Développer des projets en XML

Les modèles formels, les schémas des documents XML, les DTD et XML Schema

Déclaration du type de document

- Il s'agit de la **déclaration** (<!DOCTYPE…>) du type de document et pas la **définition (ensemble des instructions qui définissent le document)**
- Forme <!DOCTYPE ...> et peut contenir:
  - nom du document idem nom de l'élément racine
  - référence vers un DTD (définition) externe ou DTD local

Déclaration du type de document

Déclaration du type de document

- Lorsqu 'un document est accompagné d 'une déclaration de type, le parseur peut vérifier (pour un parseur validant) le respect de cette « grammaire » par le contenu du document
- Si, tel est le cas, le document est validé
- Dans les autres cas, le parseur signale les erreurs.

Déclaration du type de document

- Syntaxe pour déclaration et définition :

  <!DOCTYPE nom_document [

  (définition des éléments, attributs d'éléments et entité)]>

- La définition **des éléments, d'attributs d'éléments et des entités** consiste en le DTD et répond à une syntaxe propre.

Exemple de fichier XML

Ex11inttech.xml avec DTD local:

```
<?xml version="1.0"?>
<!DOCTYPE texte [
<!ELEMENT texte (#PCDATA)>
]>
<texte>Ceci est le texte.</texte>
```

Dans cette exemple, le DTD est à l 'intérieur du document, ce qui ne permet pas sa réutilisation pour d'autres documents.

DTD externe au document

- Un DTD externe à un document XML est réutilisable par d 'autre document.
- Dans ce cas, le document n 'est plus « standalone » (valeur par défaut, standalone="no")

DTD externe au document

- Référence se fait par :
  - identificateur Public

    <!DOCTYPE typedoc PUBLIC "ident_public" "url_dtd">
  - identificateur System

    <!DOCTYPE typedoc SYSTEM "url_dtd">

    Pour SYSTEM, il s 'agit d 'une URL

Identifiant PUBLIC pour DTD externe

- L 'identifiant public est utilisé par le parseur pour obtenir le fichier DTD à partir d 'un repository interne ou externe, sur base d 'un mécanisme propre au parseur
- L 'identifiant PUBLIC est composé de 4 parties séparées par //:
  - - ou + s'il s 'agit d'organismes reconnus
  - identificateur de l'organisation
  - mot clé permettant de supposer le format
  - la langue
  - exemple :

    -//expertit//TEXT listetel//FR
  - le mécanisme pour retrouver le fichier ne fait pas partie du standard et l 'URL est utilisée si le parseur ne peut retrouver le document

Exemple de DTD externe

Ex11inttech.xml avec DTD externe:

```
<?xml version="1.0"?>
<!DOCTYPE texte SYSTEM "ex11inttech.dtd">
<texte>Ceci est le texte.</texte>
```

contenu du fichier "ex11inttech.dtd":

```
<!ELEMENT texte (#PCDATA)>
```

Déclaration du document

- attache un DTD à un document
  - `<!DOCTYPE address-book SYSTEM "address-book.dtd">`
- doit contenir le nom de l'élément racine
  - le DTD ne déclare pas l'élément racine
- ne pas confondre avec
  - le Document Type Definition
  - la déclaration XML

Relation entre document et DTD avec déclaration mixte.

- la relation est très forte puisque le DTD décrit la structure du document
  -
```
<!DOCTYPE address SYSTEM "address-content.dtd" [
<!ELEMENT address (street,region?,postal-
code,locality,country)>
<!ATTLIST address  preferred (true | false)
"false">]>
<address>
   <street>34 Fountain Square Plaza</street>
   <region>OH</region>
   <postal-code>45202</postal-code>
   <locality>Cincinnati</locality>
   <country>US</country>
</address>
```

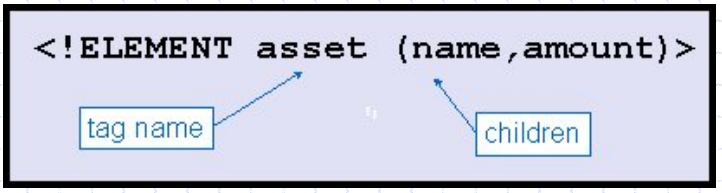
Avantages des DTDs

- l'outil XML valide la structure
- l'application connaît la structure
  - pour permettre à l'éditeur de guider l'utilisateur
- sépare les données de l'indentation
- des valeurs par défaut permettent de réduire la taille du document

Définition des éléments

- Forme générale:

  <!ELEMENT nom_element (spécification_du_contenu)>



Définition des éléments

- Type de définition d 'élément :
  Elément avec un contenu texte
  <!ELEMENT asset (#PCDATA)>
  Elément avec éléments enfants spécifiés dans l 'ordre
  <!ELEMENT asset (do,re,mi,fa,so)>
  Elément avec éléments enfants spécifiés dans n 'importe quel ordre
  <!ELEMENT asset (do|re|mi|fa|so)*>
  Pas d 'élément enfant et pas de contenu
  <!ELEMENT asset EMPTY>

Définition des éléments

- Contenu peut être :
  - éléments: (nom, tel, fax)
  - contenu mixte avec ou sans éléments:
    (#PCDATA) ou (#PCDATA | date)*
  - mots réservé EMPTY ou ANY :
    <!ELEMENT client ANY>

Définition des éléments

- Contenu mixte et éléments sont contenus entre parenthèses
- #PCDATA
  - signifie Parsed Character Data
  - indique que l'élément est du texte ou autres caractères
  - ce texte sera analysé par le parseur
  - il ne peut pas contenir des marqueurs ou les caractères ", & et [ ].
  - Pour les caractères <, > et &, il faut utiliser les entités < > et &

Entités prédéfinies

- < "<"
- & "&"
- > ">"
- ' "'"
- " "\""
- î "î"
  - `<name>Beno&#238;t Marchal</name>`

Définition des éléments

- La ',' est un séparateur d'une série séquentielle d'éléments, séquence à respecter
- Le '|' est un séparateur d'une série alternative d'éléments, pas d'ordre à respecter
- Le '?' indique qu'un élément peut apparaître 0 ou une fois
- Le '*' indique qu'un élément peut apparaître 0 ou n fois
- Le '+' indique qu'un élément peut apparaître 1 ou n fois

Définition des éléments

Elément enfant doit apparaître exactement une fois
<!ELEMENT asset (amount)>

Elément enfant peut apparaître une fois
<!ELEMENT asset (amount?)>

Elément enfant peut apparaître une fois ou plus
<!ELEMENT asset (amount+)>

Elément enfant peut apparaître zéro fois ou plus
<!ELEMENT asset (amount*)>

Définition des éléments

- Exemples:

```
<!ELEMENT client( nom, date_contact+,
  remarque?)>
<!-- client composé de 1 seul nom et d'au
  moins une date_contact, 0 ou 1
  remarque dans l'ordre spécifié -->
<!ELEMENT nom(#PCDATA)>
<!-- nom est composé de texte -->
```


Définition des éléments

- Exemples:

```
<!ELEMENT date_contact(#PCDATA | date)>
<!-- date_contact peut être du texte ou une
  date -->
<!ELEMENT remarque ANY>
<!-- remarque peut contenir n'importe quel
  élément déclaré et dans n'importe quel ordre.
  -->
```

Définition des éléments

- Exemples:

```
<!ELEMENT client(nom, tel*, compagnie?, (contact |
  info_personnel)*)>
<!-- client est composé de nom,tel (0 ou n),compagnie
  (0 ou 1) dans l'ordre -->
<!-- contact et info_personnel, spécifié dans n'importe
  quel ordre et optionnel -->
<!ELEMENT nom(prenom, nom_famille, surnom*)>
<!-- nom est composé de prenom, nom_famille et n
  surnoms optionels à spécifier dans l'ordre -->
```

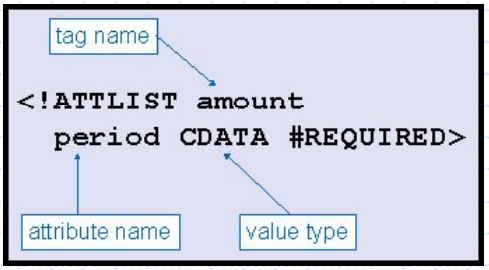

Section CDATA

- Dans un élément contenant du #PCDATA, il est possible d 'insérer une section CDATA
- Une section CDATA ne sera pas analysée par le parser et sera considérée comme du texte plein
- Exemple:

```
...
Pour obtenir les résultats voulus, utilisez la requête:
<![CDATA[Select * from facture where montant < 10000
  and montant > 5000 and libelle = ` client ']]>
...
```

Définition d'attributs d'éléments

- Forme générale de la définition d'un attribut :
  <!ATTLIST nom_element
  nom_attribut type_attribut valeurdéfaut>



Attribut

- la déclaration d'un attribut précise son type
  - `<!ATTLIST tel preferred (true | false) "false">`
- il est permit de grouper les déclarations
  - `<!ATTLIST email href CDATA #REQUIRED`
    `preferred (true | false) "false">`
- les déclarations peuvent apparaître n'importe où dans le DTD

Types des attributs

- CDATA : texte
- ID : identifiant
- IDREF : référence vers un identifiant
- IDREFS : références vers plusieurs identifiants
- ENTITY : entités
- ENTITIES : plusieurs entités
- NMTOKEN : texte suivant les règles des noms
- NMTOKENS : plusieurs NMTOKEN
- type énuméré : liste de valeurs

Valeur par défaut

- #REQUIRED : obligatoire
- #IMPLIED : l'application peut fournir une valeur par défaut
- #FIXED : la valeur ne peut changer
- littéral : la valeur par défaut

Définition d'attributs d'éléments

- Attribut CDATA le plus fréquent et indique qu'il s'agit de texte qui ne sera pas analysé par le parseur (

```
<!ATTLIST acteur
role CDATA #implied>
<!-- #implied ne spécifie aucune valeur et l'attribut est optionnel -->
```

- L'attribut peut contenir n'importe quel texte à l'exception de balises.
- Exemple:

```
<acteur role="hamlet">To be or not to be...</actor>
<acteur>Atmosphère, atmosphère...</acteur>
```


Définition d'attributs d'éléments

- Attribut ID doit être unique. Aucun autre ID du même nom ne peut être spécifié dans le document

```
<!ATTLIST nom
numero ID #required>
<!-- #required spécifie que l'attribut doit exister et une valeur doit être donnée -->
```

- Exemple:

```
<nom numero="p123">Dupont</nom>
<nom>Durand</nom>
```

!!!!! erreur, l'attribut manque

Définition d'attributs d'éléments

- Liste énumérative des valeurs possibles avec valeur par défaut

  <!ATTLIST contact type_contact (inconnu| professionnel| familial| social) "inconnu">

  <!-- la valeur par défaur doit être entre " -->
- Exemple:

  <contact type_contact="familial">Charles-Henri</contact>

  <contact>Durand</contact> (valeur par défaut)

Définition d'attributs d'éléments

- Exemple d 'attribut avec valeur alternative et obligatoire:

<!ATTLIST item custom.color (yes|no) #REQUIRED>

- Exemple d 'attribut avec valeur alternative et valeur par défaut:

<!ATTLIST item custom.color (yes|no) "yes">

Définition d'attributs d'éléments

- Exemple d 'attribut avec valeur fixe :

<!ATTLIST item.detail lang CDATA #FIXED "EN" >

- Type d 'attribut ID :

<!ATTLIST detail ID ID #REQUIRED>

- Un attribut lien :
  <!ATTLIST item href CDATA #REQUIRED>
- Attributs multiples:
  <!ATTLIST item ID ID #REQUIRED color CDATA #IMPLIED type (a|b) 'b'>

Traitement spécial des espaces

- Le processeur XML transmet tous les caractères d'espacement à l'agent client (navigateur). Le traitement des espaces dépend de l'agent.
- XML prévoit un attribut pré-défini (built-in) et utilisé comme suit:

```
<!ATTLIST document
xml:space(default|preserve) "preserve">
<!-- les espacements seront préservés, si l'attribut est
défini pour l'élément racine, tous les autres éléments
en hériteront -->
```

Définition d'entités

- XML prévoit quelques entités pré-définies (&, <, >, ', ").
- Le concepteur de DTD peut prévoir la définition d 'entités supplémentaires.
- Une entité est une unité de stockage contenant soit des données textuelles, soit des données binaires.
- Une entité est identifiée par un nom et possède un contenu.
- Après la définition d'une entité, le nom de celle-ci peut être utilisé dans le document pour récupérer son contenu.
- Il existe deux types d'entités : générales et paramétriques.

Définition d'entités générales

- Définition : <!ENTITY nom_entité…>
- Utilisation : &nom_entité;
- Exemple:

```
...
<!ENTITY avertissement "Toute copie est
  illégale">
...
<texte>&avertissement;</texte>
...
```

Définition d'entités générales externes

- Une entité peut également faire référence à un fichier externe
- Exemple :

```
...
<!ENTITY fichier_ext SYSTEM "fichier.txt">
...
<texte>&fichier_ext;</texte>
...
```


Déclaration des entités générales

- Collection de définitions d'entités générales reprise dans un fichier externe qui sera inclus dans un DTD via une entité paramétrique

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ENTITY be "Belgium">
<!ENTITY ch "Switzerland">
<!ENTITY de "Germany">
<!ENTITY fr "France">
<!ENTITY it "Italy">
<!ENTITY jp "Japan">
<!ENTITY uk "United Kingdom">
<!ENTITY us "United States">
```

Entités paramétriques

- Entité uniquement utilisable pour la DTD dans un but de simplification :

```
<!ENTITY % lang-codes "EN|GE|JP|FR">
...
  <!ATTLIST item lang (%lang-codes;) "EN">
```


Entité paramétrique

- reconnaissable au "%"

```
<!ENTITY % boolean "true | false">
<!ELEMENT tel (#PCDATA)>
<!ATTLIST tel preferred (%boolean;) 'true'>
```

- dans le sous-ensemble interne, elles ne peuvent apparaître que là où le balisage est autorisé

Entité générale et paramétrique externe

- entité externe est dans un fichier séparé
  - `<!ENTITY fr SYSTEM "france.ent">`
  - `<!ENTITY % countries SYSTEM "countries.ent">`
    `%countries;`
- géré comme pour le DTD
  - qui est une entité
- un gestionnaire contrôle l'accès aux entités
  - on peut stocker les entités dans une base de données, etc.

Schémas XML

- DTD limité face à la modélisation moderne
  - objet, héritage
- DTD utilise le langage EBNF (extended backus naur format) et non un langage XML:
  - Impossible d'utiliser un parseur XML pour découvrir dynamiquement le schéma ou générer dynamiquement un schéma.
  - Pas d'extensibilité facile.
  - moins aisé à lire par un humain.
- Pas de support réel des NameSpaces

Schémas XML

- DTD ne reconnaît que le type texte, limite pour des applications orientées "données".
- Un seul DTD par document, impossible de créer des schémas modulaires ou d'intégrer des DTD de sources différentes.
- Différentes technologies remplaçantes ont été proposées, parmi lesquelles, les plus importantes :
  - xdr – XML Data Reduced de Microsoft (xml-data), supporté par MSXML2, IE5, et les serveurs BIZZTalk (fichier avec extension xdr)
  - XML Schema (xsd) (normalisé par le W3C depuis le 2/05/2001)

Concepts de schéma revisités

- Un Schéma de manière générique :
  - technique de représentation des organisations ou structures d'une base de données
  - structure conçue à partir d'un modèle
  - Nommé les éléments supportés par le modèle
  - Description des relations existantes entre les éléments
  - Contraintes appliquées sur les éléments (type,valeurs autorisées, formatage)

XML Schema

- Technologie proposée par le W3C pour remplacer le langage EBNF et les DTD.
- Statut de XML Schema : W3C Recommendation, 2 May 2001
- Technologies plus ambitieuse d'un point de vue modélisation, types de données.
- Basé sur le language XML Schema, langage XML
- Déjà supporté par quelques outils :
  - XML Authority
  - XML Spy

XML Schema

- Language XML Schema est composé de 2 parties :
  - XML Schema Part 1: Structures http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/
    - Definition language, permet de définir les éléments de structures du schéma et les contraintes de contenu des documents XML avec le support de XML NameSpace
  - XML Schema Part 2: Datatypes http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/
    - Datatype language, permet de définir les type des éléments et des attributs

XML-Schema

- Autres technologies pour déclaration des schemas :
  - DTD écrit en EBNF (extended backus naur format)
  - xdr – XML Data Reduced de Microsoft (xml-data), supporté par MSXML2, IE5, et les serveurs BIZZTalk (fichier avec extension xdr)

Concepts de base de XML Schema

- Un schéma définit une classe de document XML
- Un document XML conforme à ce schéma est une instance de document de cette classe
- Le terme document doit être pris dans un sens large et pas uniquement l'entité physique
- Un document XML est une arborescence composée d'éléments et de sous-éléments. Les éléments peuvent également contenir des attributs.

Concepts de base de XML Schema

- Un élément contenant des éléments enfants et/ou des attributs est dit de "type complexe"
- Un élément ne contenant que du texte (numérique, date, texte ou autres) et pas d'attributs est dit de "type simple"
- Les attributs sont toujours de type "simple".
- Le schéma peut définir des types (définis par l'auteur du schéma, complexes et/ou simples) et utiliser des types pré-définis dans le langage XML Schema.
- Présentation inspirées du document "XML Schema Primer 0" du W3C.

Exemple de schéma en XML Schema

```
<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"/>
```

Exemple de schéma en XML Schema

```
<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"/>
```


Exemple de schéma en XML Schema

```
<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"/>
```

Exemple de schéma en XML Schema

```
<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>
```


Exemple de schéma en XML Schema

```
<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>
```

Éléments de base langage XML Schema

- schema : élément racine du schéma
- element : déclaration d'élément
- attribute : déclaration d'attribut
- complexType : définition d'un type complexe
- simpleType : définition d'un type simple

L'élément schema du schéma

- Préfixe du Namespace :
  - xsd:
- URI d'identification :
  - http://www.w3.org/2001/XMLSchema

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

Déclaration d'un élément

- "element" supporte, entre autres, les attributs suivant:
  - default : contenu textuel par défaut
  - fixed : contenu textuel fixé
  - maxOccurs, minOccurs : contraintes d'occurrence
  - name : nom de l'élément
  - ref : nom d'un élément global
  - type : type de l'élément
- Un élément non-typé (sans attribut type) contient le modèle de son contenu.

Déclaration d'un attribut

- "attribute" supporte, entre autres, les attributs suivant:
  - default : contenu textuel par défaut
  - fixed : contenu textuel fixé
  - maxOccurs, minOccurs : contraintes d'occurrence
  - name : nom de l'élément
  - ref : nom d'un élément global
  - type : type de l'élément
  - use : type d'utilisation
    - optional (valeur par défaut), prohibited, required
- Les attributs doivent toujours être déclarés à la fin d'un complexType.

Eléments et attributs globaux

- Globaux : éléments/attributs déclarés comme enfants de l'élément schema du schéma.
- Utilisable avec l'attribut "ref" dans la déclaration d'un élément/attribut
- Permet de réutiliser la déclaration de l'élément/attribut global à plusieurs endroits dans le schéma
- Contraintes d'utilisation:
  - Par de contraintes d'occurrence dans la déclaration globale
  - Pas de "ref" dans la déclaration globale

Déclaration d'élément par référence

```
<xsd:element name="comment" type="xsd:string"/>
...
<xsd:element ref="comment" minOccurs="0"/>
```

- Un élément peut être déclaré en référençant un élément déclaré autre part dans le schéma.
- En général, les références se font vers des éléments dits "globaux", càd via des xsd:element enfants de xsd:schema.
- Permet de centraliser la déclaration d'éléments utilisés à plusieurs reprises dans le schéma.

Contraintes d'occurrence

- Pour les éléments :
  - Se fait via attribut minOccurs et maxOccurs
  - Par défaut, minOccurs et maxOccurs valent 1
  - minOccurs="0", l'élément est optionnel
  - minOccurs <= maxOccurs
  - Valeur : décimal ou "unbounded"
  - minOccurs et maxOccurs = 0, l'élément ne peut pas être présent.
- Pour les attributs :
  - Un attribut apparait une seule fois ou pas du tout
  - Attribut xsd:use permet d'indiquer les occurrences de l'attribut : "required", "optional" (défaut), "prohibited"

Valeurs par défaut

- Valeur par défaut d'élément ou d'attribut est défini à l'aide de l'attribut "default"
- Pour un attribut, la valeur par défaut n'a de sens que si l'attribut est use="optional"
- Pour un attribut, le parseur xml donne la valeur défaut si l'attribut n'est pas présent
- Pour un élément, la parseur xml ne donne la valeur par défaut que si l'élément apparaît dans le document vide, sans contenu. Si l'élément n'est pas présent, le parseur xml ne donne rien, ni élément, ni valeur par défaut.

Valeurs fixées

- L'attribut "fixed" peut être suivi d'une valeur.
- Cet attribut peut être utilisé pour la déclaration d'un élément et d'un attribut.
- Si une valeur est spécifiée (élément et attribut), cette valeur doit être identique à celle indiquée dans le schéma.
- Si la valeur n'est pas spécifiée, le parseur donne la valeur du schéma.

Création et utilisation de type

- La définition de type se fait à l'aide de :
  - xsd:complexType
  - xsd:simpleType
- L'utilisation des types définis se fait lors de la déclaration d'un élément ou d'un attribut (avec xsd:element ou xsd:attribute) à l'aide de l'attribut "type".

Création et utilisation de type

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
</xsd:complexType>
...
<xsd:element name="purchaseOrder"
    type="PurchaseOrderType"/>
```


Définition de type complexe

- Définition de type complexe, xsd:complexType
- Attributs:
  - name : nom du type
- Peut contenir les éléments "simpleContent" ou "complexContent" qui eux-mêmes contiennent la définition du contenu.
- Contient une série de déclarations d'éléments et/ou d'attributs, xsd:element et xsd:attribute
- Cette série peut être contrainte par le groupage d'éléments:
  - xsd:sequence, xsd:choice, xsd:all

Définition de type complexe

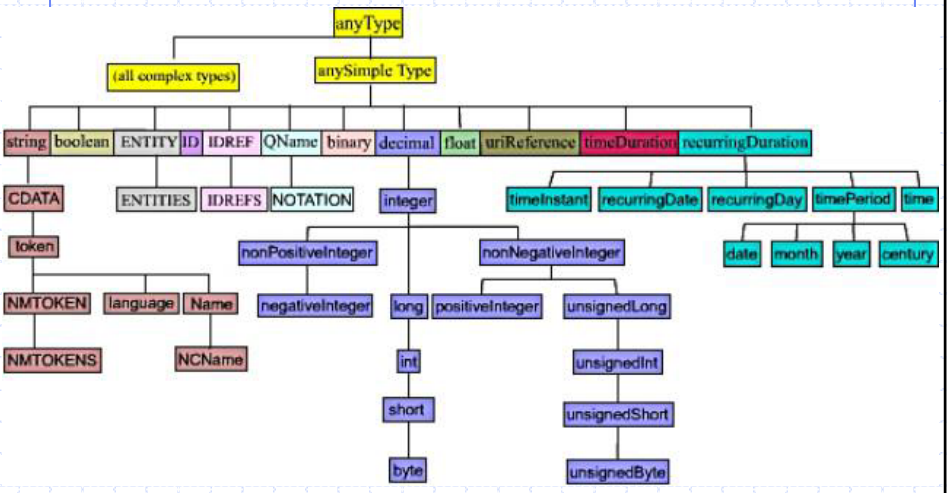
```
<xsd:complexType name="USAddress">
 <xsd:sequence>
  <xsd:element name="name"   type="xsd:string"/>
  <xsd:element name="street" type="xsd:string"/>
  <xsd:element name="city"   type="xsd:string"/>
  <xsd:element name="state"  type="xsd:string"/>
  <xsd:element name="zip"    type="xsd:decimal"/>
 </xsd:sequence>
 <xsd:attribute name="country" type="xsd:NMTOKEN"
   fixed="US"/>
</xsd:complexType>
```


Utilisation de type complexe

```
<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>
```

- Un élément d'un type complexe peut être déclaré comme étant d'un autre type complexe.

Types simples pré-définis dans XML Schema



Définition de type simple

- Définition d'un type simple: xsd:simpleType
- Attributs (entre autres) :
  - name : nom du type
- Un élément attribut peut contenir des éléments :
  - restriction, list, union

Définition de type simple dérivé d'autre type simple pré-défini ou non

- Utilisation de xsd:simpleType et de xsd:restriction
- L'élément "restriction" contient des "facets" qui restreignent le type simple

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

- Les facets sont des éléments de restriction en fonction du type simple dérivé.

Type dérivé à l'aide de "facet" expression régulière

- Utilisation du facet "pattern", supportés par tous les types simples pré-définis et dont la valeur est spécifié à l'aide d'expression régulière :

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

  - Format : 3 chiffres suivis d'un tiret suivi de 2 caractères alphabétiques.

Exemples de type dérivé

- "Facet" énumération permet de définir une liste de valeurs autorisées pour un type d'élément/attribut et peut s'appliquer à tous les types pré-définis sauf booléen :

```
<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
```


Les types "liste"

- A partir de types simples dits "atomiques" (càd indivisibles, au risque de perdre toute signification), il est possible de définir des listes à l'aide de xsd:list:

```
<xsd:simpleType name="listOfMyIntType">
  <xsd:list itemType="myInteger"/>
</xsd:simpleType>
...
<listOfMyInt>20003 15037 95977 95945</listOfMyInt>
```

Les types "liste"

- Par un type dérivé et des "facets", il est possible d'appliquer des contraintes sur les listes telles que "length", "maxLength", "minLength", "enumeration":

```
<xsd:simpleType name="USStateList">
 <xsd:list itemType="USState"/>
</xsd:simpleType>
...
<xsd:simpleType name="SixUSStates">
 <xsd:restriction base="USStateList">
  <xsd:length value="6"/>
 </xsd:restriction>
</xsd:simpleType>
...
<sixStates>PA NY CA NY LA AK</sixStates>
```


Type anonyme

- Type anonyme = type non nommé
- Utilise pour les types non réutilisés et uniquement pour le modèle de contenu d'un élément
- Pour "complexType" et "simpleType", pas d'attributs "name"
- Pour "element", pas d'attribut "type", dans ce cas, la définition du type se fait comme contenu de "xsd:element".

Type anonyme

```
<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>
```


complexType à contenu simple

- Seul les éléments de type complexes peuvent contenir des attributs
- Pour définir un type d'élément à contenu simple (dérivant de type simple) avec attributs, il faut utiliser "complexType" avec "simpleContent".

complexType à contenu 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>
...
<internationalPrice currency="EUR">423.46</internationalPrice>
```


Contenu complexe et mixte

- Définition d'un type complexe à contenu mixte à l'aide de l'attribut :
  - mixed="true"
- Contenu mixte signifie que l'élément peut contenir à la fois des éléments enfants et du contenu textuel

```
<xsd:complexType mixed="true">
```

Type d'éléments vide avec attributs

- Ce type définit un modèle de contenu vide avec des attributs

```
<xsd:element name="internationalPrice">
 <xsd:complexType>
  <xsd:attribute name="currency" type="xsd:string"/>
  <xsd:attribute name="value"    type="xsd:decimal"/>
 </xsd:complexType>
</xsd:element>
...
<internationalPrice currency="EUR" value="423.46"/>
```


Type "xsd:anyType"

- En fait cette définition est une version abrégée de :

```
<xsd:element name="internationalPrice">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:restriction base="xsd:anyType">
    <xsd:attribute name="currency" type="xsd:string"/>
    <xsd:attribute name="value"    type="xsd:decimal"/>
   </xsd:restriction>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
```

- N'importe que contenu, sans définir d'élément dans le modèle

Type "xsd:anyType"

- xsd:anyType déclare un contenu de n'importe quel type
- Il est le type par défaut lorsque aucun type n'est spécifié
- Permet de contenir du contenu simple (texte, numérique) ou complexe mixte (éléments, texte, numérique)
- anyType est le type dont dérive tous les autres types

Remarques et commentaires dans les schémas

- "annotation" permet d'introduire du commentaire dans un shéma, contient:
  - "documentation" : commentaires de l'auteur
  - "appInfo" : infos destinées aux applications traitant le schéma
- "annotation" peut être enfant de "schema", "element", "simpleType", "complexType", "attribute"

```
<xsd:simpleType name="string" id="string">
  <xsd:annotation>
   <xsd:appinfo>
     <hfp:hasFacet name="length"/>
     <hfp:hasFacet name="minLength"/>
     <hfp:hasFacet name="maxLength"/>
```

Modèles de contenu

- Un type complexe peut contenir une combinaison d'éléments enfants, dite "groupe" d'enfant.
- La description de cette combinaison correspond au modèle de contenu défini par ce type.
- Cette combinaison est décrite à l'aide de :
  - "sequence", "all", "choice"
  - qui créent un groupe

Modèles de contenu

```
<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>
```

Modèles de contenu

- Il est possible de créer des groupes, de les nommer et de les réutiliser dans d'autres groupes

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


Modèles de contenu : sequence

- Indique que les éléments du groupe doivent être spécifiés dans cet ordre.
- La cardinalité des éléments est décrite à l'aide des attributs minOccurs et maxOccurs
- Si minOccurs = 0 -> élément optionnel
- Si minOccurs = 1 -> élément obligatoire
- Défaut de minOccurs et maxOccurs = 1

Modèles de contenu : choice

- Indique qu'un seul des éléments du groupe peut être spécifié

```
<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>
```


Modèles de contenu : all

- Indique que les éléments du groupe peuvent tous être spécifiés, 0 ou 1 fois max, dans n'importe quel ordre
- "all" doit être le seul groupe au top-niveau du modèle de contenu (càd enfant direct de complexType)
- "all" ne peut pas se trouver lui-même dans un groupe
- Ne peut contenir d'élément "group"

Modèles de contenu : 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>
```


Cardinalité des groupes dans le modèle

- Grâce aux attributs minOccurs et maxOccurs, il est possible de spécifier la cardinalité d'un groupe

Groupes d'attributs

- "attributeGroup" permet de définir et de nommer un groupe d'attributs
- Ce groupe peut être réutilisé dans une définition de type complexe avec l'élément "attributeGroup" et l'attribut "ref"
- Un groupe d'attributs peut contenir un autre groupe d'attributs

Groupes d'attributs

```
<xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
 <xsd:complexType>
...
  <xsd:attributeGroup ref="ItemDelivery"/>
 </xsd:complexType>
</xsd:element>

...
<xsd:attributeGroup name="ItemDelivery">
  <xsd:attribute name="partNum"  type="SKU" use="required"/>
  <xsd:attribute name="weightKg" type="xsd:decimal"/>
</xsd:attributeGroup>
...
```

Localisation des schémas et namespaces

- Un schéma est une collection de définition de types et de déclaration d'éléments.
- L'attribut "targetNamespace" permet d'associer les définitions/déclarations du schéma à un NameSpace
- Les attributs elementFormDefault et attributeFormDefault indiquent si tous les éléments/attributs locaux (càd non top-level) doivent être qualifiés.
- L'attribut "form" peut être utilisé au niveau d'une déclaration pour indiquer si un élément/attribut doit être qualifié

Localisation des schémas et namespaces

- Exemple de schéma avec targetNamespace et éléments/attributs locaux non-qualifiés

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:po="http://www.example.com/PO1"
        targetNamespace="http://www.example.com/PO1"
        elementFormDefault="unqualified"
        attributeFormDefault="unqualified">
 <element name="purchaseOrder"
   type="po:PurchaseOrderType"/>
 <element name="comment"       type="string"/>
```

Localisation des schémas et namespaces

- Suite exemple :

```
<complexType name="PurchaseOrderType"> <sequence>
  <element name="shipTo"    type="po:USAddress"/>
  <element name="billTo"    type="po:USAddress"/>
  <element ref="po:comment" minOccurs="0"/>
  <!-- etc. -->  </sequence>
 <!-- etc. --> </complexType>
<complexType name="USAddress">
 <sequence>
  <element name="name"   type="string"/>
  <element name="street" type="string"/>
  <!-- etc. -->
 </sequence> </complexType>
<!-- etc. --></schema>
```


Localisation des schémas et namespaces

- Le processus de validation consiste :
  - à déterminer quels sont les définitions d'éléments et d'attributs qui interviennent dans la validation d'une instance de document (repérer le schéma ou les schémas)
  - à vérifier, sur base de ce(s) schéma(s), l'adéquation du contenu par rapport aux modèles
- Le schéma utilisé pour valider un document XML est repéré à l'aide des NameSpaces et d'attributs spécifiés dans les schémas et le document XML

Localisation des schémas et namespaces

- Rappel sur les namespaces :
  - Pour associer des éléments/attributs à un namespace, on utilise l'attribut "xmlns" suivi d'un préfixe, et suivi de l'URI d'identification
  - On utilise le préfixe pour qualifier un élément et un attribut, ce qui rattache l'élément/attribut à ce namespace.

Localisation des schémas et namespaces

```
<?xml version="1.0"?>
<apo:purchaseOrder xmlns:apo="http://www.example.com/PO1"
                   orderDate="1999-10-20">
   <shipTo country="US">
      <name>Alice Smith</name>
      <street>123 Maple Street</street>
      <!-- etc. -->
   </shipTo>
   <billTo country="US">
      <name>Robert Smith</name>
      <street>8 Oak Avenue</street>
      <!-- etc. -->
   </billTo>
   <apo:comment>Hurry, my lawn is going wild!</apo:comment>
   <!-- etc. -->
</apo:purchaseOrder>
```

Eléments locaux à qualifier

- Exemple de schéma où les éléments locaux doivent être qualifiés:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:po="http://www.example.com/PO1"
        targetNamespace="http://www.example.com/PO1"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">
 <element name="purchaseOrder" type="po:PurchaseOrderType"/>
 <element name="comment"       type="string"/>
 <complexType name="PurchaseOrderType">
  <!-- etc. -->
 </complexType>
 <!-- etc. -->
</schema>
```


Eléments locaux à qualifier

- Exemple d'instance de document

```
<?xml version="1.0"?>
<apo:purchaseOrder xmlns:apo="http://www.example.com/PO1"
                   orderDate="1999-10-20">
   <apo:shipTo country="US">
      <apo:name>Alice Smith</apo:name>
      <apo:street>123 Maple Street</apo:street>
      <!-- etc. -->
   </apo:shipTo>
   <apo:billTo country="US">
      <apo:name>Robert Smith</apo:name>
      <apo:street>8 Oak Avenue</apo:street>
      <!-- etc. -->
   </apo:billTo>
   <apo:comment>Hurry, my lawn is going wild!</apo:comment>
   <!-- etc. --></apo:purchaseOrder>
```

Eléments locaux à qualifier

- Exemple d'instance de document avec Namespace par défaut (non-qualifié est associé à ce NS) :

```
<?xml version="1.0"?>
<purchaseOrder xmlns="http://www.example.com/PO1"
    orderDate="1999-10-20">
   <shipTo country="US">
      <name>Alice Smith</name>
      <street>123 Maple Street</street><!-- etc. -->
   </shipTo>
   <billTo country="US">
      <name>Robert Smith</name>
      <street>8 Oak Avenue</street>
      <!-- etc. -->
   </billTo><comment>Hurry, my lawn is going wild!</comment>
   <!-- etc. --></purchaseOrder>
```


Namespace target non-spécifié dans le schéma

- Un schéma peut ne pas déclarer de target namespace
- Dans ce cas, les définitions/déclarations ne sont pas qualifiées.
- Seules les éléments/attributs du langage XML Schema doivent être qualifiée (recommandé pour ne pas confondre éléments de XML Schema avec les éléments du schéma).

Namespace target non-spécifié dans le schéma

- Les déclarations/définitions d'un schéma sans target NS valident les éléments/attributs non-qualifiés d'une instance de document.
- Un schéma sans target namespace est nécessaire pour valider un document XML ne spécifiant pas de Namespace.

Spécification explicite de l'emplacement du schéma

- Une instance de document peut contenir l'attribut :
  - xsi:schemaLocation="NS URL" : la première valeur est l'identifiant du NS, la deuxième est l'URL pour accéder au schéma. Cette ettribut est utile pour les schémas avec target namespace
  - xsi:noNamespaceSchemaLocation="URL" : attribut pour les schémas sans target NS
- Ces attributs sont à titre indicatifs et peuvent ne pas être pris en compte par les parsers selon la recommendation (voir la doc des parser).

XML-Schema

- Avantage schéma :
  - syntaxe XML
  - support type, élimine contrôle type nécessaire dans application
  - modéle de contenu via complexType avec reutilisabilité
  - extensibilité
  - écriture de schéma dynamique