haute école neuchâtel berne jura arc gestion neuchâtel delémont

Hes·so Haute Ecole Spécialisée de Suisse occidentale

Certifié ISO 9001

62-31.3 XML

XML-02-P04-Patterns

Cédric Benoit

Plan du cours

- Présentation et organisation
- Introduction à XML
- Schémas XML
- **Types de données**
  - Types prédéfinis
  - Création de types simples
  - **Patterns**

Points abordés

- Pattern
- Expression régulière
- Restrictions
- Quantifications
- Entités
- Méta-Caractères
- Exemples
- …

Pattern

pattern → est une facette (modèle) permettant d'ajouter des contraintes (restrictions) sur les valeurs correspondantes au type de données que l'on a défini.

Pattern s'emploie avec la balise ***restriction*** et s'applique aux types de données simples

Exemple :

```
...
<xs:restriction base="type de donnée">
        <xs:pattern value="expression régulière"/>
</xs:restriction>
...
```

Expression régulière (1)

La réalisation d'un pattern se fait au travers d'une expression régulière.

Une expression régulière est une chaîne de caractères constituée de :

- **Caractères ordinaires** (par exemple les lettres de a à z, ou les nombres de 1 à 9)
- **Caractères spéciaux** (méta-caractères, quantificateurs)

Expression régulière (2)

Une expression régulière ou regular expression en anglais (Regexp ou Regex) s'emplooie dans différents domaines :

- Base de données (Oracle,…)
- Langage de programmation (Javascript, Java, C#,….)
- Schéma XML (XSD, Schematron,…)
- …

Expression régulière (3)

L'expression régulière dans XML est interprétée d'une manière un peu différente.

Le schéma XML interprète implicitement l'expression régulière complète.

Exemple :

- Pattern: sapin
- Valeur conforme : "sapin"
- Valeurs non conformes : "le sapin", "sapin blanc"

Restriction - enumeration

Exemple avec la restriction **enumeration** : restreindre un nombre entier aux valeurs "1", "5" ou "15" .

```
<xs:simpleType name="TypeByte">
    <xs:restriction base="xs:byte">
        <xs:enumeration value="1"/>
        <xs:enumeration value="5"/>
        <xs:enumeration value="15"/>
    </xs:restriction>
</xs:simpleType>
```

Cette restriction accepte 1, 5 et 15, mais aussi "01" et "005".

Restriction – pattern (1)

En utilisant le même exemple de contrainte que le slide précédent, mais avec la restriction pattern.

```
<xs:simpleType name="TypeByte">
    <xs:restriction base="xs:byte">
        <xs:pattern value="1"/>
        <xs:pattern value="5"/>
        <xs:pattern value="15"/>
    </xs:restriction>
</xs:simpleType>
```

Cette restriction accepte uniquement "1", "5" et "15".

Restriction – pattern (2)

- Le type "TypeByte" précédent :
  - Dérive du type prédéfini **byte**
  - Garde une représentation de type **byte**
  - Permet d'appliquer les facettes de **byte**

- L'espace "lexical" (pattern) est limité aux valeurs :
  - "1"
  - "5"
  - "15"

Quantification (1)

Le pattern : <xs:pattern value="15"/> spécifie trois conditions :

- premier caractère = "1"
- second caractère = "5"
- le texte se termine après le caractère "5"

Chaque condition est une "pièce" de l'expression régulière.

Quantification (2)

Chaque pièce est composée :

- D'une entité représentant un caractère ou un groupe de caractère
- D'un quantificateur optionnel (+, *,…)

Les caractères (sauf les cas spéciaux) sont des entités triviales :

- "a" représente "a" , "b" -> "b" , etc…

Quantification (3)

Les quantificateurs supportent deux syntaxes :

- Caractère spécial
  - "*" = 0 ou plus
  - "+" = 1 ou plus
  - "?" = 0 ou 1

- Intervalle
  - {n} = exactement n fois
  - {n,m} = entre n et m
  - {n,} = n fois ou plus

Quantification (4)

Exemple :

```
<xs:simpleType name="TypeByte">
        <xs:restriction base="xs:byte">
                <xs:pattern value="1?5?"/>
        </xs:restriction>
</xs:simpleType>
```

Quantification (5)

Signification :

- 0 ou 1 caractère "1" suivi par
- 0 ou 1 caractère "5"

Accepte "1", "5", "15" et "" (vide)

"" n'est pas de type **byte** → OK

Quantification (6)

Définir une limite du nombre de "0" devant la valeur :

```
<xs:simpleType name="TypeByte">
        <xs:restriction base="xs:byte">
                <xs:pattern value="0{0,2}1?5?"/>
        </xs:restriction>
</xs:simpleType>
```

Caractères spéciaux (1)

Caractères spéciaux doivent être précédés du caractère "\" pour être considérés comme un caractère normal.
Exemple :

- \n ou &#x0A → Saut de ligne
- \r ou &#x0D → Retour chariot
- \t ou &#x09 → Tabulation
- \\ , \|, \., \-, \^ , \?, *, \+, \{, \}, \(, \), \[, \] …

Caractères spéciaux (2)

Le caractère "." (point) représente n'importe quel caractère (sauf saut de ligne et retour chariot).

".*" représente n'importe quelle suite de caractères.

Caractères spéciaux - Exemple

Exemple : un pattern qui correspond à n'importe quel multiple de 10.

```
<xs:simpleType name="TypeZeroEtMultipleDeDix">
    <xs:restriction base="xs:integer">
        <xs:pattern value=".*0"/>
    </xs:restriction>
</xs:simpleType>
```

→ OK puisque le type de base est **integer**.

POSIX et PCRE

POSIX (Portable Operating System Interface Unix)

- Mis en avant par PHP, il est plus simple que PCRE mais moins performant. POSIX est un standard.
- IEEE/Open Group 1003.1-2017 : https://standards.ieee.org/standard/1003_1-2017.html
- The Open Group Base Specifications : https://pubs.opengroup.org/onlinepubs/9699919799/

PCRE (Perl Compatible Regular Expression)

- Issue du langage PERL, elles sont plus performante mais un peu plus complexe.
- Perl-compatible Regular Expressions (PCRE) : https://www.pcre.org/original/doc/html/index.html

Entités (1)

Les entités conformes au langage "PERL" sont aussi supportées dans la faccette **pattern**. Par exemple :

- \s
  - Espace blanc
- \S
  - Pas un espace blanc
- \d
  - Chiffre (0-9 et autres chiffres dans d'autres alphabets)
- \D
  - Pas un chiffre

Entités (2)

- \w
  - Mot : tout caractère unicode qui n'est pas de type "ponctuation", "séparateur" ou "autre"
- \W
  - Pas un mot
- \c
  - Tout caractère utilisable dans un nom XML
  - Caractères, chiffres, ".", ":", "-" …
- \C
  - Inverse de \c

Entités - Exemple

Reprendre l'exemple de limitation à un multiple de 10 :

```
<xs:simpleType name="TypeZeroEtMultipleDeDix">
    <xs:restriction base="xs:integer">
        <xs:pattern value=".*0"/>
    </xs:restriction>
</xs:simpleType>
```

Peut être écrit différent comme suit :

- `<xs:pattern value="[+-]?\d*0"/>` → \d (PCRE)

Ou plus simplement

- `<xs:pattern value="[+-]? [0-9]*0"/>`

Entités (3)

Les entités Unicodes sont aussi supportées.

Unicode définit deux types d'entités :

- Catégories
  - classification par signification (lettres, chiffres, ponctuation, …
- Blocs
  - classification par localisation (Latin, Arabe, gothique, …)

Entités (4)

- \p{Name} = Catégorie
- \p{isName} = Bloc
- \P{Name} et \P{isName} = Inverse

```
<xs:simpleType name="TypeLettreDeBaseLatin">
    <xs:restriction base="xs:token">
        <xs:pattern value="\p{IsBasicLatin}*"/>
        <xs:pattern value="\p{L}*"/>
    </xs:restriction>
</xs:simpleType>
```

Unicode - Catégories

Exemple de catégories Unicode :

Unicode Character Class	Includes
C	Other characters (non-letters, non symbols, non-numbers, non-separators)
Cc	Control characters
Cf	Format characters
Cn	Unassigned code points
Co	Private use characters
L	Letters
Ll	Lowercase letters
Lm	Modifier letters

Unicode - Blocs

Exemple de blocs Unicode :

AlphabeticPresentationForms	Arabic	ArabicPresentationForms-A
ArabicPresentationForms-B	Armenian	Arrows
BasicLatin	Bengali	BlockElements
Bopomofo	BopomofoExtended	BoxDrawing
BraillePatterns	ByzantineMusicalSymbols	Cherokee
CJKCompatibility	CJKCompatibilityForms	CJKCompatibilityIdeographs
CJKCompatibilityIdeographsSupplement	CJKRadicalsSupplement	CJKSymbolsandPunctuation
CJKUnifiedIdeographs	CJKUnifiedIdeographsExtensionA	CJKUnifiedIdeographsExtensionB
CombiningDiacriticalMarks	CombiningHalfMarks	CombiningMarksforSymbols
ControlPictures	CurrencySymbols	Cyrillic
Deseret	Devanagari	Dingbats
EnclosedAlphanumerics	EnclosedCJKLettersandMonths	Ethiopic
GeneralPunctuation	GeometricShapes	Georgian
Gothic	Greek	GreekExtended
Gujarati	Gurmukhi	HalfwidthandFullwidthForms

Délimiteurs de chaîne

- On spécifie qu'une chaîne de caractères doit commencer par un caractère précis avec le **méta-caractère ^**.
- On spécifie qu'une chaîne de caractères doit finir par un caractère précis avec le **méta-caractère $**.
- L'expression régulière ^une belle journée$ indique que la chaîne à tester doit se composer d'une ligne constituée de "une belle journée"
- Dans XML Schema, pas besoin de ces deux délimiteurs : ^ et $ → car ceci est implicitement pris en compte.

Le méta-caractère OU « | »

Le méta-caractère "|" permet de spécifier une alternative.

Exemple : chat|chien

Valeurs conformes :

Je suis un chat

Je suis un chien

Valeurs non conformes :

Je suis un lapin

Différence entre | et les []

Les [] permettent de faire un choix logique sur un ensemble de caractères (choix étendu). Par exemple :

- [abcdefghijklmnopqrstuvwxyz]
- [a-z]
- ou de chiffres
- [0123456789]
- [0-9]

L'opérateur "|" est pratique lorsque les choix sont peu nombreux. Par exemple :

- a|b|c est l'équivalent de [abc] ou [a-c]

Le méta-caractère ()

On peut utiliser des **sous-expression** avec des parenthèses.

Exemple avec le pattern : [1-9]{2}[/\-\.][1-9]{4}

Valeurs conformes :

"12.1221", "41-4567", "35/9876"

Valeurs non conformes :

"12.12", "12.213", "12.12332"

Le méta-caractère | dans une sous-expression

L'opérateur effectue une condition logique (ou) entre sa partie gauche et sa partie droite. Par exemple :

- *lapin chat|chien → lapin chat, lapin chien ou chien*
- *lapin (chat|chien) → lapin chat ou lapin chien mais pas chien*

L'opérateur "|" a la plus faible précédence par rapport à l'ensemble des opérateurs

- (a|b)+ *=> a, b, bbbbbbb, aaaaaa*
- (a|b+) *=> a, b, bbbbbbb, aaaaaa*

Différents exemples (1)

Exemple : le caractère "|" est assimilé à l'opérateur OR (OU exclusif).

```
<xs:simpleType name="TypeByte">
    <xs:restriction base="xs:byte">
        <xs:pattern value="1|5|15"/>
    </xs:restriction>
</xs:simpleType>
```

Quelles sont les valeurs conformes ? *1*, *5*, *15*, mais aussi *1515*, …

Différents exemples (2)

Exemple : l'exemple du slide est repris, mais avec l'ajout des ().

```
<xs:simpleType name="TypeByte">
    <xs:restriction base="xs:byte">
        <xs:pattern value="(1|5|15)"/>
    </xs:restriction>
</xs:simpleType>
```

Quelles sont les valeurs conformes ? *1*, *5*, *15* uniquement

Différents exemples (3)

Exemple : limitation d'une URI à un certain format.

```
<xs:simpleType name="TypeHttpURI">
    <xs:restriction base="xs:anyURI">
        <xs:pattern value="https://.*"/>
    </xs:restriction>
</xs:simpleType>
```

Quelles sont les valeurs conformes ?
https://www.google.ch, *https://abc.123*, ...

Différents exemples (4)

Exemple : avoir un nombre entier sans les "0" devant et avec le signe positif et négatif, sans oublier la valeur 0.

```
<xs:simpleType name="TypeEntierSansZeroDevant">
    <xs:restriction base="xs:integer">
        <xs:pattern value="[+-]?([1-9][0-9]*)|0"/>
    </xs:restriction>
</xs:simpleType>
```

Quelles sont les valeurs conformes ? *0, -0, +0, 1, 12* ...

Différents exemples (5)

Exemple : avoir que le mot "sapin"

```
<xs:simpleType name="MotSapin">
        <xs:restriction base="xs:string">
                <xs:pattern value="sapin"/>
        </xs:restriction>
</xs:simpleType>
```

Quelles sont les valeurs conformes ? *sapin*

Différents exemples (6)

Exemple : avoir un texte contenant le mot "sapin".

```
<xs:simleType name="TexteSapin">
        <xs:restriction base="xs:string">
                <xs:pattern value=".*sapin.*"/>
        </xs:restriction>
</xs:simpleType>
```

Quelles sont les valeurs conformes ? *sapin*, mais aussi *Le sapin*, *sapin blanc*, *le sapin de noël* ...

Testeurs d'expressions régulières

Il existe divers sites Internet pour tester des expressions régulières, dont voici quelques exemples, où la liste n'est pas exhaustive :

- https://regex101.com/
- http://www.xul.fr/ecmascript/expression-reguliere-demo.php
- http://regexpal.com/
- …

Merci pour votre attention !