62-31.3 XML - Complément

Types de données - Types prédéfinis

Objectif : ce support reprend les types de données prédéfinis qui ont été abordés en classe dans la présentation XML-02-02-Types_Predefinis_XSD, en y ajoutant parfois des descriptions et des exemples complémentaires. A la fin du support un exemple d'adaptation de schéma XML (XSD) est montré avec quelques conseils sur l'utilisation de ces différents types de données prédéfinis.

Auteur : Cédric Benoit

Date : 20.09.2023

Table des matières

1 Introduction

Dans ce support, il est abordé de manière plus détaillée les différents types de données prédéfinis de la recommandation W3C du XML Schema Definition (XSD) version 1.0 qui ont été présentés en classe dans la présentation XML-02-02-Types_bases_XSD. Dans une présentation, il n'est pas possible de tout montrer et le but de ce support est simplement d'apporter quelques compléments d'informations et d'autres exemples, avec un rappel des principales caractéristiques de chaque type prédéfinies. A la fin du support une synthèse et un exemple d'adaptation d'un schéma XML au format XSD sont montrés, accompagnés de quelques conseils.
Pour avoir encore des informations plus théoriques et détaillées sur chacun des types de données prédéfinis, vous êtes aussi encouragés, en cas de besoin, à consulter aussi la recommandation du W3C. L'objectif d'aborder ces différents types de données est de montrer l'importance du typage des données dans le cadre de la validation d'un document XML. Comme dans d'autres domaines, plus la propreté des données est garantie au travers du typage (avoir des données de qualité), plus il sera facile de les traiter (calcul, tri, etc.) et d'avoir des résultats cohérents et pertinents.

En plus de la présentation XML-02-02-Types_Predefins_XSD, le contenu de ce support s'appuie aussi sur d'autres ressources, notamment :

- La recommandation W3C pour XML Schema Edition (XSD 1.0)[1] ;
- "Chapter 4. Using Predefined Simple Datatypes" du livre "XML Schema" de "Eric van der Vlist" disponible en ligne sur la plateforme de O'Reilly[2] ;
- Le livre "XML en concentré", 3ème édition, édité par O'REILLY[3] ;
- Anciens supports de cours de l'enseignant précédent (Claudio Cortinovis) ;
- L'expérience de l'auteur dans ce domaine.

Le contenu de ce support s'appuie sur la recommandation XSD 1.0 qui est la version de base. Nous verrons par la suite qu'il y a une version plus récente de la recommandation W3C qui apporte quelques évolutions complémentaires au typage de données prédéfinies.

1.1 Rappel

Dans le cours 62-31.3 XML, nous avons étudié, comment créer un document XML et comment définir un schéma XML au format XSD permettant de valider la structure du document XML, ainsi que dans quels types de noeuds (attribut ou élément) les données peuvent être définies. Nous avons vu également comment associer un schéma XML (XSD) à un document XML à des fins de validation.
A ce stade, nous n'avons pas encore abordé la problématique du typage de données. Par exemple, s'il y a un élément *dateNaissance* dans le document XML, il faut que nous puissions saisir une date correcte et non un chiffre ou une chaîne de caractères quelconque. Pour répondre à cette problématique, la recommandation W3C pour XML a défini des types de données prédéfinis qui vont être décrits dans ce document.

1.2 Importance du typage des données

Comme en base de données ou en programmation, il est important aussi en XML d'avoir des données typées correctement. Cela évite d'avoir des erreurs de saisie ou de traitement et d'avoir aussi des résultats erronés. La recommandation W3C a défini un certain nombre de types de données prédéfinis qui sont hiérarchisés comme le montre l'illustration ci-dessous.

[1] XML Schema Part 2: Datatypes Second Edition: https://www.w3.org/TR/xmlschema-2/#datatype
[2] Chapter 4. Using Predefined Simple Datatypes: https://www.oreilly.com/library/view/xml-schema/0596002521/ch04.html
[3] Version partielle du livre en ligne dans google books: https://books.google.ch

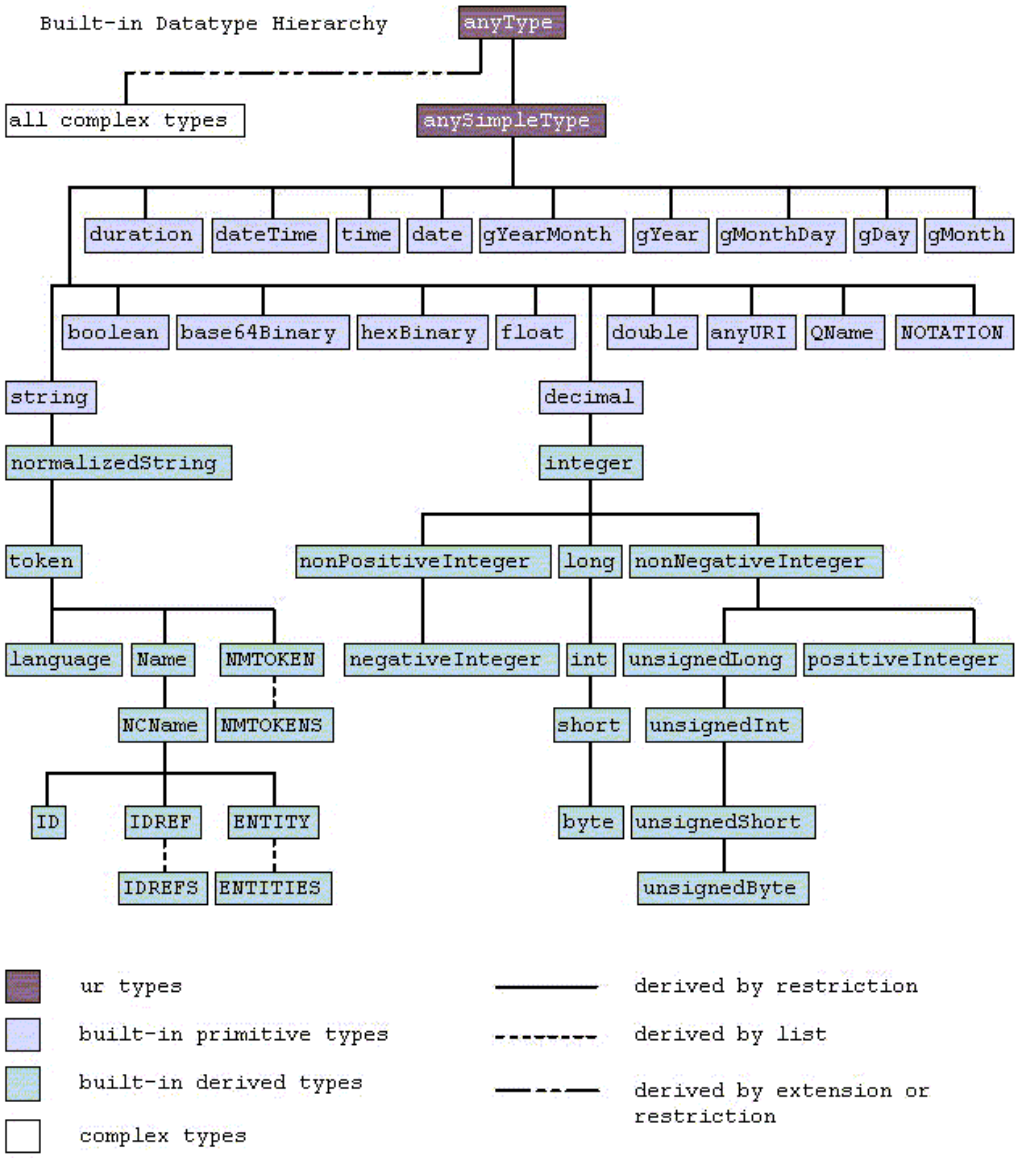

Figure 1 - Illustration XML hiérarchique des types de données prédéfinis version 1.0 du W3C[4]

Ce qui est intéressant est que ce schéma représente non seulement la hiérarchie des types, mais il permet également de distinguer les différents types, comme les types de bases (anyType, …), les types primitifs (string, …) et les types dérivés (normalizedString, …) et leur lien entre eux. Dans le cadre de ce support, ce sont les types de données prédéfinis destinés aux données simples qui vont être abordés.

2 Types prédéfinis pour les données simples

Selon la recommandation W3C, il y a un certain nombre de types de donnée prédéfinis pour les données simples, c'est-à-dire qui contiennent des valeurs comme du texte, des chiffres ou des dates par exemple. L'avantage d'avoir ces types de données est la possibilité de vérifier que les valeurs soient correctement formatées et valides. Pour les types simples (descendants de anySimpleType) prédéfinis, nous pouvons distinguer deux catégories qui sont :

- Types primitifs :
  - Types textes (string, anyURI, QName,…) : qui sont des types de données permettant de formater les valeurs d'une chaîne de caractères.

[4] Built-in datatypes: https://www.w3.org/TR/xmlschema-2/#datatype

  - Types numériques (decimal, float, double, etc.) : qui sont des types de données permettant de formater les nombres décimaux, des nombres réels ou des valeurs logiques.
  - Types temporels (dateTime, date, time, etc.): qui sont des types de données permettant de formater les valeurs du temps, comme les dates et les heures.
  - Autres (hexBinary,…) : qui sont des types de données permettant de stocker les valeurs sous différents formats (hexadécimal, base 64, etc.).

- Types dérivés par restriction ou par liste :
  - Types textes (normalizedString, token,…) : qui sont des types permettant de formater les valeurs textuelles avec différentes restrictions.
  - Types numériques (integer, long,…) : qui sont des types permettant de formater les nombres entiers avec différentes restrictions.
  - Types listes (IDREFS, NMTOKENS ou ENTITIES) : qui sont des types permettant de formater des listes de valeurs.

L'illustration ci-dessous montre graphiquement la hiérarchie des types simples (descendants de anySimpleType) prédéfinis, selon la recommandation W3C pour XML.

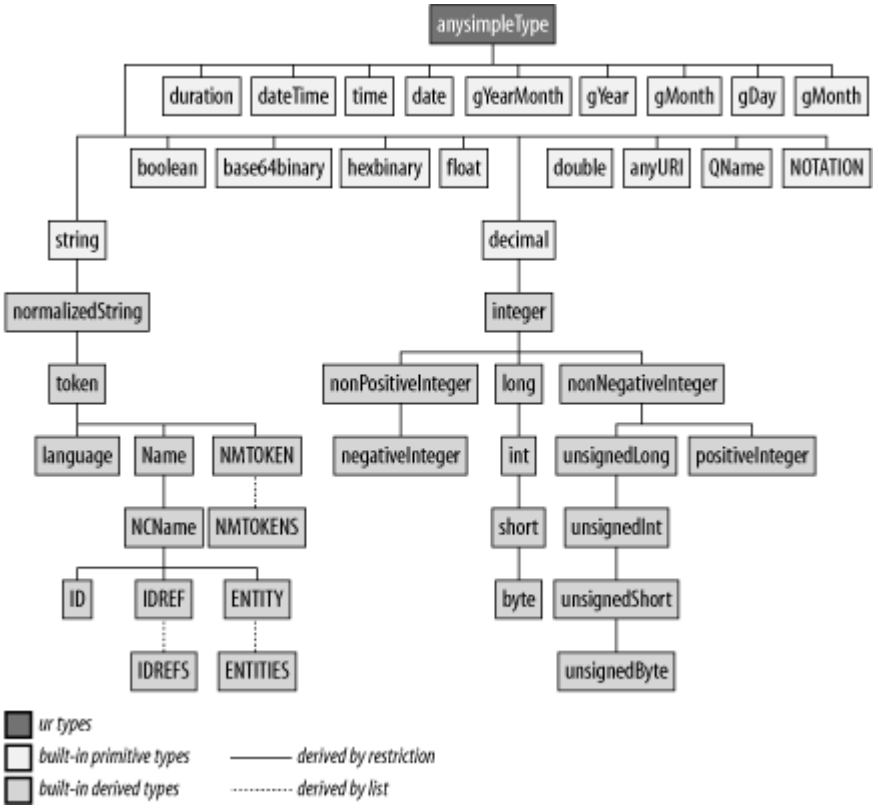

Figure 2 - Illustration XML hiérarchique des types de données simples prédéfinis version 1.0 du W3C[5]

Le type *anysimpleType* est un type qui accepte n'importe quelle valeur sans restriction et de ce fait, il est fortement déconseillé de l'utiliser, sauf dans les cas d'exception. Ce type est le type de base pour tous les types simples (primitifs et dérivés). La définition de ce type (appelé ur-type en anglais) est unique et ne peut pas être dérivée.
Dans l'illustration montrée ci-dessus, il y a également les types de données primitifs, comme *string* ou *decimal*, qui sont à la base de tous les types dérivés, comme *normalizedString* ou *integer*. Les types de données dérivés sont définis à partir d'autres types de données qui peuvent être des types primitifs ou également des types de données dérivés.

[5] Map of predefined datatypes: https://www.oreilly.com/library/view/xml-schema/0596002521/ch04.html

Comme le montre l'illustration plus haut, les types prédéfinis sont hiérarchisés et sont redéfinis par restriction. Cela signifie que plus on descend dans la hiérarchie, plus le type prédéfini sera contraignant. Chaque type dérivé reprend les mêmes caractéristiques que le type hiérarchiquement supérieur en y ajoutant des restrictions supplémentaires. Par exemple, le type *decimal* permet de saisir un nombre avec des décimales, alors que le type *integer* permet de saisir uniquement un nombre sans décimale.

Dans les chapitres qui vont suivre dans ce document, il est décrit de manière plus détaillée ces différents types prédéfinis primitifs et dérivés.

3 Types textes

Ce chapitre décrit les types de données dérivés du type de données primitif *string*, ainsi que d'autres types de données qui ont un comportement similaire, comme *hexBinary* ou *anyURI*. La hiérarchie de ces types de données est montrée dans l'illustration ci-dessous.

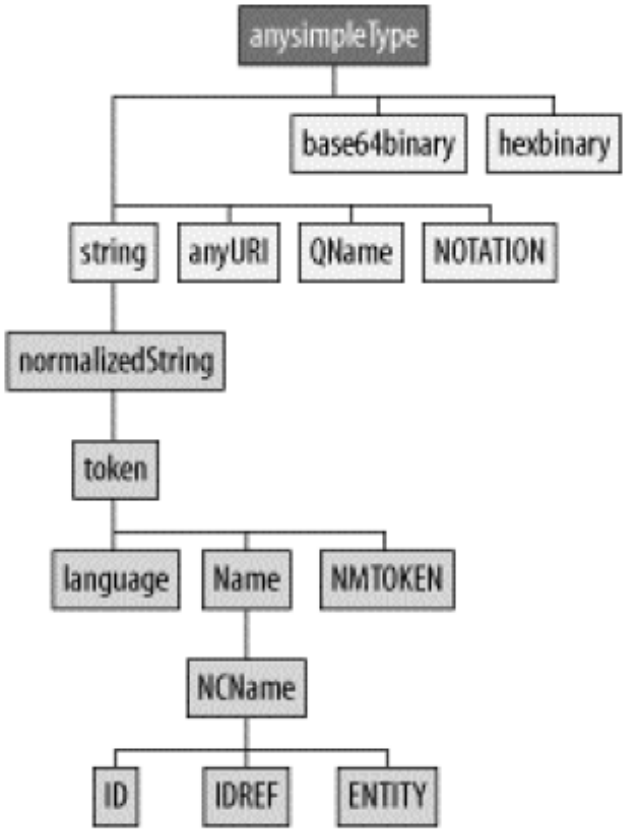

Figure 3 – Illustration hiérarchie des types textes et similaires[6]

3.1 string

Le type de données *string* est un type primitif et il peut contenir tout texte y compris tabulation, retour chariot (carriage return), espaces multiples, saut de ligne, etc.

Exemple de valeurs conformes :
"Ce matin il fait beau
et ce soir il va pleuvoir" (avec retour chariot),
"Pêche Pomme Poire" (avec tabulation), etc.

[6] Strings and similar datatypes: https://www.oreilly.com/library/view/xml-schema/0596002521/ch04.html

3.2 normalizedString

Le type de données *normalizedString* est un type dérivé de *string* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Pour être conforme, il ne doit pas contenir les caractères suivants :

- retour chariot (carriage return) (#xD) ;
- saut de ligne (line feed) (#xA) ;
- tabulation (tab) (#x9).

Ce type de données est appelé aussi chaîne de caractères normalisée.

Exemple de valeurs conformes :
"Ce matin il fait beau et ce soir il va pleuvoir",
" Ce matin il fait beau et ce soir il va pleuvoir " (espaces multiples), etc.

Exemple de valeurs non conformes :
"Ce matin il fait beau	et ce soir il va pleuvoir" (tabulation),
"Ce matin il fait beau
et ce soir il va pleuvoir" (retour chariot), etc.

3.3 token

Le type de données *token* est un type dérivé de *normalizedString* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Pour être conforme, il ne doit pas contenir les caractères suivants :

- Pas d'espaces multiples, pas d'espaces avant le début et la fin du texte

Exemple de valeurs conformes :
"Ce matin il fait beau et ce soir il va pleuvoir",
"Pêche Pomme Poire", etc.

Exemple de valeurs non conformes :
" Ce matin il fait beau et ce soir il va pleuvoir " (pas d'espaces avant "Ce matin…" et après "…pleuvoir"),
"Pêche Pomme Poire" (espaces multiples entre Pomme et Poire par exemple), etc.

3.4 language

Le type de données *language* est un type dérivé de *token* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Il permet de stocker des codes de langues standardisés selon le RFC 4646[7] et 4647[8] (qui remplace RFC 3066 et 1766). En plus du code langue ("en","fr","de", etc.) il est possible de spécifier le code du pays ("US", "FR", etc.).

Exemple de valeurs conformes :
"en": langue anglaise,
"fr": langue française,
"fr-CH": langue française utilisée en Suisse,
etc.

Ce type a un impact significatif sur le système d'information selon son utilisation. Dans le cadre de la gestion de données, nous prenons l'exemple des noms des villes en Suisse. Par exemple la ville de Neuchâtel en français se traduit en allemand par Neuenburg. En XML, pour avoir ces deux informations à disposition, nous pourrions avoir un élément *ville* avec un attribut *langue* de type *language*. Ce qui donne l'exemple ci-dessous.
…

[7] Information on RFC 4646: https://www.rfc-editor.org/info/rfc4646
[8] Information on RFC 4647: https://www.rfc-editor.org/info/rfc4647

<ville langue="fr">Neuchâtel</ville>
<ville langue="de">Neuenburg</ville>
…

Comme vous avez pu l'étudier également en HTML, l'importance de la langue joue un rôle important dans le domaine pages HTML.

3.5 NMTOKEN

Le type de données *NMTOKEN* est un type dérivé de *token* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Il correspond aux Nmtoken de XML 1.0 et la valeur correspond à une seule chaîne de caractères selon le type token, mais sans espace.

Exemple de valeurs conformes :
"Monsieur", "Madame" ou "001", "002", etc.

Exemple de valeurs non conformes :
"Monsieur Madame" (espace interdit), "001,002" (virgule interdite), etc.

3.6 Name

Le type de données *Name* est un type dérivé de *token* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Il correspond aux Name de XML 1.0 et la valeur doit commencer par une lettre, ":" ou "–".

Exemple de valeurs conformes :
"-oui", ":non", "oui", etc.

Exemple de valeurs non conformes :
"001" (commence par un chiffre), "oui,non" (virgule interdite), etc.

3.7 NCName

Le type de données *NCName* (Noncolonized Name) est un type dérivé de *Name* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Il correspond aux Namespaces de XML 1.0. Par exemple dans un document XML, le préfixe "xs" de xs:string est de type *NCName.*

Exemple de valeurs conformes :
"-oui", "oui", etc.

Exemple de valeurs non conformes :
"001" (commence par un chiffre) ou ":non" (commence par deux points), etc.

3.8 ID

Le type de données *ID* est un type dérivé de *NCName* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Il correspond à l'attribut ID de XML 1.0 et la valeur doit être unique dans le document XML.

Exemple de valeurs conformes :
"C01", "PROD238", etc.

Exemple de valeurs non conformes :
"001" ou "12132" (commence par un chiffre), etc.

3.9 IDREF

Le type de données *IDREF* est un type dérivé de *NCName* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Il correspond à l'attribut IDREF de XML 1.0 et la valeur doit correspondre à une valeur ID dans le même document XML. En comparaison avec une base de données relationnelle, cela correspond au comportement de la clé étrangère.

Exemple de valeurs conformes :
"C01", "PROD238", etc.

Exemple de valeurs non conformes :
"001" ou "12132" (commence par un chiffre), etc.

3.10 ENTITY

Le type de données *ENTITY* est un type dérivé de *NCName* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Ce type de données est utilisé à des fins de compatibilité avec un DTD (Document Type Definition) qui est le prédécesseur du XSD (XML Schema Definition). Il doit correspondre à une entité définie dans un DTD.
Ce type est spécifique pour une utilisation particulière.

Exemple :
https://www.herongyang.com/XSD/string-Datatypes-ENTITY-Values-Representations.html

3.11 QName

Le type de données *QName* (Qualified Name) est un type primitif et sa valeur est une chaîne de caractères. Il correspond au nom qualifié de XML qui représente un ensemble de tuples. La valeur de ce type doit respecter le nommage XML et la contrainte est que cette valeur ne peut contenir qu'une seule fois ":" et pas au début de la chaîne de caractères. Cet ensemble de tuples peut être illustré de la manière suivante :

{namespace name (préfixe), local part (nom) }
{xs:attribute, xs:element, xs:string, xs:language,…}

Le préfixe "xs" est associé au namespace (anyURI) qui est défini au début du document XML (fichier XSD dans le cas présent). Le local part "attribute" est de type NCName.
En résumé, "xs"+":"+"attribute" correspond au type QName. Ce qui signifie que ce type a été pensé pour le XML.

3.12 anyURI

Le type de données *anyURI* est un type primitif et sa valeur est une chaîne de caractères. Cette valeur représente un URI (Uniform Resource Identifier) selon le RFC 2396 et 2732.

Exemple de valeurs conformes :
"https://www.google.com/", etc..

Exemple de valeurs non conformes :
"Jean@dupond:http" (ce n'est pas un URI correct), etc.

3.13 NOTATION

Le type de données *NOTATION* est un type primitif et sa valeur est une chaîne de caractères. Il a été créé pour implémenter les notations de XML 1.0. Ce type fournit un mécanisme permettant à un parseur XML de localiser des programmes externes ou des instructions de traitement.
Ce type est spécifique pour une utilisation particulière.

Exemple :
https://www.oreilly.com/library/view/xml-schema/0596002521/re39.html

3.14 hexBinary

Le type de données *hexBinary* est un type primitif et il permet de coder du contenu (texte, date, etc.) en hexadécimal. Chaque caractère est codé sur deux caractères au format hexadécimal. Le format XML ne permet pas de contenir du contenu binaire et les types de données *hexBinary* et *base64Binary* permettent d'encoder non seulement du contenu binaire, mais également les formats de texte dont le contenu peut être incompatible avec le balisage XML.

Exemple de valeur textuel :
"<?xml version="1.0" encoding="UTF-8"?>", etc.

Exemple de valeurs au format hexBinary :
"3f3c6d78206c657673726f693d6e3122302e20226e656f636964676e223d54552d4622383e3f", etc.

3.15 Base64Binary

Le type de données *base64Binary* est un type de données primitif et il permet de coder du texte selon le format "base 64" décrit dans le RFC 2045. Chaque caractère est codé en paquets de 6 bits par caractère.

Exemple de valeur textuel :
"<?xml version="1.0" encoding="UTF-8"?>", etc.

Exemple de valeurs au format base64Binary :
"PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4NCg==", etc.

4 Types numériques

Dans ce chapitre est décrit les types de données numériques qui sont construits sur la base de trois types primitifs qui sont :

- *decimal*: qui concerne la famille des nombres décimaux et entiers;
- *float* et *double*: qui concerne la famille des nombres réels.

Pour être conformes, les valeurs de ces quatre types doivent être des chiffres sans espace.
La hiérarchie entre ces types de données et leur descendant est montrée dans l'illustration ci-dessous.

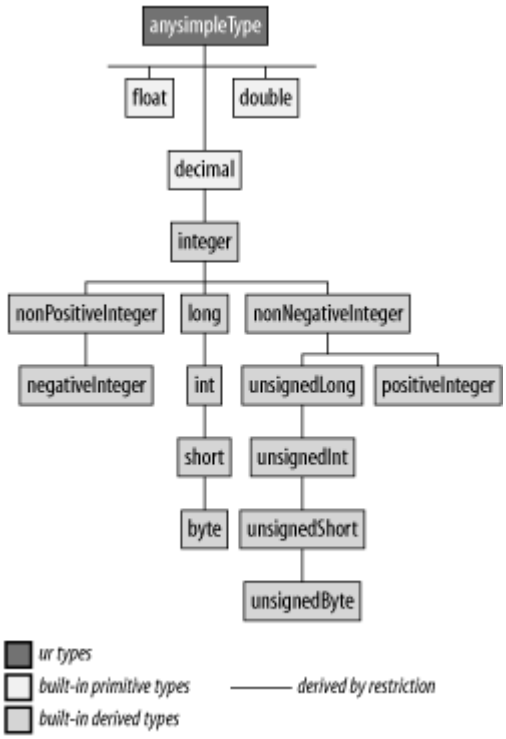

Figure 4 - Illustration hiérarchique pour les types de données numériques[9]

4.1 decimal

Le type de données *decimal* est un type primitif et il permet de saisir des nombres décimaux (base 10). Le point "." est le séparateur entre la partie entière et la partie fractionnaire. Les chiffres de 0 à 9 sont autorisés, ainsi que le signe "+" et "-" devant le nombre. Tout autre caractère (espace, notation scientifique, etc.) est interdit et la taille du nombre n'est pas limitée par le type.

Exemple de valeurs conformes :
"25.99", "+25.99", "-25.99", "25", "-.99", etc.

Exemple de valeurs non conformes :
"2.55E+2" (notation scientifique E+2 interdit),
"1'25.99" (séparateur pour les milliers interdit),
"1 25.99" (espace interdit), etc.

4.2 integer

Le type de données *integer* est dérivé du type de données *decimal* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Sa valeur n'a pas de partie fractionnaire et la taille du nombre n'est pas limitée par le type.

Exemple de valeurs conformes :
"25", "+25", "-25", "00025", etc.

Exemple de valeurs non conformes :
"1'25" (séparateur pour les milliers interdit),
"25.99" (séparateur pour les fractions interdit),

[9] Numeric datatypes: https://www.oreilly.com/library/view/xml-schema/0596002521/ch04.html

" 25" (espace interdit), etc.

4.3 Types entiers dérivés

Par rapport au type de données *integer*, le W3C a défini quatre sous-types de données liés au nombre entier plus spécifiques qui sont décrits ci-dessous.

4.3.1 nonPositiveIntger

Le type de données *nonPositiveInteger* est dérivé du type de données *integer* et reprend les mêmes caractéristiques en y ajoutant des contraintes. La restriction de ce type de données est de contenir que des nombres entiers allant de moins l'infini au nombre "0" que l'on peut aussi représenter de la manière suivante : [-∞;0].

4.3.2 negativeInteger

Le type de données *negativeInteger* est dérivé du type de données *nonPositiveInteger* et reprend les mêmes caractéristiques en y ajoutant des contraintes. La restriction de ce type de données est de contenir que des nombres entiers allant de moins l'infini au nombre "-1" que l'on peut aussi représenter de la manière suivante : [-∞;-1].

4.3.3 nonNegativeInteger

Le type de données *nonNegativeInteger* est dérivé du type de données *integer* et reprend les mêmes caractéristiques en y ajoutant des contraintes. La restriction de ce type de données est de contenir que des nombres entiers allant du nombre "0" à l'infini que l'on peut aussi représenter de la manière suivante : [0; ∞].

4.3.4 positiveInteger

Le type de données *positiveInteger* est dérivé du type de données *nonNegativeInteger* et reprend les mêmes caractéristiques en y ajoutant des contraintes. La restriction de ce type de données est de contenir que des nombres entiers allant du nombre "1" à l'infini que l'on peut aussi représenter de la manière suivante : [1; ∞].

4.3.5 long

Le type de données *long* est dérivé du type de données *integer* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Les types numériques comme *integer* ou *decimal* n'ont pas de longueur limitée au niveau du nombre. Mais avec le type de données *long* et ses descendants, des limites sont définies. Par exemple, le type de données *long*, n'accepte que l'ensemble des nombres entiers signés compris entre -9223372036854775808 et 9223372036854775807 qui correspondent à des valeurs pouvant être stockées dans 64 bits.

4.3.6 int

Le type de données *int* est dérivé du type de données *long* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Le type de données *int*, n'accepte que l'ensemble des nombres entiers signés compris entre -2147483648 et 2147483647 qui correspondent à des valeurs pouvant être stockées dans 32 bits.

4.3.7 short

Le type de données *short* est dérivé du type de données *int* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Le type de données *short* n'accepte que l'ensemble des nombres entiers signés compris entre -32768 et 32767 qui correspondent à des valeurs pouvant être stockées dans 16 bits.

4.3.8 byte

Le type de données *byte* est dérivé du type de données *short* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Le type de données *byte* n'accepte que l'ensemble des nombres entiers signés compris entre -128 et 127 qui correspondent à des valeurs pouvant être stockées dans 8 bits.

4.3.9 unsignedLong

Le type de données *unsignedLong* est dérivé du type de données *nonNegativeInteger* et reprend les mêmes caractéristiques en y ajoutant des contraintes. En suivant le même principe que le type de données *long*, le type de données *unsignedLong* et ses descendants, ont des limites définies, ne prennent en compte que l'ensemble des nombres entiers sont non signés et correspondent à des nombres entiers positifs. Par exemple, le type de données *unsignedLong*, n'accepte que l'ensemble des nombres entiers non signés compris entre 0 et 18446744073709551615 qui correspondent à des valeurs pouvant être stockées dans 64 bits.

4.3.10 unsignedInt

Le type de données *unsignedInt* est dérivé du type de données *unsignedLong* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Le type de données *unsignedInt* n'accepte que l'ensemble des nombres entiers non signés compris entre 0 et 4294967295 qui correspondent à des valeurs pouvant être stockées dans 32 bits.

4.3.11 unsignedShort

Le type de données *unsignedShort* est dérivé du type de données *unsignedInt* et reprend les mêmes caractéristiques en y ajoutant des contraintes. Le type de données *unsignedShort* n'accepte que l'ensemble des nombres entiers non signés compris entre 0 et 65535 qui correspondent à des valeurs pouvant être stockées dans 16 bits.

4.3.12 unsignedByte

Le type de données *unsignedByte* est dérivé du type de données *unsignedShort* reprend les mêmes caractéristiques en y ajoutant des contraintes. Le type de données *unsignedByte* n'accepte que l'ensemble des nombres entiers non signés compris entre 0 et 255 qui correspondent à des valeurs pouvant être stockées dans 8 bits.

4.4 float et double

Les types de données *float* et *double* sont des types primitifs et permettent de représenter les nombres réels à précision simple (32 bits) pour *float* et double (64 bits) pour *double*. Ces deux types de données permettent une grande échelle de nombres dans un stockage de longueur fixe et acceptent aussi la notation scientifique. De plus, ces deux types de données acceptent également des valeurs spéciales comme :

- Zéro positif (0) ;
- Zéro négatif (-0) ;
- Infini (INF) qui est supérieur à toute valeur ;
- Moins l'infini (-INF) qui est inférieur à n'importe quelle valeur ;

- Pas un nombre (NaN).

Exemple de valeurs conformes :
"0", "-0", "INF" (infini positif), "-INF" (infini négatif, "NaN" (Not a Number),"2.55E+2", etc.

Exemple de valeurs non conformes :
"+INF" (pas de signe positif pour infini),
"2.55 E+2" (espace interdit), etc.

4.5 boolean

Le type de données *boolean* est un type primitif et permet de représenter les valeurs logiques *true* ou *1* et *false* ou *0*. Dans ce type la valeur vraie (true) vaut 1 et la valeur fausse (false) vaut 0. Bien que le type *boolean* peut avoir des valeurs numériques comme 0 ou 1, ce n'est pas un type numérique comme *decimal*, *float* ou *double*, car ce type n'est pas conçu pour effectuer des opérations arithmétiques, comme l'addition (+), la soustraction (-), la multiplication (*), la division (/), etc.

Exemple de valeurs conformes :
"true" ou "1", "false" ou "0"

5 Types dates et heures

Pour les types de données, dates et heures, les valeurs doivent respecter le format de la norme ISO 8601, qui est le seul format pris en charge par le XML Schema Definition (XSD) du W3C.
Cela implique également l'usage du calendrier Grégorien. Le Time Zone peut être aussi indiqué de manière optionnelle (+01:00, -01:00 ou Z qui est équivalent à +00:00 ou -00:00).

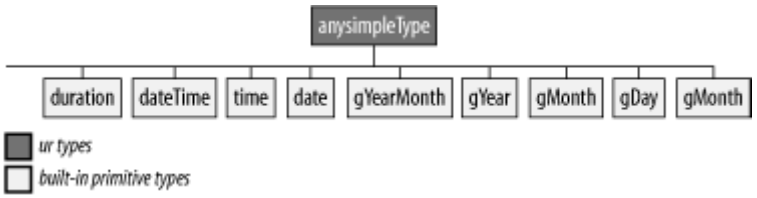

Figure 5 – Illustration hiérarchique pour les types de données dates et heures[10]

5.1 dateTime

Le type de données *dateTime* est un type primitif et permet de représenter un moment spécifique du temps (date et heure). La valeur de ce type de données doit respecter le format suivant : "CCYY-MM-DDThh:mm:ss[Time Zone]". Ce format se décrit de la manière suivante :

- CC : représente le siècle (Century) ;
- YY : représente l'année (YY) ;
- MM : représente le mois (MM) ;
- T : valeur fixe représentant le début de la zone de temps ;
- hh : représente l'heure ;
- mm : représente la minute ;
- ss : représente la seconde ;
- [Tine Zone]: représente la position relative (nombre d'heures) à UTC (fuseau horaire).

Dans "Time Zone", il est possible de saisir la lettre "Z" qui correspond à l'UTC ou le signe "+" ou "-" suivi de la différence d'heures (fuseau horaire) avec l'UTC.
Selon ce format, toutes les données doivent être saisies, excepté le "Time Zone" qui est optionnel.

[10] Date and time datatypes: https://www.oreilly.com/library/view/xml-schema/0596002521/ch04.html

Exemple de valeurs conformes :
"2022-08-15T08:00:52", "2022-08-15T08:00:52Z",
"2022-08-15T08:00:52+02:00", "-2022-08-15T08:00:52", etc.

Exemple de valeurs non conformes :
"2022-08-15" (manque les données des heures),
"-22-08-15T08:00:52" (selon le format la saisie de l'année n'est pas correcte),
"-2022-08-15T25:00:52" (l'heure 25 n'est pas correcte), etc.

5.2 date

Le type de données *date* est un type primitif et permet de saisir une partie de la date de *dateTime*.
Ce type de données permet de représenter une date dans le temps au format "CCYY-MM-DD[Time Zone]".

Exemple de valeurs conformes :
"2022-08-15", "2022-08-15Z", "2022-08-15+02:00", "2022-08-15-02:00", etc.

Exemple de valeurs non conformes :
"2022-08" (manque la donnée du jour),
"2022-13-15+02:00" (13 n'est pas un mois correct),
"2022-13-35" (35 n'est pas un jour correct), etc.

5.3 gYearMonth

Le type de données *gYearMonth* est un type primitif et permet de saisir l'année et le mois d'une date au format "CCYY-MM[Time Zone]". La lettre "g" de *gYearMonth* signifie "Grégorien" en lien avec le calendrier.

Exemple de valeurs conformes :
"2022-08", "2022-08+02:00", "2022-08-02:00", "2022-08Z", etc.

Exemple de valeurs non conformes :
"2022" (manque la donnée du mois),
"22-08" (format de l'année est incorrect),
"2022-08-15" (le jour 15 n'est pas autorisé), etc.

5.4 gYear

Le type de données *gYear* est un type primitif et permet de saisir uniquement l'année d'une date au format "CCYY[Time Zone]".

Exemple de valeurs conformes :
"2022", "2022+02:00", "2022-02:00", "2022Z", etc.

Exemple de valeurs non conformes :
"2022-08" (le mois n'est pas autorisé),
"22" (format de l'année est incorrect), etc.

5.5 time

Le type de données *time* est un type primitif et permet de saisir uniquement la partie des heures, des minutes et des secondes de *dateTime* au format : "hh:mm:ss[Time Zone]".

Exemple de valeurs conformes :
"08:02:55", "08:02:55+02:00", "08:02:55Z", etc.

Exemple de valeurs non conformes :
"08:02" (manque les secondes),
"25:02:55" (25 n'est pas une heure correcte),
"8:02:55" (format de l'heure incorrect),
"-08:02:55" (le signe – est interdit), etc.

5.6 gDay

Le type de données *gDay* est un type primitif et permet de saisir un jour du calendrier Grégorien au format : ---DD[Time Zone].

Exemple de valeurs conformes :
"---01", "---14Z", "---31+02:00", etc.

Exemple de valeurs non conformes :
"--01-" (ne respecte pas le format de gDay),
"---32" (32 n'est pas un jour correct du calendrier Grégorien),
"01" (manque "---" devant 01), etc.

5.7 gMonthDay

Le type de données *gMonthDay* est un type primitif et permet de saisir un mois et un jour du calendrier Grégorien --MM-DD[Time Zone].

Exemple de valeurs conformes :
"--08-15", " --08-15Z", " --08-15-02:00", etc.

Exemple de valeurs non conformes :
"-08-15-" (ne respecte pas le format de gMonthDay),
"--08-32" (32 n'est pas un jour correct), etc.

5.8 gMonth

Le type de données *gMonth* est un type primitif et permet de saisir un mois du calendrier Grégorien --MM[Time Zone].

Exemple de valeurs conformes :
"--01", " --06Z", "—12+06:00", "--12-06:00", etc.

Exemple de valeurs non conformes :
"-08" (ne respect pas le format de gMonth),
"--08-15" (le jour 15 n'est pas autorisé),
"--13" (13 n'est pas un mois correct), etc.

5.9 duration

Le type de données *duration* est un type primitif et permet de saisir une durée de temps au format suivant : "PnYnMnDTnHnMnS", selon la norme ISO 8601. Ce format se décrit de la manière suivante :

- P: représente le début (obligatoire) ;
- n: représente la valeur de l'année ;
- Y: représente l'année ;
- n: représente la valeur du mois ;
- M: représente le mois ;
- n: représente la valeur du jour ;
- D: représente le jour ;
- T: représente le début de la zone de temps ;
- n: représente la valeur de l'heure ;
- H: représente l'heure ;
- n: représente la valeur de la minute ;
- M: représente la minute ;
- n: représente la valeur de la seconde ;
- S: représente la seconde.

Le problème est la précision de la durée, car :
- 1 mois = entre 28 et 31 jours
- 1 année = 365 ou 366 jours

De ce fait, les durées ne sont pas toujours comparables. Pour diminuer ces indéterminations, il est possible de créer son propre type simple personnalisé dérivé du type de données *duration*.

A prendre en compte également que :
- Les nombres d'années, de mois, de jours, d'heures, de minutes et de secondes sont des entiers positifs.
- Les nombres de secondes peuvent être indiqués au format décimal positif.
- Il est possible d'exprimer une durée négative en ajoutant le signe – devant le P (-P).

Exemple de valeurs conformes :
"P2Y4M8DT12H24M16S" correspond à une durée de 2 ans, 4 mois, 8 jours, 12 heures, 24 minutes et 16 secondes
"-P60Y" correspond à une durée de moins 60 ans, etc.

Exemple de valeurs non conformes :
"10Y" (manque le P pour le début),
"P-10Y" (nombre négatif interdit à l'intérieur),
"P30S" (il manque le T après le P),
"P6M2Y" (selon format du type duration, l'ordre n'est pas respecté), etc.

6 Types listes

Les types listes sont des types dérivés et leurs valeurs sont des listes d'éléments séparés par des espaces blancs. Il existe trois types de listes prédéfinis qui sont :
- NMTOKENS → dérivé de NMTOKEN ;
- IDREFS → dérivé de IDREF ;
- ENTITIES → dérivé de ENTITY.

Chaque élément de la liste doit respecter les contraintes de son type de données dérivé (NMTOKEN, IDREF ou ENTITY).

Exemple de valeurs conformes pour le type IDREFS :
"C01 C02 C03 C04", "PROD1 PROD2", etc.

Exemple de valeurs non conformes pour le type IDREFS :
"1 2 3 4" (les chiffres ne respectent pas la contrainte du type IDREF), etc.

Exemple de valeurs conformes pour le type NMTOKEN :
"Madame Monsieur ", "1 2 3 4".

Exemple de valeurs non conformes pour le type NMTOKEN :
"Madame,Monsieur " (, ne respecte pas la contrainte du type NMTOKEN),
" 1 2 3 4 " (espace avant 1 et après 4 ne respecte pas la contrainte du type NMTOKEN).

7 Types anySimpleType

Le type de données *anySimpleType* est le type de base (ur-type) de tout type simple primitif et accepte n'importe quelle valeur sans aucune restriction. Ce type ne peut pas être dérivé et doit être utilisé uniquement quand on ne peut pas faire autrement.

Exemple de valeurs conformes :
"--01", " 128.331", " 2022-08-15T08:00:52", "Le petit train", etc.

8 Synthèse

Concernant les types prédéfinis primitifs ou dérivés, nous constations qu'il existe différents types de données. Un certain nombre de ces types préfinis (numériques, textes, dates, etc.) permettent d'améliorer la gestion du contenu d'un document XML en apportant de la précision aux données (format, limites, etc.).
Nous remarquons également que certains types de données sont plus spécifiques à des domaines particuliers, comme QName ou NCName qui sont des types plus pensés pour XML en lui-même. Ainsi que les types ENTITY et ENTITIES qui sont plus dédiés à la compatibilité pour un DTD (Document Type Definition). Ces types ne seront pas forcément utiles dans la cadre de la gestion de données plus orientée "métier" que "technique".
Ces types de données prédéfinis contribuent à améliorer la qualité des données d'un document XML pour sa validation et la précision de son contenu. Pour cela, il faut définir correctement le typage des données dans le schéma XML (XSD), qui est associé au document XML. Par exemple, si nous voulons saisir une date, il faut utiliser le type *date*. S'il faut saisir un nombre entier, il est possible d'utiliser le type *integer* et ses différentes variantes, etc. Nous verrons plus loin dans le cours, qu'il est possible encore d'affiner le typage des données, en créant des types de données simples personnalisés qui sont dérivés à partir de types de données primitifs ou d'autres types de données dérivés.

9 Exemple

Pour illustrer l'utilisation de ces différents types de données prédéfinis décrits dans ce document, nous allons utiliser un extrait d'exemple de données, qui sont fictives, de gestion de joueurs et joueuses de football. Un document XML contenant ces données est associé à un schéma XML au format XSD. Dans ce schéma XML (XSD) les nœuds (éléments et attributs) permettant de contenir les données simples ont le type de données *xs:string* qui est proposé par défaut par XMLSpy. Pour choisir le bon type de données prédéfini à attribuer à chaque élément simple ou attribut, quelques conseils sont énumérés. Pour terminer, un exemple de ce schéma XML (XSD) adapté est montré avec les types de données prédéfinis qui ont été choisis.

9.1 Schéma XSD initial

Ci-dessous est montré le schéma XML au format XSD initial avec les types de données par défaut *xs:string* proposé par XMLSpy.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- édité avec XMLSpy v2020 rel. 2 sp1 (x64) (http://www.altova.com) par HE-Arc (Haute Ecole Arc) -->
<!--Exemple de schéma xsd pour l'exemple de la gestion de joueurs et joueuses de football-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:element name="nom" type="xs:string"/>
	<xs:element name="prenom" type="xs:string"/>
	<xs:element name="date_naissance" type="xs:string"/>
	<xs:element name="telephone" type="xs:string"/>
	<xs:element name="email" type="xs:string"/>
	<xs:element name="posteJoueur" type="xs:string"/>
	<xs:element name="marie" type="xs:string"/>
	<xs:element name="remarque" type="xs:string"/>
	<xs:attribute name="idJoueur" type="xs:string"/>
	<xs:attribute name="sexe" type="xs:string"/>
	<xs:element name="joueur">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="nom"/>
				<xs:element ref="prenom"/>
				<xs:element ref="date_naissance"/>
				<xs:element ref="telephone"/>
				<xs:element ref="email"/>
				<xs:element ref="posteJoueur"/>
				<xs:element ref="marie"/>
				<xs:element ref="remarque" minOccurs="0"/>
			</xs:sequence>
			<xs:attribute ref="idJoueur"/>
			<xs:attribute ref="sexe"/>
		</xs:complexType>
	</xs:element>
	<xs:element name="football">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="joueur" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

9.2 Document XML

Ci-dessous est montré un exemple de document XML lié au schéma XML montré précédemment. Ce document XML contient des données de gestion de joueurs et joueuses de football.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Edité avec XMLSpy v2020 rel. 2 sp1 (x64) (http://www.altova.com) par HE-Arc (Haute Ecole Arc) -->
<!--Exemple de document XML contenant des données pour la gestion de joueurs et joueuses de football-->
<football xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="XML-02-02_Exemple_Schema.xsd">
	<joueur idJoueur="J1" sexe="masculin">
		<nom>Robert</nom>
		<prenom>Gilles</prenom>
		<date_naissance>1995-08-08</date_naissance>
		<telephone>0796124538</telephone>
		<email>Gilles.Robert@gmail.com</email>
		<posteJoueur>attaquant</posteJoueur>
		<marie>true</marie>
	</joueur>
	<joueur idJoueur="J2" sexe="féminin">
		<nom>Wuetrich</nom>
		<prenom>Vanessa</prenom>
		<date_naissance>2000-06-28</date_naissance>
		<telephone>0762615433</telephone>
		<email>vanessa.wuetrich@bluewin.ch</email>
		<posteJoueur>milieu offensif</posteJoueur>
		<marie>false</marie>
	</joueur>
	<joueur idJoueur="J3" sexe="masculin">
		<nom>Cuche</nom>
		<prenom>Jules</prenom>
		<date_naissance>1998-05-20</date_naissance>
		<telephone>0771262899</telephone>
		<email>JCuche@gmail.com</email>
		<posteJoueur>gardien</posteJoueur>
		<marie>false</marie>
		<remarque>Jules a subi une opération de genou gauche.
		Jules est allergique au gluten</remarque>
	</joueur>
	<joueur idJoueur="J4" sexe="féminin">
		<nom>Dumont</nom>
		<prenom>Julie</prenom>
		<date_naissance>1995-04-20</date_naissance>
		<telephone>0783219876</telephone>
		<email>JD1995@gmail.com</email>
		<posteJoueur>arrière latéral</posteJoueur>
		<marie>true</marie>
	</joueur>
</football>
```

9.3 Schéma XSD adapté

Pour adapter le schéma XSD montré précédemment, il faut analyser le contenu du document XML associé. Le but est de contenir des données correctes et de diminuer le risque d'avoir des données erronées. Donc, il faut choisir les types de données prédéfinis répondant au mieux à nos besoins avec les restrictions adéquates, afin d'éviter au mieux les erreurs de saisie dans le document XML. Si nous analysons notre document XML, nous avons les données simples suivantes :

- **idJoueur** : est une donnée qui permet d'identifier le joueur, donc il faut qu'elle soit unique dans tout le document XML. Le type de données prédéfini *ID* est le type qui convient le mieux dans le cas présent.

- **sexe** : est une donnée permettant de définir le sexe du joueur et ne peut contenir qu'un seul mot : soit "masculin", soit "féminin". Pour cela, le type *NMTOKEN* convient le mieux, car il permet de saisir une chaîne de caractères sans espaces qui est dans le cas présent un mot.

- **nom** : est une donnée permettant de définir le nom du joueur ou joueuse. Le nom peut être composé d'un seul mot ou de plusieurs mots, parfois avec des espaces ou des tirets par exemple. Ce qui serait intéressant est d'éviter d'avoir des espaces avant et après le texte. Le type de données prédéfini *token* est le type qui convient le mieux dans le cas présent.

- **prenom** : est une donnée permettant de définir le prénom du joueur ou joueuse. Nous pouvons rencontrer le même genre de situation que la donnée du **nom**, donc nous prenons le même genre type de données prédéfini.

- **date_naissance** : est une donnée permettant de définir la date de naissance du joueur ou joueuse. Comme nous voulons, avoir une date comme donnée et rien d'autre, le type de données prédéfini *date* est le type qui convient le mieux dans le cas présent.

- **telephone** : est une donnée permettant de saisir le numéro de téléphone du joueur ou joueuse. Dans cette donnée, nous voulons avoir que des chiffres entiers. Selon nos connaissances actuelles, il faudrait au moins choisir un type de données prédéfini numérique et ensuite dans les descendants de ce type, il y a des types plus restrictifs comme *nonNegativeInteger*, *unsignedLong* ou *unsignedInt* qui permettent par exemple d'éviter de pouvoir saisir un nombre négatif. Ensuite cela dépendra du nombre de chiffres qu'il est possible de saisir et la taille du nombre. Est-ce que le numéro de téléphone n'a que dix chiffres (0791112233) ou plus (+41791112233). Pour être large, on peut prendre le type *unsignedLong*, mais est-ce le bon type de donnée ? Le fait de choisir un type numérique, est-ce le bon choix, car sur le fond, on ne fait pas d'opérations arithmétique avec un numéro de téléphone ? Nous voyons qu'il n'y a pas de type de données prédéfini adéquat et qu'il n'est pas toujours simple à répondre à cette question. D'où l'importance à poser cette question aux gens du métier, afin de choisir le bon type de données prédéfini ou de créer son propre type de données. Nous verrons plus tard, qu'en créant notre propre type de donnée « telephone » (création de type simple), on pourra mieux répondre au problème du type de données adéquat pour la saisie du numéro de téléphone.

- **email** : est une donnée permettant de saisir l'adresse email du joueur ou joueuse. L'adresse email peut être composée de différents caractères et il n'y a pas de type prédéfinis pour cela. Cependant, il ne faut avoir d'espaces avant et après l'adresse email, donc le type de données prédéfini *token* est le type qui convient le mieux dans le cas présent.

- **marie** : est une donnée indiquant si un joueur ou une joueuse est marié ou non. Donc, nous avons une valeur logique oui (true) ou non (false). Donc le type de données prédéfini *boolean* est le type qui convient le mieux dans le cas présent.

- **remarque** : est une donnée qui peut une plusieurs remarques liées au joueur ou joueuse, avec des retours de chariot, etc. De ce fait, pour rester large le type *string* est gardé.

À la suite de ces analyses et à ces réflexions, nous vous proposons l'adaptation du schéma XML (XSD) comme suit :

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- édité avec XMLSpy v2020 rel. 2 sp1 (x64) (http://www.altova.com) par HE-Arc (Haute Ecole Arc) -->
<!--Exemple de schéma xsd pour l'exemple de la gestion de joueurs et joueuses de football-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:element name="nom" type="xs:token"/>
	<xs:element name="prenom" type="xs:token"/>
	<xs:element name="date_naissance" type="xs:date"/>
	<xs:element name="telephone" type="xs:unsignedLong"/>
	<xs:element name="email" type="xs:token"/>
	<xs:element name="posteJoueur" type="xs:token"/>
	<xs:element name="marie" type="xs:boolean"/>
	<xs:element name="remarque" type="xs:string"/>
	<xs:attribute name="idJoueur" type="xs:ID"/>
	<xs:attribute name="sexe" type="xs:NMTOKEN"/>
	<xs:element name="joueur">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="nom"/>
				<xs:element ref="prenom"/>
				<xs:element ref="date_naissance"/>
				<xs:element ref="telephone"/>
				<xs:element ref="email"/>
				<xs:element ref="posteJoueur"/>
				<xs:element ref="marie"/>
				<xs:element ref="remarque" minOccurs="0"/>
			</xs:sequence>
			<xs:attribute ref="idJoueur"/>
			<xs:attribute ref="sexe"/>
		</xs:complexType>
	</xs:element>
	<xs:element name="football">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="joueur" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```