XML Schéma

Références

- Quelques liens utiles
  - http://www.w3schools.com/schema/schema_intro.asp
  - http://gilles.chagnon.free.fr/cours/xml/schema.html#restriction

Insuffisance des DTD

- Limitations
  1. les DTD ne sont pas au format XML
     - Cela signifie qu'il est nécessaire d'utiliser un outil spécial pour "parser" un tel fichier, différent de celui utilisé pour l'édition du fichier XML
  2. les DTD ne supportent pas les "espaces de nom"
     - En pratique, cela implique qu'il n'est pas possible, dans un fichier XML défini par une DTD, d'importer des définitions de balises définies par ailleurs
  3. le "typage" des données est extrêmement limité
     - que du #PCDATA
     - d'où des efforts d'interprétation des données

XML Schéma

Apports des schémas / DTD

- Un grand nombre de types de données de base
  - booléens, entiers, intervalles de temps, etc.
- Il est possible de créer de nouveaux types
  - par ajout de contraintes sur un type existant
  - des types de données utilisateurs qui vous permettent de créer votre propre type de données nommé
- La notion d'héritage
  - Les éléments peuvent hériter du contenu et des attributs d'un autre élément
- Une grande facilité de conception modulaire de schémas

- Exemple : richesse apportée par le schéma
  - Document XML : note.xml

```
<?xml version="1.0" encoding="utf-8"?>
<book isbn="0836217462">
    <title>
        Being a Dog Is a Full-Time Job
    </title>
    <author>Charles M. Schulz</author>
    <character>
        <name>Snoopy</name>
        <friend-of>Peppermint
            Patty
        </friend-of>
        <since>1950-10-04</since>
        <qualification>
            extroverted beagle
        </qualification>
    </character>
    <character>
        <name>Peppermint
            Patty
        </name>
        <since>1966-08-22</since>
        <qualification>bold, brash and
tomboyish</qualification>
    </character>
</book>
```

- DTD pauvre

```
!ELEMENT book (title,author,character*)
!ELEMENT title(#PCDATA)
!ELEMENT author(#PCDATA)
!ATTLIST book isbn CDATA # FIXED "
    0836217462")
!ELEMENT character(name,friend-
    of?,since,qualification)
!ELEMENT name (#PCDATA)
!ELEMENT friend-of (#PCDATA)
!ELEMENT since (#PCDATA)
!ELEMENT qualification (#PCDATA)
```

- Un schéma pour ce document : apprécier la richesse : note.xsd

```
<xsd:schema
      xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
      <xsd:element name="book">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="title" type="xsd:string"/>
                      <xsd:element name="author" type="xsd:string"/>
                      <xsd:element name="character"
                            minOccurs="0" maxOccurs="unbounded">
                            <xsd:complexType>
                                 <xsd:sequence>
                                      <xsd:element name="name" type="xsd:string"/>
                                      <xsd:element name="friend-of" type="xsd:string"
                                                minOccurs="0" maxOccurs="unbounded"/>
                                      <xsd:element name="since" type="xsd:date"/>
                                      <xsd:element name="qualification" type="xsd:string"/>
                                 </xsd:sequence>
                                </xsd:complexType>
                  </xsd:element>
              </xsd:sequence>
              <xsd:attribute name="isbn" type="xsd:string"/>
           </xsd:complexType>
     </xsd:element>
</xsd:schema>
```

XML Schéma

Les premiers pas

Structure de base

- Comme tout document XML, un schéma XML commence par un prologue et a un élément racine

```
<?xml version="1.0" ?>
<xsd:schema …>
  <!-- déclarations d'éléments, d'attributs et de types ici -->
        …
</xsd:schema>
```

- L'élément racine est l'élément xsd:schema
  - Cet élément racine est accompagné d'attributs qui précisent le lieu de définition des éléments
- xsd utilise des espaces de noms pour distinguer les éléments appartenant à XSD (le langage) et les éléments et attributs définis par un schéma donné (par le programmeur)

XML Schéma

Les premiers pas

- Espaces de noms (namespaces) et préfixes
  - On peut soit définir un préfixe pour les éléments XSD (solution 1) soit pour vos éléments (solution 2)
  - Vous pouvez aussi choisir si vos éléments XML auront un namespace
- Solution 1 : donner un namespace au code XSD
  - Souvent on utilise le préfixe xs: pour le code XSD, parfois xsd: cela n'a pas d'importance
    - elementFormDefault="qualified" veut dire que vos balises n'auront pas de namespace

XML Schéma

Les premiers pas

- Exemple : XSD définition pour une simple recette de cuisine

```
<?xml version="1.0" encoding="UTF-8"?>
    <!-- Schema simple de recette -->
    <xs: schema
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified ">
          <xs:element name="list">
              <xs:complexType>
                  <xs:sequence>
                      <xs:element maxOccurs="unbounded" ref="recipe "/>
                  </xs:sequence>
              </xs:complexType>
          </xs:element>
    </xs:schema>
```

  - Cette solution est préférable en règle générale (mais voir plus loin comment associer une XSD à un fichier XML: il faut encore ajouter des attributs)

XML Schéma

Les premiers pas

- Solution 2: donner un namespace au code du schéma
  - Les éléments définis pour votre schéma ont un préfixe dans la définition
    - vous devez donc définir votre namespace (substituer "yourdomain.org/namespace")
  - On déclare que XML Schema a le namespace par défaut, c.a.d. les éléments XSD ne seront pas préfixés

XML Schéma

Les premiers pas

- Exemple : XSD définition pour une simple recette
  - On voit que schema et les éléments de xsd ne sont pas préfixés, mais que les éléments personnels le sont

```
<schema xmlns='http://www.w3.org/2000/10/XMLSchema'
    targetNamespace='http://yourdomain.org/namespace/ '
    xmlns:t='http://yourdomain.org/namespace/ '>
    <element name='t:list'>
            <complexType>
                    <sequence>
                            <element ref='t:recipe '
                             maxOccurs='unbounded'/>
                    </sequence>
            </complexType>
    </element>
</schema>
```

XML Schéma

Les premiers pas

- Validation (liaison du fichier XML avec le fichier XSD)
  - Un document XML décrit par un XSD est appelé instance document
  - **Association d'un XSD avec un fichier XML, Solution 1**
    - Il faut déclarer le namespace xsi: (XMLSchema-instance)
    - L'attribut *xsi:noNamespaceSchemaLocation* définit l'URL de votre XSD
    - Attention: il faut utiliser cela tel quel !!!
  - Exemple : **XML file (recipe-no-ns.xml)**

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<list xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation=" recipe-no-ns.xsd ">
    <recipe> ....
</list>
```

XML Schéma

Les premiers pas

- Validation
  - XSD file (recipe-no-ns.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
<xs:element name="list">…
```

XML Schéma

Les premiers pas

- Validation
  - Association d'un XSD avec XML, Solution 2, **Solution à retenir:**
    - **L'idée est que chaque fragment XML fait tjrs partie d'un namespace :**
      - Les fichiers XML et XSD doivent inclure une namespace declaration pour un domaine
    - Le fichier XML doit inclure en plus :
      - une déclaration pour le XMLSchema-instance namespace
      - un attribut xsi:schemaLocation qui dit où trouver XSD
        - Cet attribut peut contenir plusieurs pairs "namespace-URL"

- Exemple
- XML file (recipe.xml)

```
<?xml version="1.0" encoding="ISO-8859-1" ?> <list
    xmlns="http://myrecipes.org/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=" http://myrecipes.org/ recipe.xsd " >
<recipe>
<meta> .....</meta> ......
</recipe>
</list>
```

- XSD file (recipe.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>
    <!-- Simple recipe Schema -->
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema "
        targetNamespace="http://myrecipes.org/"
        xmlns="http://myrecipes.org/"
        elementFormDefault="qualified">
        ....
    </xs:schema>
```

XML Schema

Remarque

- La XSD définit un namespace pour vos balises
- Il faut substituer http://myrecipes.org/" par un URL de votre choix, mais de préférence un URL sur lequel vous avez le contrôle (par exemple votre home page)

XML Schéma

- Exemple

```
<manifest xmlns ="http://www.imsglobal.org/xsd/imscp_v1p1"
    xmlns:imsmd ="http://www.imsglobal.org/xsd/imsmd_v1p2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    identifier="MANIFEST-1"
xsi:schemaLocation= "http://www.imsglobal.org/xsd/imscp_v1p
    imscp_v1p1.xsd
http://www.imsglobal.org/xsd/imsmd_v1p2 imsmd_v1p2p2.xsd">
<metadata>
<imsmd:lom> ..... </imsmd:lom>
</metadata>
<organizations default="learning_sequence_1"> .....
```

- Cet exemple
  - montre comment utiliser deux espaces de nom pour deux XSD
    - imscp_v1p1 est le namespace par défaut (sans préfixe)
    - imsmd_v1p1 est le namespace pour les métadonnées

XML Schéma

- Extrait du fichier ims_cp_rootv1p1.xsd

```
<xsd:schema
  xmlns = "http://www.imsglobal.org/xsd/imscp_v1p1"
  targetNamespace = "http://www.imsglobal.org/xsd/imscp_v1p1"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
  version = "IMS CP 1.1.4"
  elementFormDefault = "qualified">
```

XML Schéma

Les premiers pas

- Résumé des attributs de xsd:schema

```
<xsd:schema
    xmlns = anyURI
    id= ID
    attributeFormDefault = qualified | unqualified
    elementFormDefault = qualified | unqualified
    blockDefault = (#all|list of (extension | restriction | substitution))
    finalDefault = (#all | list of (extension | restriction | list | union))
    targetNamespace = anyURI
    version = token
    any attributes
>
((include | import | redefine | annotation)*,(((simpleType | complexType |
group | attributeGroup) | element | attribute | notation),annotation*)*)
</xs:schema>
```

XML Schéma

Les premiers pas

- Attributs de xsd:schema
  - xmlns
    - Une référence URI qui spécifie un ou plusieurs namespaces à utiliser dans ce schéma
  - Id
    - Optionnel. Signale un unique identifiant pour l'élément
  - targetNamespace
    - Optionnel. Une URI référence du namespace de ce schéma
  - elementFormDefault
    - Optionnel. La valeur doit être "qualified" ou "unqualified". "unqualified" indique que les éléments du namespace cible ne sont pas obligés d'être accompagnés d'un préfixe du namespace
  - attributeFormDefault
    - Optionnel. La valeur doit être "qualified" ou "unqualified". …
  - blockDefault
    - Optionnel. définit la valeur par défaut de l'attribut **block** sur l'élément et les types complexes dans le **targetNamespace** du schéma
  - finalDefault
    - Optionnel. Indique la valeur par défaut de l'attribut final des éléments, simpleType, et complexType dans le namespace cible
  - Version
    - Optionnel. Spécifie la version du schéma

XML Schéma

Déclaration d'éléments

- Un élément, dans un schéma, se déclare avec la balise <xsd:element>
- Exemple :

```
<?xml version="1.0" ?>
<xsd:schema …>
    <xsd:element name="remarque" type="xsd:string"/>
    <xsd:element name="contacts" type="typeContacts"/>
    <!-- déclarations de types ici -->
</xsd:schema>
```

Ce schéma déclare deux éléments :

- remarque et contacts
- A chaque élément est associé un type via l'attribut type
  - remarque de type xsd:string, type simple prédéfini de XML Schema
  - contacts de type typeContacts, type complexe défini par l'utilisateur

XML Schéma

Les types de données

- XML Schema permet donc de spécifier des types de données bien plus finement que le langage DTD
- Il distingue notamment deux types :
  - types simples
  - types complexes

Types simples

- Ne peuvent comporter ni attributs, ni éléments enfants
- Il en existe de nombreux, prédéfinis, mais il est également possible d'en "dériver" de nouveaux
- Syntaxe de déclaration

```
<xs:element name="xxx" type="yyy"/>
```

XML Schéma

Types simples : exemple

- Document XML

```
<lastname>Refsnes</lastname>
<age>34</age>
<dateborn>1968-03-27</dateborn>
```

- Schéma associé

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

Les types simples (1)

- string
  - Confirm this is electric
- normalizedString
  - Confirm this is electric
- token
  - Confirm this is electric
- byte
  - -1, 126
- unsignedByte
  - 0, 126
- base64Binary
  - GpM7
- hexBinary
  - 0FB7
- integer
  - -126789, -1, 0, 1, 126789
- positiveInteger
  - 1, 126789
- negativeInteger
  - -126789, -1
- nonNegativeInteger
  - 0, 1, 126789
- nonPositiveInteger
  - -126789, -1, 0
- int
  - -1, 126789675
- unsignedInt
  - 0, 1267896754

Les types simples (2)

- long
  - -1, 12678967543233
- unsignedLong
  - 0, 12678967543233
- short
  - -1, 12678
- unsignedShort
  - 0, 12678
- decimal
  - -1.23, 0, 123.4, 1000.00
- float
  - -INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
- double
  - -INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
- boolean
  - true, false 1, 0
- time
  - 13:20:00.000, 13:20:00.000-05:00
- dateTime
  - 1999-05-31T13:20:00.000-05:00
- duration
  - P1Y2M3DT10H30M12.3S
- date
  - 1999-05-31
- gMonth
  - --05--
- gYear
  - 1999

Les types simples (3)

- gYearMonth
  - 1999-02
- gDay
  - ---31
- gMonthDay
  - --05-31
- Name
  - shipTo
- QName
  - po:USAddress
- NCName
  - USAddress
- anyURI
  - http://www.example.com/,
  - http://www.example.com/doc.html#ID5
- language
  - en-GB, en-US, fr
- ID
  - "A212"
- IDREF
  - "A212"
- IDREFS
  - "A212" "B213"
- ENTITY
- ENTITIES
- NOTATION
- NMTOKEN, NMTOKENS
  - US
  - Brésil Canada Mexique

XML Schéma

Type simple

Valeur par défaut et valeur fixée

- Un élément simple peut avoir une valeur par défaut ou une valeur fixée
  - Une valeur par défaut est affectée si aucune autre valeur ne l'est

    ```
    <xs:element name="color" type="xs:string" default="red"/>
    ```
  - Une valeur fixée est aussi affectée mais peut être changée

    ```
    <xs:element name="color" type="xs:string" fixed="red"/>
    ```

XML Schéma

Déclaration d'attributs

- Les attributs sont déclarés de types simples

  ```
  <xs:attribute name="xxx" type="yyy"/>
  ```

- Seuls les éléments complexes peuvent avoir des attributs
  - Exemple d'élément XML

    ```
    <lastname lang="EN">Smith</lastname>
    ```

- Voici la définition correspondante de type simple

  ```
  <xs:attribute name="lang" type="xs:string"/>
  ```

- On peut également affecter une valeur par défaut

  ```
  <xs:attribute name="lang" type="xs:string" default="EN"/>
  ```

- On peut également affecter une valeur fixe

  ```
  <xs:attribute name="lang" type="xs:string" fixed="EN"/>
  ```

XML Schéma

Déclaration d'attributs

- Tous les attributs sont optionnels par défaut, on peut cependant les expliciter :
  - <xs:attribute name="lang" type="xs:string" use="optional"/>
- On peut les rendre obligatoires
  - <xs:attribute name="lang" type="xs:string" use="required"/>

XML Schéma

- Déclaration d'attributs : restriction
  - Des restrictions sont utilisées sur les attributs pour ne tolérer que des valeurs acceptables
  - Les restrictions en XML sont appelées facettes
- Restriction à des valeurs
  - Cet exemple définit un élément age avec une restriction : la valeur doit être comprise ente 0 et 100

```
<xs:element name="age">
    <xs:simpleType>
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="100"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

XML Schéma

Restriction à un ensemble de valeurs

- Pour limiter l'étendue d'un élément de XML à un ensemble de valeurs acceptables, on a l'habitude d'utiliser l'énumération
- Cet exemple montre l'élément "car" avec restriction à "Audi", "Golf" et "BMW"

```
<xs:element name="car">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="Audi"/>
            <xs:enumeration value="Golf"/>
            <xs:enumeration value="BMW"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

XML Schéma

Restriction à un ensemble de valeurs

- On peut également l'écrire comme suit :

```
<xs:element name="car" type="carType"/>
    <xs:simpleType name="carType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Audi"/>
            <xs:enumeration value="Golf"/>
            <xs:enumeration value="BMW"/>
        </xs:restriction>
    </xs:simpleType>
```

- Dans ce cas-ci le type "carType" peut être employé par d'autres éléments parce que ce n'est pas une partie de l'élément "voiture"

XML Schéma

Restriction à une série de valeurs

- Restrictrion de "letter" à des lettres minuscules :

```
<xs:element name="letter">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="[a-z]"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

XML Schéma

Restriction à une série de valeurs

- Restriction de "initials" à trois lettres majuscules :

```
<xs:element name="initials">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="[A-Z][A-Z][A-Z]"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

XML Schéma

Restriction à une série de valeurs

- Restriction de "choice" à l'une des trois lettres x, y ou z :

```
<xs:element name="choice">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="[xyz]"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

XML Schéma

Autres exemples de restrictions

- On préserve tous les espaces dans la chaîne

```
<xs:restriction base="xs:string">
<xs:whiteSpace value="preserve"/>
```

- On remplace toutes les zones d'espaces, de tabulations, de justifications par des espaces

```
<xs:restriction base="xs:string">
<xs:whiteSpace value="replace"/>
```

- On remplace toutes les zones d'espaces dans la chaîne par un espace

```
<xs:restriction base="xs:string">
<xs:whiteSpace value="collapse"/>
```

XML Schéma

Résumé des contraintes sur les types de données

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

XML Schéma

Composition de types simples

Liste

- Les types listes sont des suites de types simples (ou *atomiques*).
- XML Schema possède trois types de listes intégrés :
  - NMTOKENS (suite séparée par des blancs), ENTITIES et IDREFS
- Il est également possible de créer une liste personnalisée, par "dérivation" de types existants
- Par exemple :

```
<xsd:simpleType name="numéroDeTéléphone">
  <xsd:list itemType="xsd:unsignedByte" />
</xsd:simpleType>
```

- Un élément conforme à cette déclaration serait :

```
<téléphone>01 44 27 60 11</téléphone>
```

Un autre exemple

- Type List pour 6 états Us

```
<xsd:simpleType name="SixUSStates">
  <xsd:restriction base="USStateList">
    <xsd:length value="6"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="USStateList">
  <xsd:list itemType="USState"/>
</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>
```

XML Schéma

Union

- Les listes et les types simples intégrés ne permettent pas de choisir le type de contenu d'un élément
  - On peut désirer, par exemple, qu'un type autorise soit un nombre, soit une chaîne de caractères particuliers
  - Il est possible de le faire à l'aide d'une déclaration d'union
- Par exemple, sous réserve que le type simple numéroDeTéléphone ait été préalablement défini (voir précédemment), on peut déclarer...

```
<xsd:simpleType name="numéroDeTéléphoneMnémoTechnique">
  <xsd:union memberTypes="xsd:string numéroDeTéléphone" />
</xsd:simpleType>
```

- Les éléments suivants sont alors des "instances" valides de cette déclaration

```
<téléphone>18</téléphone>
<téléphone>Pompiers</téléphone>
```

XML Schéma

Types ou éléments complexes

- Un élément de type simple ne peut contenir de sous-éléments, et des attributs, d'où l'intérêt des éléments complexes
- Il y a quatre sortes d'éléments complexes :
  - Éléments vides
  - Éléments qui contiennent d'autres éléments
  - Éléments qui contiennent seulement du texte
  - Éléments qui contiennent d'autres éléments et du texte

XML Schéma

Exemples d'éléments complexes

- Élément vide

```
<product pid="1345"/>
```

- Élément contenant d'autres éléments

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

- Élément contenant que du texte

```
<food type="dessert">Ice cream</food>
```

- Élément contenant des éléments et du texte

```
<description> It happened on
  <date lang="norwegian">03.03.99</date> ....
</description>
```

XML Schéma

- Comment définir un élément complexe ?
  - Observons cet élément complexe qui contient d'autres éléments

```
<employee>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</employee>
```

  - On peut définir l'élément dans un schéma de différentes manières

XML Schéma

1. L'élément "employee" peut être déclaré directement en distinguant ses composants, comme ceci:

```
<xs:element name="employee">
 <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
 </xs:complexType>
</xs:element>
```

- Mais ceci
  - ne conduit que "employee" à utiliser l'appellation complexType
  - Les enfants "firstname" et "lastname" sont entourés de "sequence" les conduisant à être utilisés dans l'ordre où ils apparaissent

XML Schéma

2. L'élément "employee" peut avoir un type d'attribut qui se réfère au nom du type complex à utiliser :

```
<xs:element name="employee" type="personinfo"/>
    <xs:complexType name="personinfo">
    <xs:sequence>
            <xs:element name="firstname" type="xs:string"/>
            <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
    </xs:complexType>
```

- En utilisant cette méthode
  - plusieurs éléments peuvent ainsi se référer au type complexe "persoinfo", comme ceci :

XML Schéma

Exemple :

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>
<xs:complexType name="personinfo">
    <xs:sequence>
                <xs:element name="firstname" type="xs:string"/>
                <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
```

XML Schéma

- On peut également se baser sur un élément d'un type complexe existant et lui ajouter des éléments :

```
<xs:element name="employee" type="fullpersoninfo"/>
    <xs:complexType name="personinfo">
        <xs:sequence>
          <xs:element name="firstname" type="xs:string"/>
          <xs:element name="lastname" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="fullpersoninfo">
        <xs:complexContent> // définit des extensions ou des restrictions sur complexType
            <xs:extension base="personinfo">
                    <xs:sequence>
                            <xs:element name="address" type="xs:string"/>
                            <xs:element name="city" type="xs:string"/>
                            <xs:element name="country" type="xs:string"/>
                    </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
```

"employee" doit contenir, dans l'ordre : "firstname", "lastname", "adresse", "ville et "pays"

XML Schéma

- Comment définir un type complexe pour un élément vide ?
  - Exemple
    - <product prodid="1345" />
  - L'élément "product" ci-dessus n'a aucun contenu
  - Pour définir un type sans contenu, nous devons définir un type qui permet seulement des éléments dans son contenu, mais nous ne déclarons réellement aucun élément, comme ceci :

```
<xs:element name="product">
        <xs:complexType>
          <xs:complexContent>
            <xs:restriction base="xs:integer">
              <xs:attribute name="prodid" type="xs:positiveInteger"/>
            </xs:restriction>
          </xs:complexContent>
        </xs:complexType>
</xs:element>
```

XML Schéma

- On peut déclarer l'élément "product" de manière plus compacte :

```
<xs:element name="product">
        <xs:complexType>
           <xs:attribute name="prodid" type="xs:positiveInteger"/>
        </xs:complexType>
</xs:element>
```

XML Schéma

- On peut également donner à l'élément de complexType un nom, et laisser l'élément "product" avoir un attribut de type qui se rapporte au nom du complexType

  ➔ si on emploie cette méthode, plusieurs éléments peuvent se rapporter au même type complexe :

```
<xs:element name="product" type="prodtype"/>
    <xs:complexType name="prodtype">
      <xs:attribute name="prodid" type="xs:positiveInteger"/>
    </xs:complexType>
```

XML Schéma

Type complexe à éléments

- Un type complexe à éléments est un type complexe contenant un élément qui contient seulement d'autres éléments
- Exemple : élément "person" contenant d'autres éléments

```
<person>
   <firstname>John</firstname>
   <lastname>Smith</lastname>
</person>
```

- On peut définir l'élément "person" dans un schéma comme suit :

```
<xs:element name="person">
   <xs:complexType>
            <xs:sequence>
                     <xs:element name="firstname" type="xs:string"/>
                     <xs:element name="lastname" type="xs:string"/>
            </xs:sequence>
   </xs:complexType>
</xs:element>
```

XML Schéma

Type complexe à éléments

- Sachant que la séquence conduit à utiliser les fils dans l'ordre où ils ont été déclarés, on peut proposer la solution suivante qui consiste à :
  - donner à l'élément de complexType un nom, et laisser l'élément "person" avoir un attribut de type qui se rapporte au nom du complexType
  - En employant cette méthode, plusieurs éléments peuvent se rapporter au même type complexe :

```
<xs:element name="person" type="persontype"/>
  <xs:complexType name="persontype">
    <xs:sequence>
        <xs:element name="firstname" type="xs:string"/>
        <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
```

XML Schéma

Type complexe à éléments textuels

- Ce type peut contenir des attributs et du texte, donc on ajoute un élément de simpleContent autour du contenu de manière à pouvoir faire des extensions ou des restrictions de ce contenu :
- Avec l'extension

```
<xs:element name="somename">
   <xs:complexType>
         <xs:simpleContent>
               <xs:extension base="basetype">
               .... ....
               </xs:extension>
         </xs:simpleContent>
   </xs:complexType>
</xs:element>
```

XML Schéma

- OU avec la restriction

```
<xs:element name="somename">
   <xs:complexType>
         <xs:simpleContent>
               <xs:restriction base="basetype">
               .... ....
               </xs:restriction>
         </xs:simpleContent>
   </xs:complexType>
</xs:element>
```

XML Schéma

- Exemple

```
<shoesize country="france">35</shoesize>
```

- L'exemple suivant déclare un complexType "shoesize"
- Le contenu est défini comme un entier et l'élément "shoesize" contient un attribut appelé "country" :

```
<xs:element name="shoesize">
  <xs:complexType>
      <xs:simpleContent> // permet l'extension de complexType de ce type
        <xs:extension base="xs:integer">
            <xs:attribute name="country" type="xs:string" />
        </xs:extension>
      </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

XML Schéma

- Nous pourrions également donner à l'élément de complexType un nom, et laisser "shoesize" avoir un attribut de type qui se rapporte au nom du complexType

  ➔si vous employez cette méthode, plusieurs éléments peuvent se rapporter au même type complexe :

```
<xs:element name="shoesize" type="shoetype"/>
  <xs:complexType name="shoetype">
    <xs:simpleContent>
          <xs:extension base="xs:integer">
            <xs:attribute name="country" type="xs:string" />
          </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
```

XML Schéma

Type complexe à contenu mixte

- Il peut contenir des attributs, des éléments et du texte
- Exemple : l'élément "letter "

```
<letter>
   Dear Mr.<name>John Smith</name>.
   Your order <orderid>1032</orderid>
   will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
```

- Notez que le texte qui paraît entre les éléments "name", "orderid " et " shipdate" sont tous les enfants de "letter"

XML Schéma

Type complexe à contenu mixte

- Le texte suivant déclare l'élément "letter " :

```
<xs:element name="letter">
  <xs:complexType mixed="true" >
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="orderid" type="xs:positiveInteger"/>
            <xs:element name="shipdate" type="xs:date"/>
        </xs:sequence>
  </xs:complexType>
</xs:element>
```

- Pour permettre à du texte d'apparaître entre les enfants-éléments de "letter", l'attribut mixed doit être à "vrai"
- L'étiquette <xs:sequence> signifie que les éléments définis (name, orderid et shipdate) doivent apparaître dans cet ordre à l'intérieur d'un élément "letter"

XML Schéma

Type complexe à contenu mixte

- Nous pourrions également donner à l'élément de complexType un nom, et laisser l'élément "letter" avoir un attribut de type qui se rapporte au nom du complexType
- Si on utilise cette méthode, plusieurs éléments peuvent se rapporter au même type complexe :

```
<xs:element name="letter" type="lettertype"/>
  <xs:complexType name="lettertype" mixed="true">
      <xs:sequence>
          <xs:element name="name" type="xs:string"/>
          <xs:element name="orderid" type="xs:positiveInteger"/>
          <xs:element name="shipdate" type="xs:date"/>
      </xs:sequence>
</xs:complexType>
```

XML Schéma

Type complexe avec indicateurs

- Les indicateurs permettent de contrôler l'usage des éléments à l'intérieur du type complexe
- Il existe 7 types d'indicateurs :
  - Indicateurs d'ordre :
    - All
    - Choice
    - Sequence
  - Indicateurs d'occurrence :
    - maxOccurs
    - minOccurs
  - Indicateurs de groupes :
    - Group name
    - attributeGroup name

XML Schéma

Type complexe avec indicateurs

Indicateurs d'ordre

- Sont employés pour définir comment les éléments devraient se produire
- L'indicateur all
  - indique que les éléments enfant peuvent apparaître dans n'importe quel ordre et que chaque élément doit se produire une et une seule fois :

```
<xs:element name="person">
   <xs:complexType>
      <xs:all>
                  <xs:element name="firstname" type="xs:string"/>
                  <xs:element name="lastname" type="xs:string"/>
      </xs:all>
   </xs:complexType>
</xs:element>
```

XML Schéma

Type complexe avec indicateurs

- L'indicateur choice
  - indique qu'un seul parmi les éléments enfants peut se produire:

```
<xs:element name="person">
   <xs:complexType>
      <xs:choice>
                  <xs:element name="employee" type="employee"/>
                  <xs:element name="member" type="member"/>
      </xs:choice>
   </xs:complexType>
</xs:element>
```

XML Schéma

Type complexe avec indicateurs

- L'indicateur sequence
  - spécifie que les éléments enfants doivent apparaître dans un certain ordre :

```
<xs:element name="person">
   <xs:complexType>
      <xs:sequence>
                  <xs:element name="firstname" type="xs:string"/>
                  <xs:element name="lastname" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

XML Schéma

Type complexe avec indicateurs

Indicateurs d'occurrence

- spécifie comment un élément peut se produire :
- L'indicateur maxOccurs :
  - indique le nombre maximum de fois qu'un élément peut se produire

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
        maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

  - Le temps d'occurrence minimum par défaut est 1

XML Schéma

Type complexe avec indicateurs

- L'indicateur minOccurs :
  - indique le nombre minimum de fois qu'un élément peut se produire

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
        <xs:element name="full_name" type="xs:string"/>
        <xs:element name="child_name" type="xs:string"
            maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

  - Pour permettre à un élément d'apparaître un nombre illimité de fois, utiliser un maxOccurs="unbounded"

XML Schéma

Type complexe avec indicateurs

Indicateurs de groupe

- employés pour définir des ensembles relatifs d'éléments
- Groupe d'éléments : déclarés par :

```
<xs:group name="groupname">
...
</xs:group>
```

- On doit définir des all, choice, ou un élément d'ordre à l'intérieur de la déclaration de groupe :
  - le suivant définit un groupe d'éléments qui doivent se produire comme une séquence

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>
```

XML Schéma

Type complexe avec indicateurs

- Après avoir défini un groupe, on peut s'y référer dans un autre groupe ou type complexe comme ceci :

```
<xs:group name="persongroup">
   <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="birthday" type="xs:date"/>
   </xs:sequence>
</xs:group>

<xs:element name="person" type="personinfo"/>
<xs:complexType name="personinfo">
   <xs:sequence>
      <xs:group ref="persongroup"/>
      <xs:element name="country" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
```

XML Schéma

Type complexe avec indicateurs

- Groupe d'attributs :
  - Défini à l'aide de la déclaration attributeGroup :

```
<xs:attributeGroup name="groupname">
...
</xs:attributeGroup>
```

  - Exemple :

```
<xs:attributeGroup name="personattrgroup">
   <xs:attribute name="firstname" type="xs:string"/>
   <xs:attribute name="lastname" type="xs:string"/>
   <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>
```

XML Schéma

Type complexe avec indicateurs

- Une fois défini, on pourra s'y référer dans un autre groupe ou autre type complexe
- Exemple :

```
<xs:attributeGroup name="personattrgroup">
    <xs:attribute name="firstname" type="xs:string"/>
    <xs:attribute name="lastname" type="xs:string"/>
    <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>

<xs:element name="person">
    <xs:complexType>
        <xs:attributeGroup ref="personattrgroup"/>
    </xs:complexType>
</xs:element>
```

XML Schéma

L’élément <any>

- permet d’étendre le document de XML avec des éléments non indiqués par le schéma
- L'exemple suivant est un fragment du schéma "family.xsd". Il montre une déclaration de l'élément "person". En employant <any> on peut prolonger (après <lastname>) "person" avec n'importe quel élément :

```
<xs:element name="person">
  <xs:complexType>
      <xs:sequence>
        <xs:element name="firstname" type="xs:string"/>
        <xs:element name="lastname" type="xs:string"/>
        <xs:any minOccurs="0"/>
      </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML Schéma

L’élément <any>

- Maintenant on veut prolonger l'élément "person" avec un élément "children"
- Dans ce cas-ci on peut faire ainsi, même si l'auteur du schéma ci-dessus n’avait jamais déclaré aucun élément "children"!
- Exemple : regarder : children.xsd

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:…>
        <xs:element name="children">
            <xs:complexType>
            <xs:sequence>
                <xs:element name="childname" type="xs:string"
                    maxOccurs="unbounded"/>
           </xs:sequence>
            </xs:complexType>
        </xs:element>
</xs:schema>
```

XML Schéma

L'élément <any>

- Le fichier XML ci-dessous (appelé " Myfamily2.xml"), utilise des composants issus de deux schémas différents; "family.xsd" et "children.xsd" :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<persons xmlns… xsi:SchemaLocation="http://www.microsoft.com family.xsd
   http://www.w3schools.com children.xsd">
        <person>
            <firstname>Hege</firstname>
            <lastname>Refsnes</lastname>
            <children>
                    <childname>Cecilie</childname>
            </children>
        </person>
        <person>
            <firstname>Stale</firstname>
            <lastname>Refsnes</lastname>
        </person>
    </persons>
```

XML Schéma

L'élément <anyAttribute>

- Nous permet d'étendre le document XML avec des attributs non prévus par le schéma
- Exemple : on peut ajouter des attributs à l'élément "person"

```
<xs:element name="person">
  <xs:complexType>
      <xs:sequence>
          <xs:element name="firstname" type="xs:string"/>
          <xs:element name="lastname" type="xs:string"/>
      </xs:sequence>
      <xs:anyAttribute/>
  </xs:complexType>
</xs:element>
```

XML Schéma

L’élément <anyAttribute>

- Maintenant nous voulons prolonger l'élément "person" avec un attribut "gender"
- Exemple : attribute.xsd

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:…>
    <xs:attribute name="gender">
      <xs:simpleType>
        <xs:restriction base="xs:string">
                <xs:pattern value="male|female"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
</xs:schema>
```

XML Schéma

L’élément <anyAttribute>

- Le fichier XML ci-dessopus (called "Myfamily3.xml"), utilise des composants de deux schémas différents "family.xsd" et "attribute.xsd" :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<persons xmlns="http://www.microsoft.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:SchemaLocation="http://www.microsoft.com family.xsd
  http://www.w3schools.com attribute.xsd">
    <person gender="female">
      <firstname>Hege</firstname>
      <lastname>Refsnes</lastname>
    </person>
    <person gender="male">
      <firstname>Stale</firstname>
      <lastname>Refsnes</lastname>
    </person>
</persons>
```

XML Schéma

L’élément Substitution

- Permet à un élément d’être substitué par un autre
- Problème :
  - nous avons des utilisateurs de deux pays différents : Angleterre et Norvège. Nous voudrions laisser l'utilisateur choisir s’il veut employer des noms d'éléments norvégiens ou anglais dans le document de XML
- Pour résoudre ce problème, nous pourrions définir un substitutionGroup dans le schéma de XML
- D'abord, nous déclarons un élément principal et ensuite nous déclarons les autres éléments comme remplaçables pour l'élément principal
- Exemple :

```
<xs:element name="name" type="xs:string"/>
<xs:element name="navn" substitutionGroup="name"/>
```

- Dans l'exemple,
  - l'élément "name" est l'élément principal et "navn" est remplaçable par "name"

XML Schéma

- Soit un fragment d'un schéma :

```
<xs:element name="name" type="xs:string"/>
<xs:element name="navn" substitutionGroup="name"/>
    <xs:complexType name="custinfo">
        <xs:sequence>
            <xs:element ref="name"/>
        </xs:sequence>
    </xs:complexType>
<xs:element name="customer" type="custinfo"/>
<xs:element name="kunde" substitutionGroup="customer"/>
```

- Un document XML valide (au regard du schéma précédent) doit ressembler à ceci :

```
<customer>
   <name>John Smith</name>
</customer>
```

ou cela :

```
<kunde>
   <navn>John Smith</navn>
</kunde>
```

XML Schéma

- Regroupement en bloc des éléments de substitution
  - Pour empêcher d'autres éléments de substituer avec un élément indiqué, employez l'attribut block :

```
<xs:element name="name" type="xs:string" block="substitution"/>
```

  - Soit ce fragment d'un document XML Schema :

```
<xs:element name="name" type="xs:string" block="substitution"/>
<xs:element name="navn" substitutionGroup="name"/>
  <xs:complexType name="custinfo">
        <xs:sequence>
              <xs:element ref="name"/>
        </xs:sequence>
  </xs:complexType>
<xs:element name="customer" type="custinfo" block="substitution"/>
<xs:element name="kunde" substitutionGroup="customer"/>
```

XML Schéma

- Un document valide de XML (selon le schéma ci-dessus) ressemble à ceci :

```
<customer>
    <name>John Smith</name>
</customer>
```

- MAIS CECI N'EST PLUS VALIDE :

```
<kunde>
    <navn>John Smith</navn>
</kunde>
```

XML Schéma

- Exemple
  - Soit le document XML : "shiporder.xml"
  - Créer le Schéma "shiporder.xsd"
  - Étape 1 :
    - ouvrir un nouveau dossier que nous appellerons "shiporder.xsd ". Pour créer le schéma, nous pourrions simplement suivre la structure dans le document de XML et définir chaque élément comme nous le trouvons. Nous commencerons par la déclaration standard de XML suivie de l'élément de xs:schema qui définit un schéma :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
... ...
</xs:schema>
```

    - Dans ce schéma on se réfère au namespace standard (xs), donné à l'URI : http://www.w3.org/2001/XMLSchema

XML Schéma

- Étape 2 :
  - On doit définir l'élément "shiporder". Cet élément a un attribut et il contient d'autres éléments, donc nous le considérons comme type complexe
  - Les éléments d'enfant de l'élément " shiporder " est entourés par un élément de xs:sequence qui définit un ordre sur les éléments secondaires :

```
<xs:element name="shiporder">
   <xs:complexType>
      <xs:sequence>
                    .. ...
      </xs:sequence>
                    ...
   </xs:complexType>
</xs:element>
```

XML Schéma

- Étape 3 :
  - Ensuite, nous devons définir l'élément " orderperson " comme type simple (parce qu'il ne contient aucun attribut ou d'autres éléments)
  - Le type (xs:string) est mis en tête avec le préfixe de namespace lié au schéma de XML qui indique un type de données prédéfini de schéma :

```
<xs:element name="orderperson" type="xs:string"/>
```

- Étape 4 :
  - Après, nous devons définir deux éléments qui sont de type complexe : " shipto " et " item "
  - Nous commençons par définir l'élément d'" shipto ":

```
<xs:element name="shipto">
    <xs:complexType>
        <xs:sequence>
                <xs:element name="name" type="xs:string"/>
                <xs:element name="address" type="xs:string"/>
                <xs:element name="city" type="xs:string"/>
                <xs:element name="country" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

XML Schéma

- Maintenant nous pouvons définir l'élément "item ". Cet élément peut apparaître de multiples fois à l'intérieur d'un élément " shiporder ". Ceci est indiqué en plaçant l'attribut de maxOccurs de l'élément "item" à "unbounded". Notez que l'élément "note" est facultatif. Nous avons indiqué ceci en plaçant l'attribut minOccurs à zéro :

```
<xs:element name="item" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="note" type="xs:string" minOccurs="0"/>
      <xs:element name="quantity" type="xs:positiveInteger"/>
      <xs:element name="price" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML Schéma

- Étape 5 :
  - Nous pouvons maintenant déclarer l'attribut de l'élément "shiporder". Puisque c'est un attribut obligatoire nous indiquons use="required"

    <xs:attribute name="orderid" type="xs:string" use="required"/>
- Le résultat se trouve dans : shiporder.xsd
  - Cette méthode de conception est très simple mais peut s'avérer très difficile à lire et à maintenir quand les documents sont complexes !
- On propose alors d'autres manières de procéder :
  - En modularisant le schéma : shiporder2.xsd
    - Définition d'abord de tous les éléments et attributs puis référence à eux en employant l'attribut de référence
  - En nommant des types : shiporder3.xsd
    - définit des classes ou des types nous permettant de réutiliser des définitions d'éléments. Ceci est fait en nommant les éléments de simpleTypes et de complexTypes, puis en se référant à eux en utilisant l'attribut de type de l'élément