributs
  (par exemple, avec les valeurs d'une colonne)
- Utiliser d'autres fonctions ou réaliser d'autres calculs que ceux permis par SQL

Comment faire si la structure imposée ne me convient pas ?

→ Recours à XSLT pour obtenir la structure souhaitée

Générer des événements SAX pendant le parcours du résultat d'une requête SQL



Pseudo-code:

```
C = Connect( "order-db", "localhost:1234", "toto", "secret" )
Q = C.Query( "SELECT * FROM orders WHERE id= " + ID )
H = SAXHandler( … )
H.StartDocument()
A = Attributes( "id", ID )
H.StartElement( "order" , A )
For each R in Q.Result {
  A = Attributes( "part-number", R.pn, "quantity", R.qty )
  H.StartElement( "item" , A )
  H.StartElement( "price" )
  H.Characters( R.price )
  H.EndElement( "price" )
  H.StartElement( "product" )
  H.Characters( R.prod )
  H.EndElement( "product" )
  H.EndElement( "item" )
}
H.EndElement( "order" )
H.EndDocument()
```

Par programme, on peut créer directement la structure XML cible, qui peut être arbitrairement complexe

```
<?xml version="1.0" encoding="iso-8859-1"?>
<order id="1234">
  <item part-number="4321" quantity="1">
    <price>4.49</price>
    <product>
      Le rouge &amp;amp; le noir &amp;lt; de Stendhal &amp;gt;
    </product>
  </item>
 <item part-number="9876" quantity="12">
    <price>1.12</price>
```

Pourquoi utiliser SAX plutôt que d'écrire directement des tags ?
Parce qu'on évite tous les écueils syntaxiques :

- gestion de l'encoding
- gestion des échappements
- etc

Laisser ces difficultés au processeur qui prend tout ça en charge très bien

```
Println("<product>");
Println(prod);
Println("</product>");
```

orders				
id	prod	qty	pn	price
1234	Le rouge & le noir < de Stendhal >	1	4321	4.49 €
1234	Pinces à linge rouges	12	9876	1.12 €
1234	Enclume noire	3	5544	29 €
1234	Cheval noir	1	4455	219 €

```
<product>
Le rouge & le noir < de Stendhal >
</product>
```

Erreurs XML ☹

- Programmation XML native
- Possibilité de mixer SQL dans des tags
- Permet de créer immédiatement la structure souhaitée, sans avoir recours à XSLT

Tag structurel

Tag fonctionnel

```
<?xml version="1.0" encoding="iso-8859-1"?>
<order id="{ $sys:env/order-id }"
    xmlns:rdbms="http://www.inria.fr/xml/active-tags/rdbms"
    xmlns:xcl="http://www.inria.fr/xml/active-tags/xcl"
    xmlns:sys="http://www.inria.fr/xml/active-tags/sys">
  <rdbms:connect name="order-db"
      url="…" driver="…" host="…" passwd="…" user="…"/>
  <rdbms:select connexion="{$order-db}" name="orders"
      query="SELECT * FROM orders WHERE id= ? ">
    <xcl:param value="{ $sys:env/order-id }"/>
  </rdbms:select>
  <xcl:for-each select="{ $orders }">
    <item part-number="{ pn }" quantity="{ qty }">
      <price currency="{ currency }">{ price }</price>
      <product>{ product }</product>
    </item>
  </xcl:for-each>
</order>
```

Expression XPath

```
<?xml version="1.0" encoding="iso-8859-1"?>
<order id="1234">
  <item part-number="4321" quantity="1">
    <price>4.49</price>
    <product>Le rouge &amp;amp; le noir &amp;lt; de Stendhal &amp;gt;</product>
  </item>
 <item part-number="9876" quantity="12">
    <price>1.12</price>
    <product>Pinces à linges rouges</product>
  </item>
```

☑ XML ☺

Extension SQL native de l'ISO

ISO/IEC 9075-14:2003
Information technology -- Database languages -- SQL -- Part 14: XML-Related Specifications (SQL/XML)

Oracle, SQLServer, PostgreSQL, etc

Type XML

Fonctions de construction:

- XMLELEMENT
- XMLATTRIBUTES
- XMLFOREST
- XMLCONCAT
- XMLNAMESPACES
- XMLCOMMENT
- XMLPI
- XMLDOCUMENT
- XMLAGG
- etc.

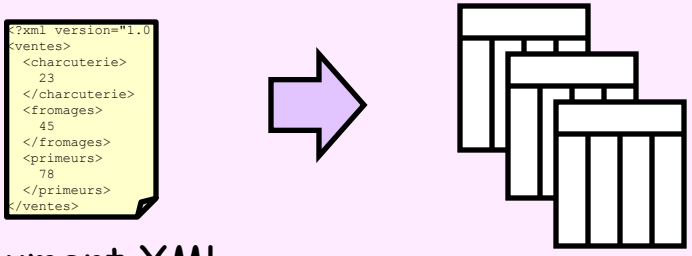
Fonctions d'interrogation:

- XMLQUERY
- XMLTABLE

http://www.postgresql.org/docs/current/static/functions-xml.html

```
SELECT xmlelement(name foo, xmlattributes('xyz' as bar),
                            xmlelement(name abc),
                            xmlcomment('test'),
                            xmlelement(name xyz));

                  xmlelement
----------------------------------------------
 <foo bar="xyz"><abc/><!--test--><xyz/></foo>
```

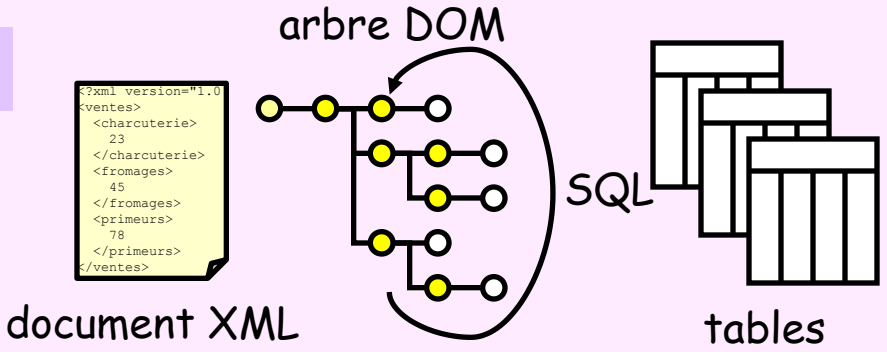


document XML

tables

Opération inverse de la précédente

Modèle spécifique



Il faut faire un programme qui parcours le document et mette à jour les tables

- Chaque programme ne traite qu'**un cas spécifique**
- Long à coder
- Peu évolutif
- Difficilement maintenable

Modèle généraliste

Faire en sorte que quel que soit le document XML, on puisse le stocker dans la base

Comment stocker ?

- CLOB
- {*parent*, *child*}
- {*path*, *value*}

CLOB → grande chaîne

Tout le XML stocké dans une colonne

orders	
id	xml
1234	<?xml version="1.0" enco <order id="1234"> <item part-number="432 <price>4.49</price> <product>Le rouge &a </item> <item part-number="9876 <price>1.12</price> <product>Pinces à li </item> </order>

- XML monolithique,
- pas d'accès à la structure,
- même la recherche plein-texte peut-être inefficace

`Le rouge &amp; le noir` → "Le rouge & le noir" not found

`&#339;ufs` → "œufs" not found

`Poivre<!----> et sel` → "Poivre et sel" not found

⇒ à éviter

{parent, child}

```
<?xml version="1.0" encoding="iso-8859-1"?>
<order id="1234">
  <item part-number="4321" quantity="1">
    <price>4.49</price>
    <product>Le rouge &amp; le noir &lt; de Stendhal &gt;</product>
  </item>
  <item part-number="9876" quantity="12">
    <price>1.12</price>
    <product>Pinces à linges rouges</product>
  </item>
</order>
```

documents
node
1
37
123
175

nodes	
parent	child
1	2
1	3
1	4
3	5
3	6
3	7
3	8
7	9

values		
node	name	value
1	order	null
2	id	1234
3	item	null
4	item	null
5	part-number	4321
6	quantity	1
7	price	null
8	product	null
9	null	4.49

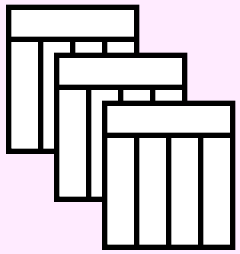
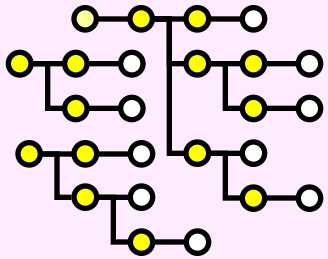
élément
attribut
text

{path, value}

```
<?xml version="1.0" encoding="iso-8859-1"?>
<order id="1234">
  <item part-number="4321" quantity="1">
    <price>4.49</price>
    <product>Le rouge &amp;amp; le noir &amp;lt; de Stendhal &amp;gt;</product>
  </item>
  <item part-number="9876" quantity="12">
    <price>1.12</price>
    <product>Pinces à linges rouges</product>
  </item>
</order>
```

nodes	
path	value
/order	null
/order/@id	1234
/order/item[1]	null
/order/item[1]/@part_number	4321
/order/item[1]/@quantity	1
/order/item[1]/price	null
/order/item[1]/price/text()	4.49
/order/item[1]/product	null
/order/item[1]/product/text()	Le rouge & le noir < de Stendhal >
/order/item[2]	null
/order/item[2]/@part_number	9876

Une base de données XML ou relationnelle est une application qui masque la façon dont sont stockées les données : elle n'offre qu'une vue logique et un langage de requête **adaptée** à cette vue logique

	SGBDR	BD XML
Vue logique	 tables	 forêt DOM collections
Langage de requête	SQL	XPath XQuery
Services	Transactions ACID	Transactions (non standardisé) Full-text, scripting, etc

Bases de données XML natives

	Avantages	Inconvénients
SGBDR	Modèle éprouvé Gestion des transactions	Totalement inadapté aux données récursives Limité aux données tabulaires
BD XML	Peut reproduire un modèle relationnel	Pas de modèle commun pour gérer des transactions •verrouillage : granularité ? •niveaux ACID

Bases de données XML natives

Produit	Editeur	License	DB Type
Berkeley DB XML	Sleepycat Software	Open Source	Key-value
dbXML	dbXML Group	Open Source	Proprietary
eXist	Wolfgang Meier	Open Source	Natif XML
ozone	ozone-db.org	Open Source	Object-oriented
Tamino	Software AG	Commercial	Proprietary. Relational through ODBC
X-Hive/DB	X-Hive Corporation	Commercial	Proprietary. Relational through JDBC
Xindice	Apache Software Foundation	Open Source	Proprietary (Model-based)
BaseX	Konstanz University	Open Source	Natif XML

et bien d'autres encore…

XML:DB initiative : 2 projet principaux :
- XML:DB API
- XUpdate

http://xmldb-org.sourceforge.net/

XML:DB API offre une abstraction des bases de données XML natives

Concepts de base

- ***Driver***

S'apparente aux drivers de JDBC

```
String driver = "org.acme.xmldb.driver.DatabaseImpl";
```

```
Class c = Class.forName(driver);
Database database = (Database) c.newInstance();
DatabaseManager.registerDatabase(database);
```

- **Collections**

Unité de stockage élémentaire pour les documents XML

```
Collection col = DatabaseManager.getCollection("xmldb:acme://host:port/mydb/");
```

- **Services**

Permet d'agir sur la base

```
XPathQueryService service = (XPathQueryService) col.getService("XPathQueryService", "1.0");
```

- **Ressources**

```
Node XMLResource#getContentAsDOM();
Object BinaryResource#getContext();
```

Documents binaires et XML

- ***Core levels***

Regroupement de fonctionnalités

```
if ( ! database.getConformanceLevel().equals("1") ) {
  throw new XMLDBException( ErrorCodes.NOT_IMPLEMENTED );
}
```

- Modularité
- Extensibilité

Exemple

```
String uri = "xmldb:provider://host:port/db/addressbook";
Collection col = DatabaseManager.getCollection( uri );
String xpath = "//person[firstname='Bill']";
XPathQueryService service = (XPathQueryService) col.getService("XPathQueryService", "1.0");

ResourceSet resultSet = service.query(xpath);
ResourceIterator results = resultSet.getIterator();
while (results.hasMoreResources()) {
  XMLResource res = (XMLResource) results.nextResource();
  Node node = res.getContentAsDOM();
  …/…
}
```

Les requêtes ne retournent pas un seul document, mais un ensemble de documents

URI XML:DB

XML:DB offre aussi la possibilité d'adresser les ressources de la base grâce à une URI

```
xmldb:provider://host:port/path/to/document.xml
xmldb:provider://host:port/path/to/image.jpg
```

Stockage des ressources

Il faut spécifier à XML:DB si la ressource est binaire ou XML

```
XMLResource res = (XMLResource) col.createResource(id, XMLResource.RESOURCE_TYPE);
res.setContentAsDOM( node );
Col.storeResource( res );
```

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

```
xquery version "1.0";
declare namespace rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#";
declare namespace dc="http://purl.org/dc/elements/1.1/";

for $p in distinct-values(doc('/db/library/biblio.rdf')//dc:creator)
let $books := //rdf:Description[dc:creator&=$p]
return
        <result>
                <creator titles="{count($books)}">{$p}</creator>
                {for $b in $books return $b/dc:title}
        </result>
```

résultats

```
<result>
<creator titles ="1" > Sinclair, Bruce </creator>
<dc:title>A Centennial History of the American Society of Mechanical
Engineers 1880-1980</dc:title>
</result>
```

```
<result>
<creator titles="1">Banham, Reyner</creator>
<dc:title>A Concrete Atlantis : U.S. Industrial Building and European
Modern Architecture 1900-1925</dc:title>
</result>
```

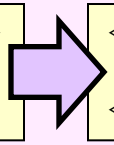
```
<result >
<creator titles ="1" > Furter, William F. </ creator >
<dc:title > A century of chemical engineering </ dc:title >
</ result >
```

Mise à jour des documents XML avec des opérations XML

http://xmldb-org.sourceforge.net/xupdate/

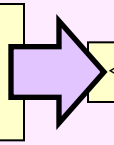
```
<xupdate:XXX xmlns:xupdate="http://www.xmldb.org/xupdate">
```

```
<xupdate:element name="address">
    <town>San Francisco</town>
</xupdate:element>
```



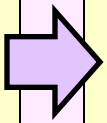
```
<address>
  <town>San Francisco</town>
</address>
```

```
<xupdate:element name="address">
    <xupdate:attribute name="id">2</xupdate:attribute>
</xupdate:element>
```



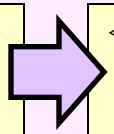
```
<address id="2"/>
```

```
<xupdate:append select="/addresses" child="last()">
    <xupdate:element name="address">
        <town>San Francisco</town>
    </xupdate:element>
</xupdate:append>
```



```
<addresses>
  <address>
    <town>Los Angeles</town>
  </address>
  <address>
    <town>San Francisco</town>
  </address>
</addresses>
```

```
<xupdate:update select="/addresses/address[2]/town">
    New York
</xupdate:update>
```



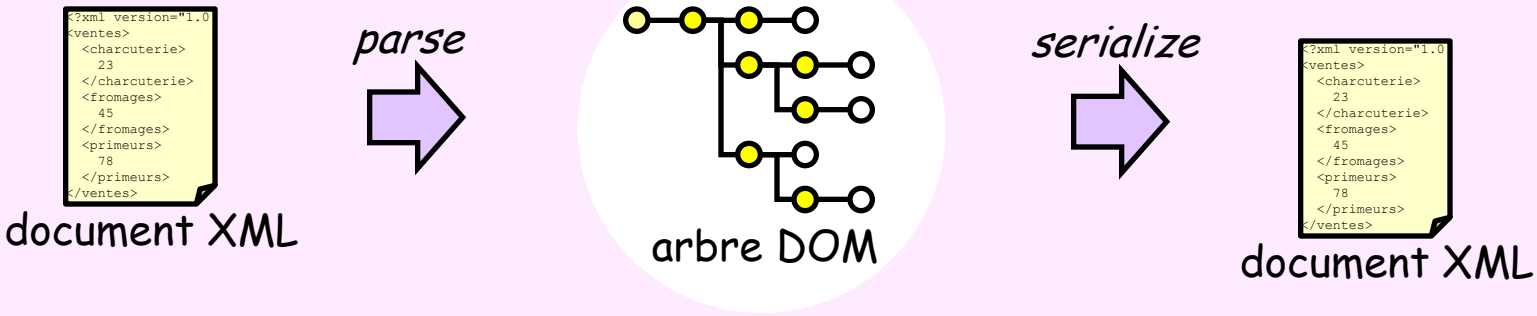
```
<addresses>
  <address>
    <town>Los Angeles</town>
  </address>
  <address>
    <town>New York</town>
  </address>
</addresses>
```

Autre opérations :

```
<xupdate:insert-before>
<xupdate:insert-after>
<xupdate:remove>
<xupdate:rename>
<xupdate:variable>
<xupdate:value-of>
<xupdate:if>
```

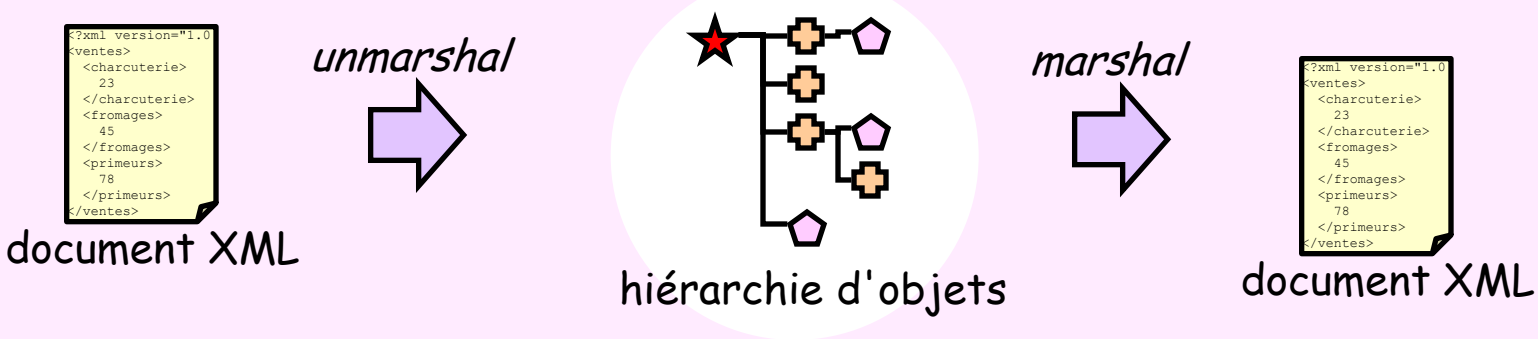
inutilisable →

Modèle généraliste



Au lieu d'obtenir un modèle de données XML, on cherche à instancier des classes qui correspondent aux éléments rencontrés

Modèle spécifique



Des techniques permettent de réaliser la désérialisation (*unmarshal*) automatiquement

- Mapping automatique
- Mapping manuel

→ Castor

http://castor.exolab.org

JAXB

Mapping manuel : exemple

- Définir son jeu de balises et d'attribut et l'espace de nommage
- Associer les structures XML à des classes

```
<?xml version="1.0"
  encoding="ISO-8859-1"?>
<root>
  <foo param="…" />
  <foo param="…">
    <bar value="…" type="…">
      Blah
    </bar>
    <foo param="…" />
  </foo>
  <bar value="…" type="…">
      Blah blah blah
  </bar>
</root>
```

```
try {
  DocumentBuilderFactory
    factory = DocumentBuilderFactory.newInstance();
  factory.setNamespaceAware(true);
  DocumentBuilder
    builder = factory.newDocumentBuilder();
  Document doc = builder.parse(fileName);
  Element root = doc.getDocumentElement();
  RootAction rootAction = null;
  if (root.getNodeName().equals("root")) {
    rootAction = RootAction.unmarshal(root);
  }
  rootAction.run();
} catch (ParserConfigurationException e) {
  // report configuration error
} catch (SAXException e) {
 // report parsing error
} catch (IOException e) {
 // report IO error
}
```

```
public class AbstractAction {
  Vector actions;
  AbstractAction parent;
  public static AbstractAction unmarshal(AbstractAction parent, Element e) {
    for (Node n=e.getFirstChild(); n!=null; n=n.getNextSibling()) {
      if (n.getNodeType()==Node.ELEMENT_NODE) {
        if (n.getNodeName().equals("foo")) {
          parent.addAction(FooAction.unmarshal((Element) n));
        } else if (n.getNodeName().equals("bar")) {
          parent.addAction(BarAction.unmarshal((Element) n));
        }
      }
    }
    return parent;
  }
  private final void addAction(AbstractAction action) {
    actions.add(action);
    action.parent = this;
  }
  public void run() {
    // Execute all dependant abstract actions
    doItBefore();
    for (Enumeration action = actions.elements() ; action.hasMoreElements() ;) {
      ((AbstractAction) action.nextElement()).run();
    }
    doItAfter();
  }
  public void doItBefore() {}
  public void doItAfter() {}
}
```

Mapping manuel


```
public class FooAction extends AbstractAction {
  private String fooParam;
  // Creates a new instance of fooAction
  private FooAction() {
    actions = new Vector();
  }
  public static FooAction unmarshal(Element e) {
          FooAction fooAction = new FooAction();
          fooAction.fooParam = e.getAttribute("param");
          AbstractAction.unmarshal(fooAction, e);
          return fooAction;
  }
  public void doItBefore() {
    // do foo actions before running dependant sub-actions
  }
  public void doItAfter() {
    // do foo actions after running dependant sub-actions
  }
}
```

Les actions peuvent être :

- connection à une base de données
- mise à jour d'une table
- envoi d'un mail
- action sur un modèle de données
- …

On peut enrichir le code pour conditionner l'exécution d'une action en fonction du résultat de la précédente

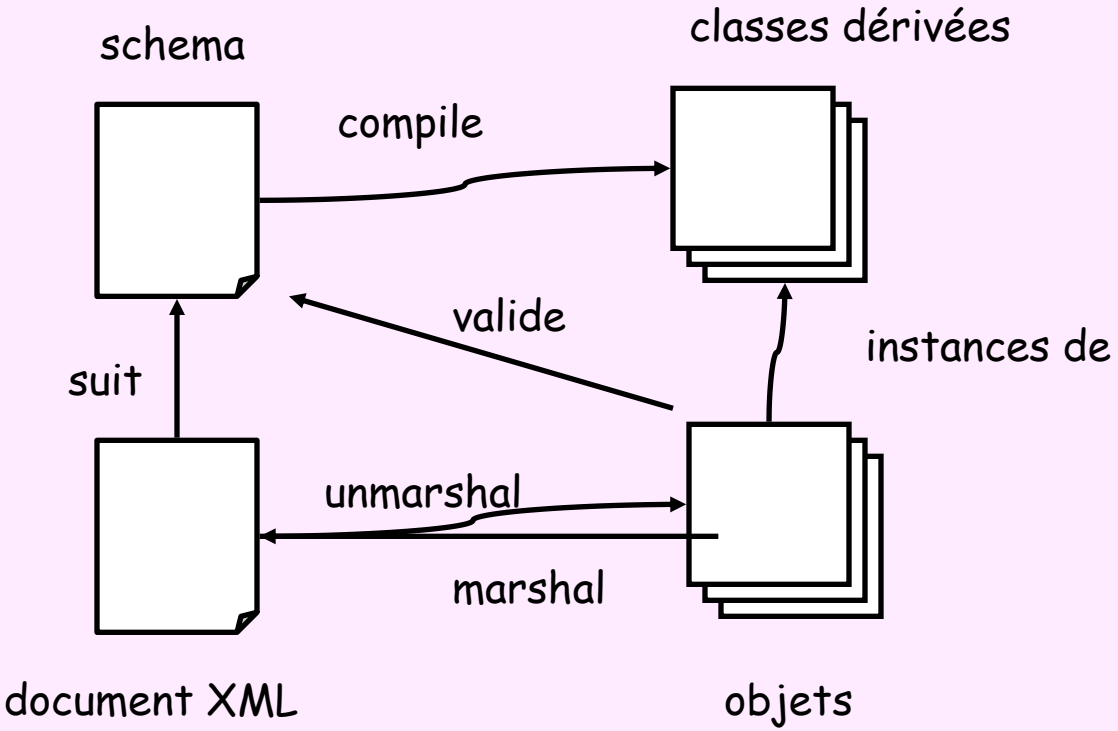
```
public class RootAction extends AbstractAction {
private RootAction() {
    actions = new Vector();
    parent = null;
  }
  public static RootAction unmarshal(Element e) {
    RootAction root = new RootAction();
    AbstractAction.unmarshal(root, e);
    return root;
  }
}
```

On peut réaliser des processeurs *multi-threads* dont le document XML est analysé à l'initialisation du programme, et exécuté pour chaque *thread* avec un jeu de données différent (attention à la façon d'isoler les données)
→ servlet

…/…

JAXB

Utilisation d'un schéma XML pour *mapper* directement les tags à une classe



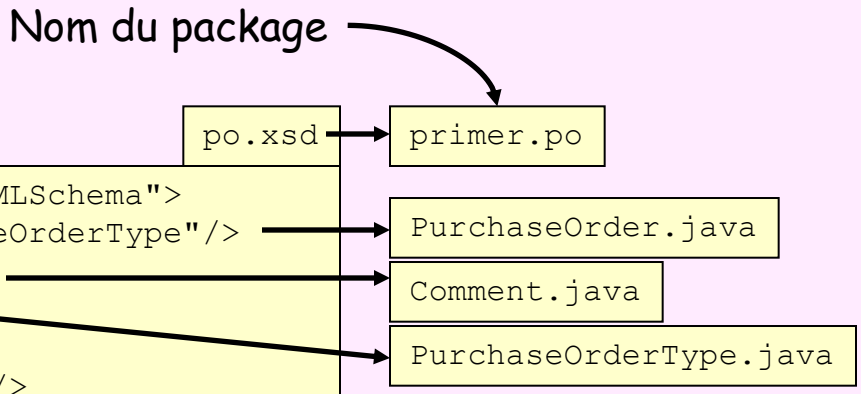
Java permettait déjà de sérialiser et désérialiser des instances d'objet, mais cela se faisait en binaire. Avec JAXB, c'est en XML.

Exemple JAXB

Données

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>123 Maple Street</street>
    <city>Mill Valley</city>
    <state>CA</state>
    <zip>90952</zip>
  </shipTo>
  <billTo country="US">
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    <city>Old Town</city>
    <state>PA</state>
    <zip>95819</zip>
  </billTo>
  <comment>Hurry, my lawn is going wild!</comment>
  <items>
    <item partNum="872-AA">
      <productName>Lawnmower</productName>
      <quantity>1</quantity>
      <USPrice>148.95</USPrice>
      <comment>Confirm this is electric</comment>
    </item>
    <item partNum="926-AA">
      <productName>Baby Monitor</productName>
      <quantity>1</quantity>
      <USPrice>39.98</USPrice>
      <shipDate>1999-05-21</shipDate>
    </item>
  </items>
</purchaseOrder>
```

Schéma




```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="purchaseOrder" type="PurchaseOrderType"/>
<xsd:element name="comment" type="xsd:string"/>
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items" type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
<xsd:complexType name="USAddress">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="city" type="xsd:string"/>
    <xsd:element name="state" type="xsd:string"/>
    <xsd:element name="zip" type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
```

.../...

…/…

Schéma

```
<xsd:complexType name="Items">
  <xsd:sequence>
    <xsd:element name="item" minOccurs="1" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="productName" type="xsd:string"/>
          <xsd:element name="quantity">
            <xsd:simpleType>
              <xsd:restriction base="xsd:positiveInteger">
                <xsd:maxExclusive value="100"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:element>
          <xsd:element name="USPrice" type="xsd:decimal"/>
          <xsd:element ref="comment" minOccurs="0"/>
          <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="partNum" type="SKU" use="required"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<!-- Stock Keeping Unit, a code for identifying products -->
<xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

Items.java

Items.ItemType

Utilisation

```
JAXBContext jc = JAXBContext.newInstance( "primer.po" );
Unmarshaller u = jc.createUnmarshaller();
PurchaseOrder po = (PurchaseOrder) u.unmarshal( new FileInputStream( "po.xml" ) );
USAddress address = po.getShipTo();
displayAddress(address);
Items items = po.getItems();
displayItems(items);
```

```
public static void displayAddress( USAddress address ) {
  // display the address
  System.out.println( "\t" + address.getName() );
  System.out.println( "\t" + address.getStreet() );
  System.out.println( "\t" + address.getCity() +
    ", " + address.getState() + " " + address.getZip() );
  System.out.println( "\t" + address.getCountry() + "\n");
}
public static void displayItems( Items items ) {
  // the items object contains a List of
  //primer.po.ItemType objects
  List itemTypeList = items.getItem();
  for(Iterator iter = itemTypeList.iterator(); iter.hasNext();) {
    Items.ItemType item = (Items.ItemType)iter.next();
    System.out.println( "\t" + item.getQuantity() +
      " copies of \"" + item.getProductName() +
      "\"" );
  }
}
```