Schémas XML

Plan

1. Introduction : objectifs et histoire des schémas XML
2. Types simples
3. Types complexes
4. Dérivation de types
5. Validation par schéma

Références :

XML Schema Part 1: Structures
http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

XML Schema Part 2: Datatypes
http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

Limitations des DTD

Les DTDs ont des limitations :

- Trop contraignantes : tout doit être défini
- Pas assez contraignantes : nombres d'occurences, par exemple
- Très peu de contraintes sur les chaînes contenues dans les éléments et sur les valeurs d'attributs
- Pas d'héritage
- Syntaxe particulière, différente de la syntaxe de balisage des instances

Objectif

Développer un mécanisme de schéma (au sens des bases de données) pour exprimer des contraintes sur un type de document

- Contraintes structurales
  - comme les DTD : ordre, occurence des éléments, attributs
  - mais en plus : intégration des espaces de noms, contraintes incomplètes, héritage
- Typage des données : chaînes de caractères, entiers, dates, données binaires, uri, etc.
- Mode d'expression des schémas : structure d'éléments et attributs, syntaxe XML
- Validation des instances par rapport aux schémas

Histoire

Les schémas XML ont été créés au W3C dans l'activité XML

- octobre 1998 : création du XML Schema WG
- février 1999 : publication de la note "XML Schema Requirements"
- mars 1999 : publication des premiers drafts
- mai 2001 : recommandation XML Schema 1.0 en 3 documents
  - Primer
  - Structures
  - Datatypes
- juillet 2002 : nouveau charter (maintenance)

Structure d'un schéma XML

Un schéma XML est un document XML

Ses éléments de haut niveau servent essentiellement à déclarer

- les éléments du type de document défini
- les types
- les notations

Les éléments subordonnés servent essentiellement à déclarer

- les modèles de contenus
- les attributs applicables aux éléments

Exemple de schéma XML

DTD

```
<!ELEMENT text (#PCDATA | emph | name)*>
<!ATTLIST text timestamp NMTOKEN #REQUIRED>
```

Schéma XML

```
<element  name="text">
    <type content="mixed">
       <element ref="emph"/>
       <element ref="name"/>
       <attribute name="timestamp"
                  type="date"
                  minOccurs="1"/>
   </type>
</element>
```

D'après H. Thomson

Caractéristiques des schémas XML

- Distinction entre définition de types et définition des éléments qui peuvent apparaître dans les instances
- Les éléments ont un nom et un type
- Types simples et complexes
- Mécanismes de construction de types
- Classes d'équivalence
- Construction de schémas modulaire
- Mécanismes d'extension
- Facilités de documentation des schémas

Exemple : Purchase Order Instance

Extrait de « XML Schema Part 0: Primer »

```
<?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>
```

Exemple : Purchase Order Schema

Extrait de « XML Schema Part 0: Primer »

```
<xsd:schema   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:annotation>
   <xsd:documentation xml:lang="en">
    Purchase order schema for Example.com.
    Copyright 2000 Example.com. All rights reserved.
   </xsd:documentation>
  </xsd:annotation>

  <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>

  <xsd:complexType  name="Items">
   <xsd:sequence>
     <xsd:element  name="item"  minOccurs="0"  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>
```

Types

Un type est un ensemble de contraintes sur les contenus d'élément et les valeurs d'attribut

- types simples : contraintes sur les valeurs des chaînes de caractères
  - pour les éléments qui ne contiennent que du contenu de base (texte, nombres, dates, etc.)
  - pour les attributs
- types complexes : contraintes sur les éléments qui contiennent d'autres éléments et/ou ont des attributs

Les attributs ont toujours un type simple.

Un certain nombre de types simples sont prédéfinis : chaîne de caractères, nombre décimal, date, entier positif, etc.

Types simples

Types simples : types des éléments qui ne contiennent pas d'autres éléments ou types des attributs

Les types simples sont prédéfinis ou dérivés des types prédéfinis

Types prédéfinis :

string, normalizedString, token, byte, unsignedByte, base64Binary, hexBinary, integer, positiveInteger, negativeInteger,

nonNegativeInteger, nonPositiveInteger, int, unsignedInt, long, unsignedLong, short, unsignedShort, decimal, float, double, boolean, time, dateTime, duration, date, gMonth, gYear, gYearMonth, gDay, gMonthDay, Name, QName, NCName, anyURI, language, ID, IDREF, IDREFS, ENTITY, ENTITIES, NOTATION, NMTOKEN, NMTOKENS

Hiérarchie des types prédéfinis

Types simples dérivés

On peut créer de nouveaux types simples en dérivant des types simples existants (prédéfinis ou dérivés)

Un nouveau type simple peut être défini par

- **restriction** : sous−ensemble des valeurs d'un type existant
- **liste** : liste de valeurs d'un type existant
- **union** : choix de valeurs parmi un ensemble de types existant

Un nouveau type simple est déclaré par un élément `simpleType` qui contient un élément `restriction`, `list` ou `union`

Restriction de type simple

La restriction d'un type simple se fait par *facettes* :

- pour les nombres, dates, durées :
  valeur minimum, valeur maximum, nombre de chiffres, patterns, énumération
- pour les chaînes, noms, ID :
  patterns (expressions régulières), longueur, longuueur minimum, longueur maximum

Exemples :

```
<xsd:simpleType  name="myInteger">
   <xsd:restriction base="xsd:integer">
     <xsd:minInclusive value="10000"/>
     <xsd:maxInclusive value="99999"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType  name="SKU">
   <xsd:restriction base="xsd:string">
     <xsd:pattern value="\d{3}−[A−Z]{2}"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType  name="USState">
   <xsd:restriction base="xsd:string">
     <xsd:enumeration value="AK"/>
     <xsd:enumeration value="AL"/>
     <xsd:enumeration value="AR"/>
     <!−− and so on ... −−>
   </xsd:restriction>
</xsd:simpleType>
```

Types simples liste

La plupart des types simples sont *atomiques* (valeur indivisible)

Un type liste est un type simple qui prend pour valeur une liste de valeurs atomiques

Trois types liste prédéfinis : `NMTOKENS`, `IDREFS`, `ENTITIES`

On peut créer de nouveaux types liste par dérivation de types atomiques existant

Exemple :

```
<xsd:simpleType  name="listOfMyIntType">
   <xsd:list  itemType="myInteger"/>
</xsd:simpleType>
```

Une valeur possible pour le type listOfMyIntType :

```
20003 15037 95977 95945
```

Types simples union

Un type union est un type simple qui prend une valeur constituée d'une ou plusieurs instances d'un type choisi dans une union de types atomiques ou liste

Exemple :

```
<xsd:simpleType  name="zipUnion">
   <xsd:union  memberTypes="USState  listOfMyIntType"/>
</xsd:simpleType>
```

Une valeur possible pour le type zipUnion : `CA`

Une valeur possible pour le type zipUnion : `95630`

```
95977 95945
```

Types complexes

Types complexes : types des éléments qui contiennent d'autres éléments et/ou ont des attributs

Les types complexes sont déclarés par l'élément `complexType` qui contient

- des déclarations d'éléments contenus
  - déclarations directes
  - déclarations par référence à un autre élément
- des déclarations d'attributs

Les éléments sont déclarés par l'élément `element`

Les attributs sont déclarés par l'élément `attribute`

Exemple de type complexe (1)

```
<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>
```

Exemple de type complexe (2)

```
<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>
```

L'élément `comment` est défini ailleurs dans le schéma comme un élément global :

```
<xsd:element  name="comment"  type="xsd:string"/>
```

Déclarations globales

Les éléments de déclaration fils de l'élément `schema` déclarent des éléments et des attributs *globaux*

Les éléments et attributs globaux peuvent être référencés par d'autres déclarations avec l'attribut `ref` à la place de `type`

Tout élément global peut être la racine d'une instance

Les déclarations globales ne peuvent pas

- contenir de références (attribut `ref`)
- porter de contraintes de cardinalité (attributs `minOccurs`, `maxOccurs`, `use`)

Contraintes d'occurence

Déclaration d'éléments

Le nombre d'occurences est contrôlé par les attributs `minOccurs` et `maxOccurs`

Valeurs possibles :

- `minOccurs` : entier positif ou nul
- `maxOccurs` : entier positif ou nul, `unbounded`
- `minOccurs`≤ `maxOccurs`

Valeurs par défaut des deux attributs : 1 (une occurence obligatoire si les deux attributs sont omis)

Déclarations d'attributs

L'occurence est contrôlée par l'attribut `use`

Valeurs possibles : `required`, `optional`, `prohibited`

Valeurs par défaut

Les valeurs par défaut sont définies par l'attribut `default`

Déclaration d'éléments

L'attribut `default` spécifie la valeur à considérer si l'élément est vide dans l'instance

Aucune action si l'élément est absent dans l'instance

Déclarations d'attributs

L'attribut `default` spécifie la valeur à considérer si l'attribut est absent dans l'instance

Seulement si l'attribut est optionnel : `use="optional"`

Valeurs fixes

Une valeur fixe peut être spécifiée dans les déclarations d'éléments et d'attributs

Spécifiée par l'attribut `fixed`

Les concepts de valeur fixe et de valeur par défaut sont exclusifs

Exemple :

```
<xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
```

Si l'attribut `contry` est présent dans une instance sa valeur *doit* être `"US"`

S'il est absent le processeur de schémas fournit `country="US"`

Définitions de types anonymes

Pour plus de concision, un type utilisé une seule fois peut être défini en même temps qu'il est utilisé

Exemple : les éléments `item` et `quantity` sont définis avec des types anonymes

```
<xsd:complexType name="Items">
  <xsd:sequence>
   <xsd:element name="item" minOccurs="0" 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>
```

Contenu simple, type complexe

Un élément qui contient une valeur de type simple, mais possède des attributs

Exemple de dérivation d'un type complexe à partir d'un type simple :

```
  <xsd:element name="internationalPrice">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="xsd:decimal">
          <xsd:attribute name="currency" type="xsd:string"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>
```

Exemple d'nstance :

```
  <internationalPrice  currency="EUR">423.46</internationalPrice>
```

Contenu mixte

Un élément qui contient, au même niveau, des chaînes de caractères et des sous−éléments

Exemple de modèle de contenu mixte :

```
<xsd:element  name="letterBody">
  <xsd:complexType  mixed="true">
   <xsd:sequence>
    <xsd:element name="salutation">
     <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="quantity"     type="xsd:positiveInteger"/>
    <xsd:element  name="productName"  type="xsd:string"/>
    <xsd:element name="shipDate"     type="xsd:date" minOccurs="0"/>
   <!−− etc. −−>
   </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Exemple d'instance :

```
<letterBody>
   <salutation>
     Dear Mr.
      <name>Robert Smith</name>.
   </salutation>
   Your order of
    <quantity>1</quantity>
    <productName>Baby  Monitor</productName>
   shipped from our warehouse on
    <shipDate>1999−05−21</shipDate>.  ....
</letterBody>
```

Contenu non contraint

Un élément qui n'a aucune contrainte sur son contenu

Type de base d'où sont dérivés tous les types simples et complexes :
`anyType`

`anyType` est le type par défaut

Exemple de modèle de contenu non contraint :

```
   <xsd:element  name="anything"  type="xsd:anyType"/>
```

ou

```
   <xsd:element  name="anything"/>
```

Contenu vide

Un élément qui n'a pas de contenu mais peut porter des attributs

Exemple de modèle de contenu vide :

```
<xsd:element   name="internationalPrice">
  <xsd:complexType>
   <xsd:complexContent>
    <xsd:restriction  base="xsd:anyType">
     <!−− no element declared −−>
      <xsd:attribute  name="currency"  type="xsd:string"/>
     <xsd:attribute  name="value"     type="xsd:decimal"/>
    </xsd:restriction>
```

```
   </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

Exemple d'instance :

```
<internationalPrice  currency="EUR"  value="423.46"/>
```

Contenu nul

Un élément qui peut être présent dans l'instance sans contenu, mais qui peut aussi avoir un contenu.

Permet la distinction entre

- élément absent
- élément présent sans contenu

Exemple de déclaration :

```
<xsd:element  name="shipDate"  type="xsd:date"  nillable="true"/>
```

Exemple d'instance :

```
<shipDate  xsi:nil="true"></shipDate>
```

Séquences, ensembles et choix

Trois éléments pour exprimer les modèles de contenu :

- Séquence (`sequence`) : les éléments déclarés par les fils doivent apparaître dans l'instance dans le même ordre que dans le schéma
- Ensemble sans ordre (`all`) : les éléments déclarés par les fils peuvent apparaître au plus une fois dans l'instance, dans un ordre quelconque
  `all` doit être le seul fils d'un modèle de contenu
- Choix (`choice`) : un seul élément déclaré par un des fils peut apparaître dans l'instance

Un élément offre une facilité d'écriture :

- Groupe (`group`) : équivalent des entités paramètres de XML (interdit dans `all`)

Les attributs `minOccurs` et `maxOccurs` s'appliquent aux éléments `sequence`, `all`, `choice` et `all`

Exemple de séquence, choix

```
<xsd:complexType   name="PurchaseOrderType">
  <xsd:sequence>
   <xsd:choice>
    <xsd:group    ref="shipAndBill"/>
     <xsd:element  name="singleUSAddress"  type="USAddress"/>
   </xsd:choice>
   <xsd:element  ref="comment"  minOccurs="0"/>
   <xsd:element  name="items"   type="Items"/>
  </xsd:sequence>
  <xsd:attribute  name="orderDate"  type="xsd:date"/>
</xsd:complexType>

<xsd:group  name="shipAndBill">
   <xsd:sequence>
      <xsd:element  name="shipTo"  type="USAddress"/>
      <xsd:element  name="billTo"  type="USAddress"/>
```

```
    </xsd:sequence>
</xsd:group>
```

Exemple `all`

```
<xsd:complexType   name="PurchaseOrderType">
   <xsd:all>
      <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:all>
    <xsd:attribute  name="orderDate"  type="xsd:date"/>
</xsd:complexType>
```

Extension de types complexes

Construction d'un nouveau type complexe en ajoutant des composants à un type complexe existant

Analogue à l'extension de type simple (voir `internationalPrice`)

Exemple de dérivation d'un type complexe à partir d'un autre type complexe :

```
  <complexType  name="Address">
   <sequence>
    <element name="name"    type="string"/>
    <element  name="street"  type="string"/>
    <element name="city"    type="string"/>
   </sequence>
  </complexType>

  <complexType  name="UKAddress">
   <complexContent>
    <extension  base="ipo:Address">
     <sequence>
       <element  name="postcode"  type="ipo:UKPostcode"/>
```

```
    </sequence>
     <attribute name="exportCode" type="positiveInteger" fixed="1"/>
   </extension>
  </complexContent>
 </complexType>
```

Équivalent à :

```
 <complexType name="UKAddress">
  <sequence>
   <element name="name"    type="string"/>
   <element name="street" type="string"/>
   <element name="city"    type="string"/>
   <element name="postcode" type="ipo:UKPostcode"/>
  </sequence>
  <attribute name="exportCode" type="positiveInteger" fixed="1"/>
 </complexType>
```

Utilisation des types dérivés

Dans une instance, on peut utiliser un élément du type dérivé là où un élément du type de base est autorisé

Le type dérivé est indiqué dans l'instance par l'attribut `xsi:type`

Exemple d'instance :

```
<shipTo exportCode="1" xsi:type="ipo:UKAddress">
```

au lieu de

```
<shipTo>
```

Restriction de types complexes

Construction d'un nouveau type complexe en supprimant des composants à un type complexe existant

Comparable à la restriction de types simples : les valeurs du nouveau type constituent un sous−ensemble des valeurs du type de base

- ajout d'une valeur par défaut
- ajout d'une valeur fixe
- spécification du `type` quand aucun type n'était spécifié
- réduction du nombre d'occurences (attributs `minOccurs` et `maxOccurs`)

La restriction doit répéter explicitement tous les composants du type de base qui sont conservés

Exemple de restriction

Contrainte supplémentaire : au moins un élément `item` dans l'élément `Items`

```
<complexType  name="ConfirmedItems">
 <complexContent>
  <restriction base="ipo:Items">
   <sequence>

    <!−− item element is different than in Items −−>
     <element name="item" minOccurs="1" maxOccurs="unbounded">

      <!−− remainder of definition is same as Items −−>
      <complexType>
       <sequence>
```

```
        <element name="productName" type="string"/>
       ...
        <element name="shipDate" type="date" minOccurs="0"/>
      </sequence>
       <attribute name="partNum" type="ipo:SKU" use="required"/>
     </complexType>
    </element>

   </sequence>
  </restriction>
 </complexContent>
</complexType>
```

Contrôle de la création de types dérivés (1)

On peut interdire l'extension et/ou la restriction d'un type complexe lors de sa définition

Attribut `final` (valeurs : `extension`, `restriction`, `#all`)

Exemple : interdire toute restriction du type `Address` (mais pas l'extension)

```
<complexType  name="Address"  final="restriction">
 <sequence>
  <element name="name"    type="string"/>
  <element name="street"  type="string"/>
  <element name="city"    type="string"/>
 </sequence>
</complexType>
```

Contrôle de la création de types dérivés (2)

On peut interdire la dérivation d'un type simple facette par facette, lors de sa définition

Attribut `fixed` dans la définition de la facette concernée

Exemple : interdire toute modification de la longueur du type `Postcode` (mais pas des autres facettes)

```
<simpleType  name="Postcode">
  <restriction base="string">
    <length value="7" fixed="true"/>
  </restriction>
</simpleType>
```

Annotations

L'élément `annotation` permet d'annoter les schémas XML pour le lecteur humain et les applications

L'élément `annotation` a deux sous−éléments :

- `documentation` : destiné au lecteur humain
- `appInfo` : fournit de l'information aux applications et feuilles de style

L'élément `annotation` peut apparaître au début de la plupart des constructions

Schémas et instances

Une instance ne référence pas forcément un schéma

Une instance peut référencer son schéma par l'attribut `xsi:schemaLocation`

Un validateur est libre d'utiliser cette information on non

Exemple :

```
<purchaseReport
    xmlns="http://www.example.com/Report"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
     xsi:schemaLocation="http://www.example.com/Report
                         http://www.example.com/Report.xsd"
   period="P3M"  periodEnding="1999−12−31">
 <!−− etc. −−>
</purchaseReport>
```

Schéma XML et Infoset

- la validité par rapport à une DTD et la *well−formedness* sont définies pour des séquences de caractères
- la validité par rapport à un schéma XML est définie pour un *Infoset*

Infoset

- modèle de données abstrait pour les documents XML
- définit un petit nombre de composants : élément, attribut, etc.
- chaque composant a des propriétés : nom, enfants, etc.

Validation

La validation des instances n'est pas obligatoire

Deux fonctions d'un validateur :

1. Vérification de la conformité par rapport aux contraintes exprimées dans le schéma
2. Ajout d'informations à l'infoset : types, valeurs par défaut, etc.

Autres mécanismes

- Espaces de noms
- Contrôle d'unicité : éléments `unique`, `key`, `keyref`
- Importations avec dérivations possibles de définitions globales venant de schémas d'espaces de noms différents