Résumé

Résumé pour le travail écrit n° 1 de la branche « Systèmes d'Information »

Table des matières

1 GIT **2**
- 1.1 Connaître la différence entre les systèmes de contrôle de version centralisés et distribués . . . 2
  - 1.1.1 Centralisés (client-serveur) . . . 2
  - 1.1.2 Distribués (DVCS) . . . 2
- 1.2 Définir GIT et expliquer son rôle . . . 2
- 1.3 Pouvoir comparer GIT avec d'autres systèmes de contrôle de version comme SVN ou CVS . . . 3
- 1.4 Connaître les commandes de base GIT . . . 3

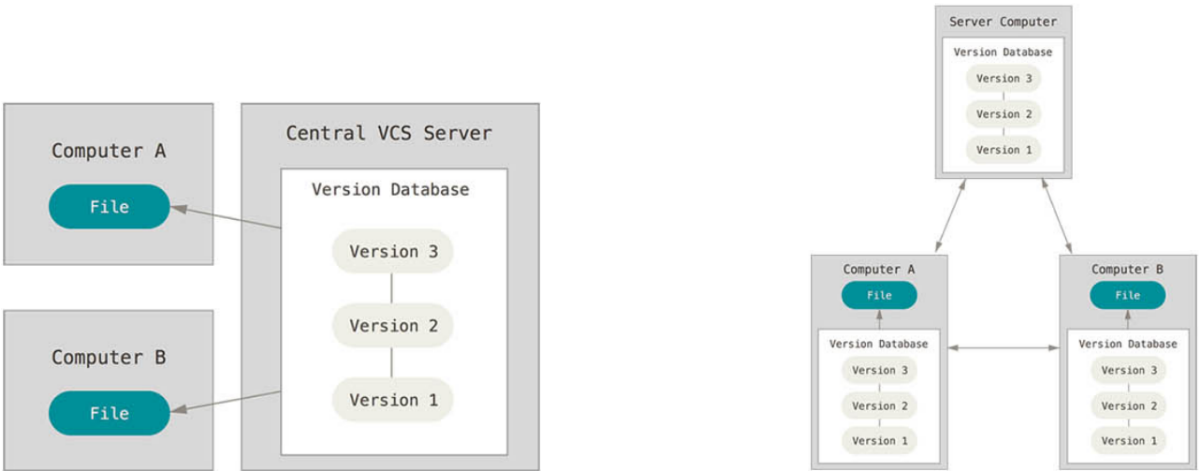
2 XML **4**
- 2.1 Savoir définir XML et expliquer son rôle . . . 4
  - 2.1.1 Composants d'un XML . . . 4
- 2.2 Etre capable de construire un document XML (Well-formed & Valid) . . . 5
  - 2.2.1 Well-formed . . . 5
  - 2.2.2 Valide . . . 5
- 2.3 Comprendre les modèles de production des documents . . . 7
- 2.4 Expliquer la différence entre les documents linéaires et structurés . . . 8
  - 2.4.1 Documents linéaires . . . 8
  - 2.4.2 Document structurés . . . 8

3 XML Schema **9**
- 3.1 Définir ce que sont les modèles de documents et pourquoi ils sont utilisés . . . 9
- 3.2 Définir DTD et expliquer son rôle . . . 9
- 3.3 Evaluer, juger et critiquer la différence entre DTD et XML Schema . . . 10
- 3.4 Pouvoir illustrer, expliquer et identifier les composants principaux d'un XML Schema . . . 10
  - 3.4.1 Les éléments, les attributs, types simples et complexes . . . 11
  - 3.4.2 Composants d'aide (annotation, etx.) . . . 13
- 3.5 Pouvoir expliquer et identifier les notions suivantes . . . 13
  - 3.5.1 Déclaration d'éléments locaux et globaux . . . 13
  - 3.5.2 Définition de types nommés et anonymes (Inline declaration, referencing declarations, named and anonymous types) . . . 14
  - 3.5.3 Dérivation par restriction et par extension . . . 14
- 3.6 Maîtriser l'utilisation et justifier les avantages et les désavantages des principes suivants . . . 16
  - 3.6.1 Russian Doll, Salami Slice, Venetian Blind, Garden of Eden . . . 16
  - 3.6.2 Local élément vs Global élément . . . 19
  - 3.6.3 Types nommés vs types anonymes (Named vs. Anonymous Types) . . . 19
- 3.7 Exposer, expliquer les notions suivantes . . . 19

3.7.1 Compositors (sequence, any, choice), groupe (d'éléments et d'attributs) et les modèles de contenu (mixed, empty, etc...) . . . 19
3.7.2 Namesapce, targetNamespace, qualified, unqualified, elementFormDefault, etc... . . . 20
3.8 Illustrer les notions suivantes et savoir quand les utiliser . . . 22
3.8.1 import . . . 22
3.8.2 include . . . 22
3.8.3 redefine . . . 22

1 GIT

1.1 Connaître la différence entre les systèmes de contrôle de version centralisés et distribués



(a) Centralisé (modèle client-serveur) (b) Distribué

FIGURE 1 – Les différents systèmes de contrôle

1.1.1 Centralisés (client-serveur)

— Toutes les informations sont contenues dans un dépôt unique
— Exemples : SVN, CVS.

1.1.2 Distribués (DVCS)

— Les dépôts (repositories) sont distribués sur plusieurs machines (serveur(s) + machines des utilisateurs)
— Commits possibles sans réseau
— Uniquement push et pull nécessitent une connexion réseau
— Exemples : GIT, Mercurial...

1.2 Définir GIT et expliquer son rôle

GIT est : libre, open source, rapide, flexible et très utilisé. La première version fut écrite par Linus Torvalds en 2005. Un fichier suivi par GIT peut être dans 3 états :

modifié fichier modifié mais pas encore validé en base

indexé fichier modifié marqué pour qu'il fasse partie du prochain instantané du projet

validé les données sont stockées en sécurité dans votre base de données **locale**

Un projet GIT se divise en trois section principales :

Le répértoire de travail (working directory) est une extraction unique d'une version du projet. Ces fichiers sont extraits depuis la base de données compressés dans le répertoire Git et placés sur le disque pour pouvoir être utilisés ou modifiés.
git add permet de passer dans la staging area

La zone d'index (staging area) est un simple fichier, généralement situé dans le répertoire Git, qui stocke les informations concernant ce qui fera partie du prochain instantané
git commit permet de passer dans le GIT repository

répertoire GIT (GIT repository) est l'endroit où Git stocke les méta-données et la base de données des objets de votre projet. C'est la partie la plus importante de Git, et c'est ce qui est copié lorsque vous clonez un dépôt depuis un autre ordinateur

1.3 Pouvoir comparer GIT avec d'autres systèmes de contrôle de version comme SVN ou CVS

« En cas de doute, faire l'exacte opposé de ce que propose CVS »

GIT information = flux instantanés

Les autres information = liste de fichiers et leurs modifications dans le temps

1.4 Connaître les commandes de base GIT

```
# En savoir plus sur une commande
git help nom_commande
git nom_commande --help

# Récupérer un dépot vie le système
# mettre le contenu du dépôt dans le répertoire courant, ce dernier doit être vide.
# ou mettre le dépôt dans un répertoire créé automatiquement au nom du projet
git clone <URL_DEPOT> .

git pull # Mettre a jour le dépot local
git sattus # Vérifier l'état de votre dépot local
git add FileName # Placer de nouveaux fichiers à l'index ou ré-indexer un fichier modifié
git reset chemin/du/fichier # Retire le fichier de l'index (pas encore commit)
git rm [-r] filename # Supprime le fichier de l'index (déjà commit)
git mv str1 str2 # Pareil que le mv linux
git commit -m "Message du commit"
git push [origin <branchName>] # Envoyer les commits sur le dépôt distant

git branch <branchName> # Crée une branche
git checkout <branchName> # Change la branche courante
git merge <branchname> # Fusionne la branche <branchName> dans la branche actuelle
```

```
# A expliquer
git stash
git rebase
```

2 XML

2.1 Savoir définir XML et expliquer son rôle

e**X**tensible **M**arkup **L**anguage. Langage de balisage extensible : on peut définir de nouvelles balises. Sépare le contenu de la présentation : permet le traitement automatique des documents.

« Le langage XML est un langage qui permet de décrire des données à l'aide de balises et de règles que l'on peut personnaliser »

- openclassroom.com

A quoi peut servir XML ?
- Publication électronique internationalisée
- Application Web

2.1.1 Composants d'un XML

- Un document XML contient du texte
- Ce n'est pas obligatoirement un fichier, il peut
  - être stocké dans une base de données
  - être créé à la volée en mémoire par un programme
  - être la combinaison de plusieurs fichiers imbriqués
  - ne jamais exister sous forme de fichier permanent
- Un XML est constitué d'unités élémentaires de stockage nommées entités, celle-ci contient :
  - soit du texte (des caractères)
  - soit des données binaires (des images)
  - **Jamais les deux à la fois pour le même élément**
- Syntaxe de déclaration

```
<?xml version ="version_number" encoding="encoding_declarations"
standalone="standalone_status"?>
<!-- Exemple -->
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

- Des commentaires :

```
<!-- this is a comment -->
```

- Références d'entités symboliques : `\&`pour &, &lt ; pour >, ...
- Sections littérales CDATA (utile pour empêcher toute interprétation de ce qu'elles contiennes), exemple :

```
<![CDATA[
<?xml version=‘‘1.0’’ standalone=‘‘yes’’?>
<salutation>
Bonjour tout le monde
```

`<salutation>`
`]]>`

On ne peut pas imbriquer le texte avec le type CDATA

— Des balises :
`<balise>...</balise>, <bal_vide> /`

Les noms des balises : commencent par (a-z | A-Z | _). Les espaces entre les mots sont interdits.

2.2 Etre capable de construire un document XML (Well-formed & Valid)

2.2.1 Well-formed

— Un document bien formé est syntaxiquement correct
— Il respecte les règles de syntaxe suivante :
  — Il doit y avoir un élément racine
  — Chaque élément doit avoir un tag fermant
  — Les tags sont sensibles à la casse
  — Les éléments doivent être correctement imbriqués
  — Les valeurs d'attributs doivent être entre guillemets

2.2.2 Valide

— Il est Well-formed
— **Il est conforme à un modèle :** DTD (Document type definition) ou XML Schema

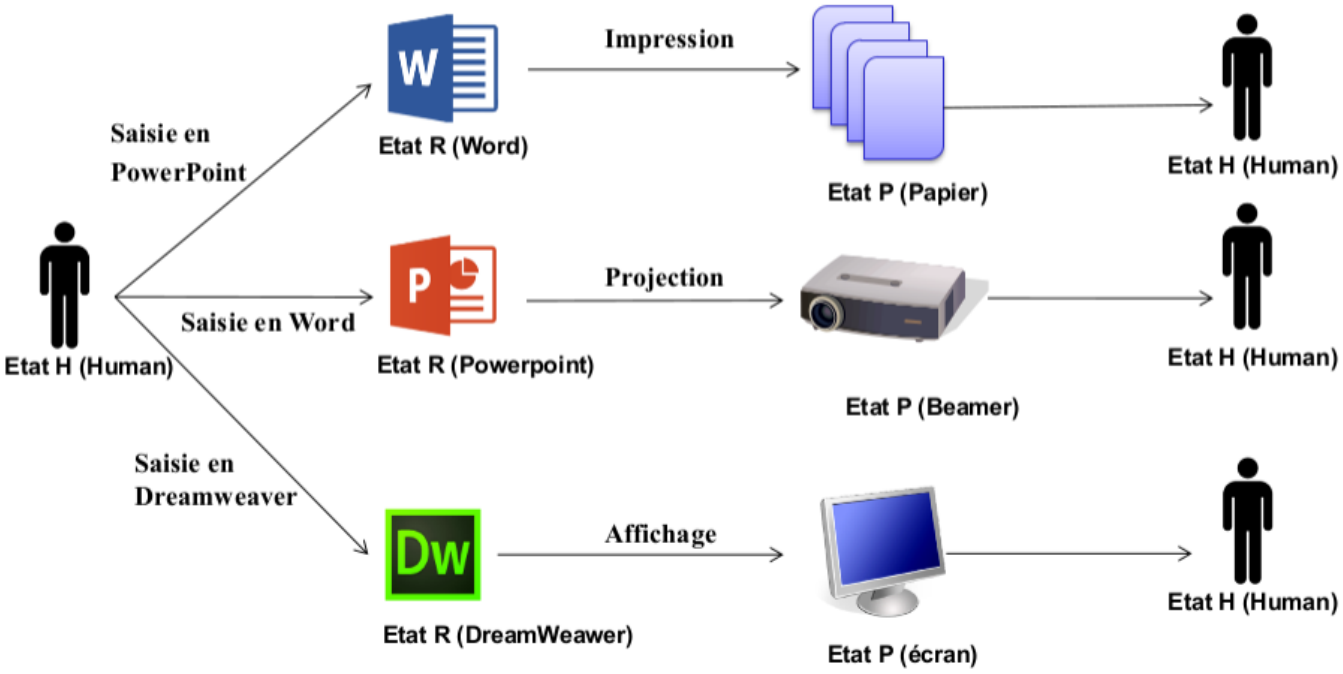
Un XML valide facilite l'échange entre les applications

Exemple de document XML

```
<?xml version="1.0" encoding="UTF-8"?>
<message status="public" xmlns="unifr">
  <subject>Question</subject>
  <sender>elena.mugellini@hefr.ch</sender>
  <receiver>rolf.ingold@unifr.ch</receiver>
  <body>
    Hello Rolf, Do you think that it is useful to speak about XML in the course?
    Would you have an example which I can show?
  </body>
  <signature>Elena</signature>
</message>
```

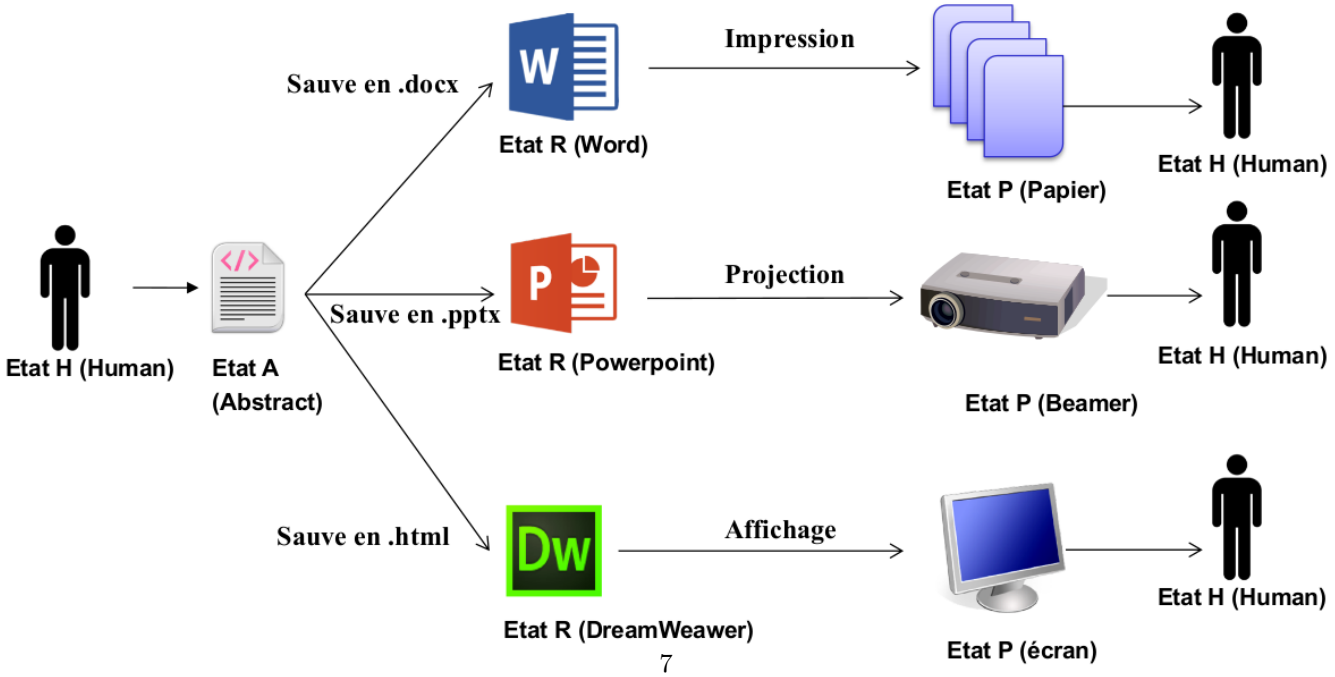
2.3 Comprendre les modèles de production des documents

HRPH : Human -> Rendering -> Presentation -> Human



(a) Modèle de production (documents linéaires)

HARPH : Human -> **Abstraction** -> Rendering -> Presentation -> Human



(b) Modèle de production (documents structuré)

FIGURE 2 – Les différents modèles de production

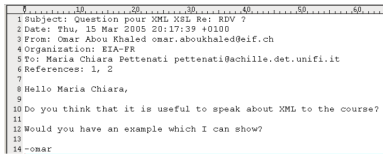
2.4 Expliquer la différence entre les documents linéaires et structurés

2.4.1 Documents linéaires

- Les documents linéaires sont modelisés par un flot de données qui mêle le contenu du document aux informations de présentation et/ou de typage
- Les formats de documents linéaires sont très proches de leurs modèles ; ils sont une simple transcription de l'information contenue dans le modèle
- Exemples : textes simples, documents formatés

- Ne contient que la composante textuelle du document
- Aucune information supplémentaire n'est représentée

- Le contenu textuel du document est entrecoupé d'indications de changement du flot de texte (RTF, Latex, HTML, PDF, DOCX, ...)



(a) Texte simple

(b) Document formaté

FIGURE 3 – Exemples de documents linéaires

Désavantages

Le style est mêlé au contenu & la réutilisation des données exiges une dupplication de l'effort de saisie.

2.4.2 Document structurés

- Fondés sur les relations logiques entre les différents composants du document plutôt que sur leur apparence et leur position dans la page
- Cette différence se répercute sur les formats de stockage qui dissocient
  - l'information de structure su document
  - des informations nécessaires à son exploitation (formatage, édition, etc...)
- Représenter les documents sous forme structurée facilite :
  - l'édition et la mise à jour du contenu
  - le contrôle de la présentation
  - le travail collaboratif
  - la recherche d'information

Le document structuré est composé de deux parties :

- Structure logique
  - Décrit la relation entre les éléments
  - Décrit le point de vue de l'auteur / du lecteur
  - Exemple : chapitres, sections, titres, figures, légendes, ...
- Structure physique
  - Décrit le rendu du document (format, style, ...)

— Décrit la mise en page (présentation)
— Contient des pages, regions, colonnes, blocs, lignes
— Les structures physique sont généralement décrites par des boîtes juxtaposées et/ou imbriquées

(a) Structure logique

(b) Structure physique

FIGURE 4 – Exemple : structure d'un message

3 XML Schema

3.1 Définir ce que sont les modèles de documents et pourquoi ils sont utilisés

Un modèle de document est une sorte de « règle à suivre ». Ils sont utiles pour pré-définir des structures de documents et s'assurer que tous les documents produits avec ce modèle respectent les mêmes règles.

On développe des modèles pour des documents
— Qui ont une longue durée de vie
— Qui seront utilisés par plusieurs utilisateurs

Il est très important d'investir du temps dans la structure de ses documents
— Moins de maintenance
— Moins d'effort de programmation

3.2 Définir DTD et expliquer son rôle

DTD : **D**ocument **T**ype **D**efinition. C'est un modèle de document

Plusieurs types de déclaration
— éléments
— attributs
— entités
— notations

Exemple de document DTD :

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
 <to>Tove</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>Don't forget me this weekend</body>
</note>
```

(a) DTD interne

```
## XML document with a reference to
## an external DTD
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
 <to>Tove</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>Don't forget me this weekend!</body>
</note>
----------------------------------------
## The DTD document
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

(b) DTD externe

FIGURE 5 – DTD interne & externe

Désavantages

- Les DTD offrent des possibilités de **typage** très limitées
- Les **contraintes** de répétition sont limitées
- Les DTD ne sont pas féfinis comme des documents XML : deux syntaxes différentes
- Les DTD ne sont pas extensibles

Les DTD sont trop orientés documents textuels.

3.3 Evaluer, juger et critiquer la différence entre DTD et XML Schema

Voir les désavantages du DTD (ci-dessus)

3.4 Pouvoir illustrer, expliquer et identifier les composants principaux d'un XML Schema

Un schéma XML fournit une description de documents XML **en utilisant également XML**

Avantages

- Plus performant : typage de données, définition précise des occurences
- Plus flexible : posibilité d'extensions et de réutilisations
- Plus complexes

Objectifs

- Définir la structure et les contenus des documents
- Fournir un ensemble de types primitifs pour les données
- Permettre l'évolutio des schémas
- Définir des descriptions et des contraintes spécifiques
- Un document XML doit pouvoir être validé relativement à son schéma

3.4.1 Les éléments, les attributs, types simples et complexes

Un élément est composé d'attributs, en général *name*, *type*, *ref* (pour les globaux), *minOccurs*, ...

Un type simple est déjà défini. Il contient « uniquement » des données.
Quelques exemples : string, anyURI, boolean, decimal, float, duration, time, date, integer, ...
Un type complex est un type créé par l'utilisateur, qui permet d'ajouter plus de contraintes, il ne peut que contenir des *compositors* (voir plus pas ce que sont les compositors).

```
<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>
```

FIGURE 6 – Exemple de type simple

```
<?xml version = "1.0"?>
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
  <xs:element name = "incomingEmail">
    <xs:complexType>
      <xs:all>
        <xs:element name = "from" type = "xs:string"/>
        <xs:element name = "subject" type = "xs:string"/>
        <xs:element name = "message" type = "xs:string" minOccurs= "0"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<?xml version = "1.0"?>
<incomingEmail>
  <subject>XML Schema workshop</subject>
  <from>john.milton@company.com</from>
</incomingEmail>
```

FIGURE 7 – Exemple de type complexe

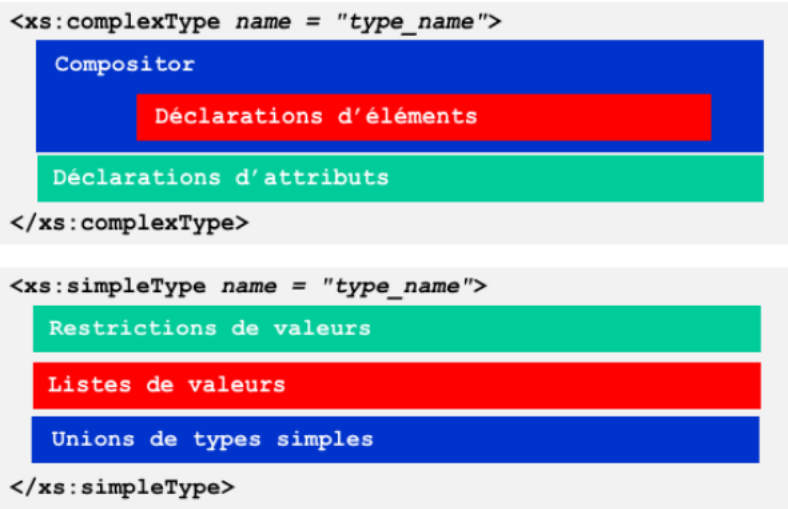
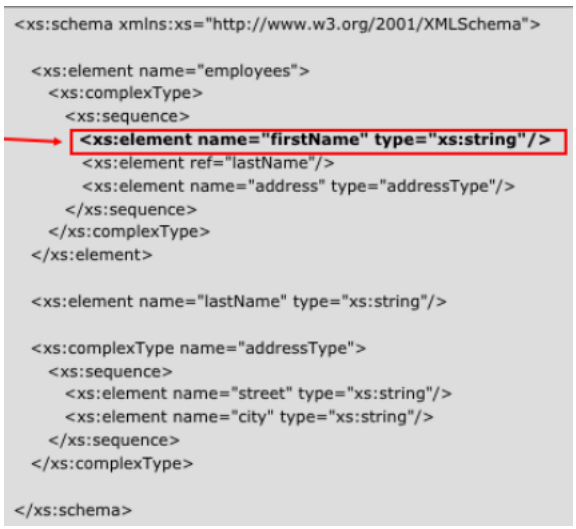


FIGURE 8 – Comparaison entre le type simple et le tpe complexe

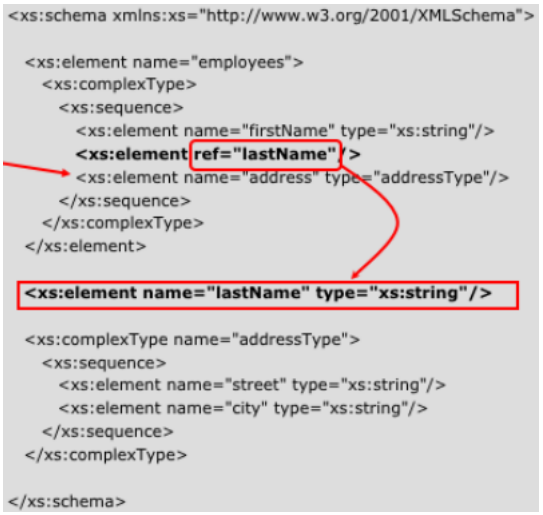
3.4.2 Composants d'aide (annotation, etx.)

3.5 Pouvoir expliquer et identifier les notions suivantes

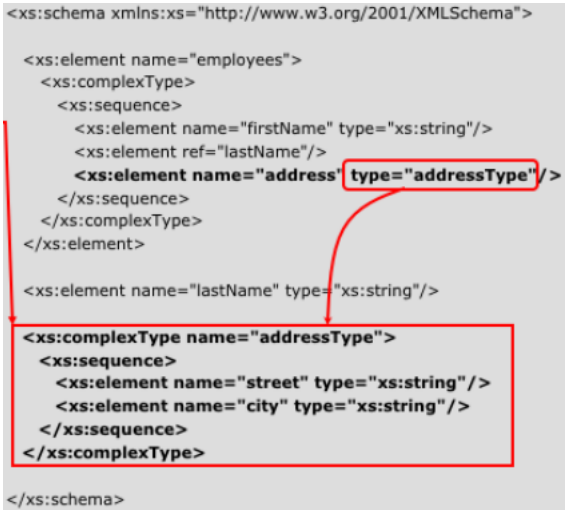
3.5.1 Déclaration d'éléments locaux et globaux



(a) Inline declaration (local)



(b) Referencing declaration (global)



(c) Named types

FIGURE 9 – Types de déclarations

3.5.2 Définition de types nommés et anonymes (Inline declaration, referencing declarations, named and anonymous types)

Inline declarations Elle est définie **localement**, au moment où on a besoin d'un élément ou d'un attribut, à l'intérieur d'un autre composant du schéma

Referencing declarations Définition de la déclaration d'une manière **globale** et ensuite référencer cette déclaration depuis une autre définition

Named types C'est la méthode de nommage du type de définition pour qu'il soit référencé ensuite dans le document. Déclaration une fois et utilisation plusieurs fois dans le document

Définition locale vs globale

Anonymous type Le type est défini de manière locale. De ce fait, il ne peut être utilisé que pour l'élément dans lequel il a été défini

Named type Le type est défini de manière golbal. De ce fait il peut être référencé depuis plusieurs autres endroits dans le document

(a) local (b) global

FIGURE 10 – Définition locale vs globale

3.5.3 Dérivation par restriction et par extension

Types simples

Restriction de types simples :
- Longueur
- Enumération de valeur
- Valeur min et max
- Valeurs décimales
- Pattern
- Liste
- Union (pour « fusionner » deux restrictions)

Les restrictions sont effectules sur les bases suivantes : *string*, *decimal*

```
<xs:element name="isbn">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:length value="13"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

(a) Restriction de la longueur

```
<xs:element name="size">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="S"/>
            <xs:enumeration value="M"/>
            <xs:enumeration value="L"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

(b) Enumeration

```
<xs:element name="cost">
    <xs:simpleType>
        <xs:restriction base="xs:decimal">
            <xs:minInclusive value="0"/>
            <xs:maxExclusive value="999.99"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

(c) Restriction min max

```
<xs:element name="cost">
    <xs:simpleType>
        <xs:restriction base="xs:decimal">
            <xs:totalDigits value="5"/>
            <xs:fractionDigits value="2"/>
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="999.99"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

(d) Valeurs décimales

```
<xs:element name="sku">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="[A-Z]{2}-[0-9]{3}"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Quantifier

Character class

Charact (Unicod

(e) Pattern

```
<xs:element name="sizes">
    <xs:simpleType>
        <xs:list itemType="xs:decimal"/>
    </xs:simpleType>
</xs:element>
```

(f) Liste

```
<xs:simpleType name="sizeNums">
    <xs:restriction base="xs:decimal"/>
</xs:simpleType>

<xs:simpleType name="sizeNames">
 <xs:list itemType="xs:string" ><xs:simpleType>
    <xs:restriction base="xs:string">
        <xs:enumeration value="XL"/>
        <xs:enumeration value="L"/>
        <xs:enumeration value="M"/>
        <xs:enumeration value="S"/>
    </xs:restriction>
</xs:list>
</xs:simpleType>

<xs:element name="sizes">
    <xs:simpleType>
        <xs:union memberTypes="sizeNames sizeNums"/>
    </xs:simpleType>
</xs:element>
```

(g) Union

FIGURE 11 – Dérivations de type simple

Types complexes

Les types complexes peuvent être dérivés par

extension d'un type simple ou complexe existant

restriction d'un autre type complexe

```
<xs:schema xmlns:xs="…">
<xs:complexType name="Address">
    <xs:sequence>
        <xs:element name="Street" type="xs:string"/>
        <xs:element name="City" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<xs:element name="CompleteAddress">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="Address">
                <xs:sequence>
                    <xs:element name="PostCode" type="xs:integer"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
</xs:schema>
```

FIGURE 12 – Exemple de dérivation par extension

```
<xs:schema xmlns:xs="…">
<xs:complexType name="Address">
   <xs:sequence>
      <xs:element name="Street" type="xs:string" minOccurs="1" maxOccurs="10"/>
      <xs:element name="City" type="xs:string"/>
   </xs:sequence>
</xs:complexType>

<xs:element name="ShortAddress">
    <xs:complexType>
        <xs:complexContent>
            <xs:restriction base="Address">
                <xs:sequence>
                    <xs:element name="Street" type="xs:string" minOccurs="1" maxOccurs="2"/>
                    <xs:element name="City" type="xs:string"/>
                </xs:sequence>
            </xs:restriction>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
</xs:schema>
```

FIGURE 13 – Exemple de dérivation par restriction

3.6 Maîtriser l'utilisation et justifier les avantages et les désavantages des principes suivants

3.6.1 Russian Doll, Salami Slice, Venetian Blind, Garden of Eden

Russian Doll (poupées russes). Les éléments sont dans les éléments qui sont dans les éléments... Un seul élément global

Salami Slice Tous les éléments sont globaux (difficile de reconnaître la racine). Réutilisable. On utilise que des référence vers les éléments, au sein de la racine

Venetian Blind Extension du Russian Doll. Contient un seul élément global. Tous les autres élément sont locaux. En revanche, les types complexes et les groupes sont globaux

Garden of Eden Combinaison du Venetian Blind et du Salami Slice. Les éléments et les types complexes sont globaux

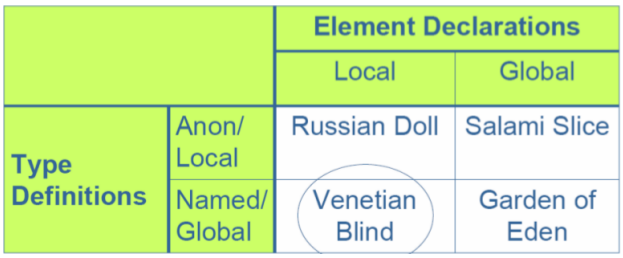


FIGURE 14 – Resumé des modèles de schéma

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://schemas.sun.com/point/russiandoll"
  xmlns:tns="http://schemas.sun.com/point/russiandoll"
  elementFormDefault="qualified">

  <xsd:element name="Line">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="PointA">
          <xsd:complexType>
            <xsd:attribute name="x" type="xsd:integer"/>
            <xsd:attribute name="y" type="xsd:integer"/>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="PointB">
          <xsd:complexType>
            <xsd:attribute name="x" type="xsd:integer"/>
            <xsd:attribute name="y" type="xsd:integer"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

(a) Russian Doll

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://schemas.sun.com/point/salami"
    xmlns:tns="http://schemas.sun.com/point/salami"
    xmlns="http://schemas.sun.com/point/salami"
    elementFormDefault="qualified">

    <xsd:element name="PointA">
        <xsd:complexType>
            <xsd:attribute name="x" type="xsd:integer"/>
            <xsd:attribute name="y" type="xsd:integer"/>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="PointB">
        <xsd:complexType>
            <xsd:attribute name="x" type="xsd:integer"/>
            <xsd:attribute name="y" type="xsd:integer"/>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="Line">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="PointA"/>
                <xsd:element ref="PointB"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

(b) Salami Slice

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://schemas.sun.com/point/venetianblind"
    xmlns:tns="http://schemas.sun.com/point/venetianblind"
    xmlns="http://schemas.sun.com/point/venetianblind"
    elementFormDefault="qualified">

    <xsd:complexType name="PointType">
        <xsd:attribute name="x" type="xsd:integer"/>
        <xsd:attribute name="y" type="xsd:integer"/>
    </xsd:complexType>

    <xsd:element name="Line">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="PointA" type="PointType"/>
                <xsd:element name="PointB" type="PointType"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

(c) Venetian Blind

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://schemas.sun.com/point/gardenofeden"
    xmlns="http://schemas.sun.com/point/gardenofeden"
    elementFormDefault="qualified">

    <xsd:complexType name="PointType">
        <xsd:attribute name="x" type="xsd:integer"/>
        <xsd:attribute name="y" type="xsd:integer"/>
    </xsd:complexType>

    <xsd:complexType name="LineType">
        <xsd:sequence>
            <xsd:element ref="PointA"/>
            <xsd:element ref="PointB"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="PointA" type="PointType"/>

    <xsd:element name="PointB" type="PointType"/>

    <xsd:element name="Line" type="LineType"/>
</xsd:schema>
```

(d) Garden of Eden

FIGURE 15 – Les 4 « design patterns »

3.6.2 Local élément vs Global élément

Les élément locaux sont dans la racine. Les éléments globaux y sont « à côté ». Les éléments locaux ont l'avantage de donner au document la même forme que le document XML devant être validé. Les éléments globaux sont réutilisables (attribut *ref*).

3.6.3 Types nommés vs types anonymes (Named vs. Anonymous Types)

Pareil pour les types. En leur donnant un nom il deviennent réutilisables (attribut *type*)

3.7 Exposer, expliquer les notions suivantes

3.7.1 Compositors (sequence, any, choice), groupe (d'éléments et d'attributs) et les modèles de contenu (mixed, empty, etc...)

Compositors

- séquences `sequence` tous les éléments doivent apparaître dans l'ordre
- choix `choice` défini un choix exclusif (comme un radiobutton)
- « Tous » `all` permet aux éléments d'apparaître dans n'importe quel ordre (si *minOccurs* et *maxOccurs* ne sont pas précisés ils doivent apparaître chacun une fois, peu importe l'ordre)

- Il est possible de combiner sequence et choice, mais al ne peut être combiné avec aucun autre compositors.

Groupes

Les groupes ne peuvent pas être étendus ou restreints (à l'inverse des types complexes). Quand un groupe est référencé, c'est comme si son contenu avait été copié là où il est référencé.

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

 <xs:group name = "NameGroup">
  <xs:sequence>
   <xs:element name = "FirstName" type = "xs:string"/>
   <xs:element name = "LastName" type = "xs:string"/>
  </xs:sequence>
 </xs:group>

 <xs:element name = "Customers">
  <xs:complexType>
   <xs:group ref="NameGroup" minOccurs = "0" maxOccurs = "unbounded"/>
  </xs:complexType>
 </xs:element>

</xs:schema>
```

FIGURE 16 – Exemple de groupe d'éléments

On peut faire pareil avec les attributs

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

 <xs:attributeGroup name = "custLengthGroup">
  <xs:attribute name = "noCustomers" type = "xs:integer"/>
  <xs:attribute name = "Customers" type = "xs:integer"/>
 </xs:attributeGroup>

 <xs:group name = "NameGroup">
  <xs:sequence>
   <xs:element name = "FirstName" type = "xs:string"/>
   <xs:element name = "LastName" type = "xs:string"/>
  </xs:sequence>
 </xs:group>

 <xs:element name = "Customers">
  <xs:complexType>
   <xs:group ref = "NameGroup" minOccurs = "0" maxOccurs = "unbounded"/>
   <xs:attributeGroup ref = "custLengthGroup"/>
  </xs:complexType>
 </xs:element>

</xs:schema>
```

FIGURE 17 – Exemple de groupe d'atributs

Modèles de contenu (complexType)

text-only content contient seulement du texte (un élément de type string)

element-only content contient seulement des éléments (dans une séquence par exemple)

mixed content contient du texte et des éléments fils *mixed=true* (comme attribut d'un type complexe)

empty content ne contient rien *<xs :complexType />* (dans un élément)

any Un élément pouvant avoir n'importe quel type de contenu

3.7.2 Namesapce, targetNamespace, qualified, unqualified, elementFormDefault, etc...

Namespace

Namespace permet aux concepteurs de qualifier chaque schéma dans le but de les distinguer clairement entre eux. Il résolvent aussi les conflits de noms entre des éléments de la même instance de document.
Il est utilisé pour identifier l'origine (implicitement sa sémantique) d'un élément

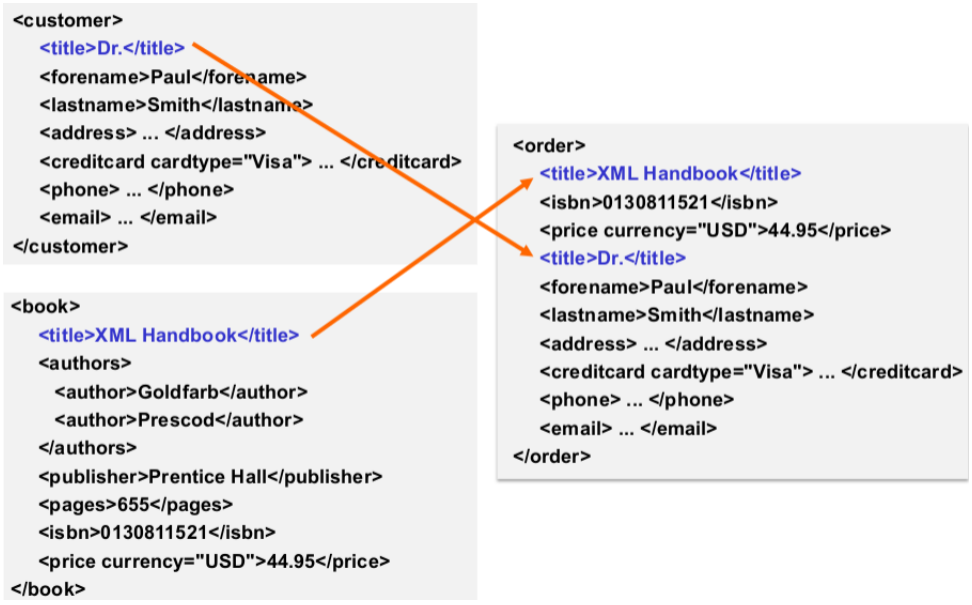


FIGURE 18 – Exemple de conflits

FIGURE 19 – Déclaration des espaces de noms

Déclaration des espaces de noms

- Eléments avec le même nom mais avec différets namespaces, peuvent être distingués
- Tous les éléments d'un document qui sont du même namespace peuvent être identifiés et on peut déterminer leur origine

Types d'attributs associés aux namespaces

xmlns `xmlns:xs="http://www.w3.org/2001/XMLSchema"` indique que les éléments et les types utilisés dans le schéma viennent de `http://www.w3.org/2001/XMLSchema`. Ils spécifient aussi que les éléments et les types qui viennent de `http://www.w3.org/2001/XMLSchema` doivent avoir le préfixe `xs:`. Si il n'est pas accompagné du *xs*, il indique que le namespace **par défaut** est celui ente guilemets

xs :schemaLocation Une fois que le namespace est disponible, on peut utiliser cet attribut. Il a deux valeurs : la première est le namespace et la deuxième le nom de l'XML Schema : `xsi:schemaLocation="https://www.w3schools.co`

xs :noNamespaceSchemaLocation référence un XML Schema qui n'a pas de target namespace

defaultNamespace ? namespace par défaut ?

targetNamespace `targetNamespace="https://www.w3schools.com"` indique que les éléments sont définis par ce schéma viennent de `https://www.w3schools.com`

elementFormDefault
- unqualified
  - les éléments déclarés globalement doivent être qualifiés
  - les éléments déclarés localement ne doivent pas être qualifiés
- qualified
  - les éléments déclarés (localement et globalement) doivent être qualifiés
  - On peut définir un namespace par défaut pour les éléments déclarés localement

attributeFormDefault
- qualified
  - tous les attributs doivent être qualifiés explicitement
  - pas de namespace par défaut possible
- unqualified
  - seulement les attributs déclarés globalement doivent être qualifiés explicitement

Il est recommandé d'utiliser *elementFormDefault="qualified"* et *attributeFormDefault="unqualified"*

Etre qualifié signifie être précédé par le préfixe correspondant au namespace (je crois, pas sûr).

3.8 Illustrer les notions suivantes et savoir quand les utiliser

Comment combiner plusieurs XML Schema ? De ces trois manières différentes :

3.8.1 import

Quand les deux schémas ont des espaces de noms cibles (targetNamespace) différents

3.8.2 include

Quand les deux schémas ont le même espace de noms cibles

3.8.3 redefine

Pour redéfinir un schéma

```
<import
  namespace="URI"
  schemaLocation="URI">
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:contact="http://www.myserver.com/Contact"
    targetNamespace="http://www.myserver.com/CustomerRecords"
    xmlns="http://www.myserver.com/CustomerRecords">

<import namespace="http://www.myserver.com/Contact"
    schemaLocation="http://www.myserver.com/contact.xsd"/>
```

Déclaration global

(a) import

Product.xsd

```
<?xml version="1.0"?>
<xsd:schema
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.company.org"
      xmlns="http://www.product.org"
      elementFormDefault="qualified">
  <xsd:complexType name="ProductType">
    <xsd:sequence>
      <xsd:element name="Type" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Person.xsd

```
<?xml version="1.0"?>
<xsd:schema
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.company.org"
      xmlns="http://www.person.org"
      elementFormDefault="qualified">
  <xsd:complexType name="PersonType">
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string"/>
      <xsd:element name="SSN" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

```
<include
  schemaLocation="URI">
```

```
<?xml version="1.0"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.company.org"
    xmlns="http://www.company.org"
    elementFormDefault="qualified">
  <xsd:include schemaLocation="Person.xsd"/>
  <xsd:include schemaLocation="Product.xsd"/>
  <xsd:element name="Company">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Person" type="PersonType"
              maxOccurs="unbounded"/>
        <xsd:element name="Product" type="ProductType"
              maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

d'Information I 39

(b) include

Myschema1.xsd:

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

<xs:complexType name="pname">
  <xs:sequence>
    <xs:element name="firstname"/>
    <xs:element name="lastname"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="customer" type="pname"/>

</xs:schema>
```

Myschema2.xsd:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSch
ema">

<xs:redefine schemaLocation="Myschema1.xsd">
  <xs:complexType name="pname">
    <xs:complexContent>
      <xs:extension base="pname">
        <xs:sequence>
          <xs:element name="country"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:redefine>

<xs:element name="author" type="pname"/>

</xs:schema>
```

(c) redefine

FIGURE 20 – Syntaxe : import, include, redefine