Chapitre VIII

Contraintes de cohérence : Unicité et Existence

XMLSchema permet de spécifier des contraintes de cohérence qui s'appliquent globalement sur un document (instance XML). Un tel document utilisant un schéma contraint devra non seulement être valide par rapport à son schéma, mais en plus vérifier des contraintes de cohérence.
En XMLSchema v1.0, les contraintes de cohérence sont de deux types :

- les contraintes d'unicité
- les contraintes d'existence

Mais qu'est ce qu'une contrainte de cohérence ? Le plus simple est de prendre un exemple concret : Supposons que nous modélisions un centre de soin. Il paraît évident que les personnels de soin doivent avoir un identifiant (l'**existence** de l'identifiant doit être vérifiée) et que cet identifiant est unique (l'**unicité** de cet identifiant doit être respectée) de façon à ce qu'aucune personne ne puisse avoir le même identifiant.

NB : En XMLSchema v1.1, il existe en plus de cela les `assertions`. Ce sont des composants de XML Schema 1.1 qui permettent de contraindre l'existance et les valeurs relatives à des éléments et/ou des attributs. Un autre exemple pourrait être de vérifier que la valeur d'un attribut, par exemple `temperatureMin`, est bien inférieure ou égale à celle d'un autre attribut (respectivement `temperatureMax`). Dans ce cours, nous ne traiterons pas des composants au delà de la version v1.0, la première raison étant que la prise en charge de cette version 1.1 est assez compliquée et demande l'installation du processeur XML Saxon, la deuxième étant que son installation est payante !

1 Contraintes d'unicité

Une contrainte d'unicité spécifie qu'il ne peut exister qu'un seul élément ou attribut d'une propriété fixée dans un élément donné. Cette notion correspond à celle des clefs des bases de données.

En XMLSchema, l'unicité est obtenue en utilisant les éléments `xsd:key` ou `xsd:unique`.

1.a Définir une contrainte d'unicité

Un exemple complet

Commençons par détailler un exemple complet de schéma contraint : la modélisation d'une matrice mathématique. En mathématiques, une matrice est un tableau de coefficients spécifiant une transformation applicable sur des vecteurs. On peut modéliser une matrice comme une collection de lignes (rows) contenant chacune le même nombre de cases (cells). Notons qu'un vecteur est une matrice à une seule dimension. Le schéma XML montré figure VIII.1 modélise une matrice.

Dans ce schéma XML, 2 contraintes d'unicité sont appliquées : une sur les numéros de lignes, l'autre sur les numéros de cases dans chaque ligne:

- Les cellules doivent avoir un numéro `cellNo` unique défini par une clef `cellNoKey`. Cette clef est déclarée et définie lors de la déclaration de l'élément `row : mat:Row`. C'est en effet quand une ligne (row) est déclarée que la portée de la clef est définie et s'applique aux cellules de la ligne.
- De même, les lignes ont un numéro `rowNo` unique défini par une clef `rowNoKey`. Cette clef est déclarée et définie lors de la déclaration de l'élément `matrix : mat:Matrix`. C'est en effet quand une matrice (matrix) est déclarée que la portée de la clef est définie et s'applique aux lignes de la matrice.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- SHEMA 1 -->
<xsd:schema version="1.0"
            targetNamespace="https://www.timc.imag.fr/matrix"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:mat="https://www.timc.imag.fr/matrix"
            elementFormDefault="qualified">

    <!-- Déclaration de l'élément racine 'matrix:Matrix' -->
    <xsd:element name="matrix" type="mat:Matrix">
        <xsd:unique name="rowNoKey">
            <xsd:selector xpath="mat:row" />
            <xsd:field xpath="@rowNo" />
        </xsd:unique>
    </xsd:element>

    <!-- Définition du type complexe 'Cell' (cellules, c'est à dire les cases
        de la matrice) -->
    <xsd:complexType name="Cell">
                <xsd:attribute name="cellNo" type="xsd:int"  use="required"/>
                <xsd:attribute name="value" type="xsd:int" use="required"/>
    </xsd:complexType>

    <!-- Définition du type complexe 'Row' (ligne de la matrice) -->
    <xsd:complexType name="Row">
        <xsd:sequence>
            <xsd:element name="cell" type="mat:Cell" maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:attribute name="rowNo" type="xsd:int" use="required"/>
    </xsd:complexType>

    <!-- Définition du type complexe 'Matrix' (une matrice) -->
    <xsd:complexType name="Matrix">
        <xsd:sequence>
            <xsd:element name="row" type="mat:Row" maxOccurs="unbounded">
                <xsd:unique name="cellNoKey">
                    <xsd:selector xpath="mat:cell" />
                    <xsd:field xpath="@cellNo" />
                </xsd:unique>
            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="matName" type="xsd:string"/>
    </xsd:complexType>

</xsd:schema>
```

Figure VIII.1: Modélisation d'une matrice avec 2 contraintes d'unicité (Schema XML `Matrix.xsd`).

Un instance de document XML valide par rapport à ce schéma pourra être celle montrée figure VIII.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- INSTANCE 1 : une matrice 3x3 -->
<matrix
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns='https://www.timc.imag.fr/matrix'
    xsi:schemaLocation='https://www.timc.imag.fr/matrix Matrix.xsd'
    matName="m0">
    <row rowNo="0">
        <cell cellNo="0" value="1"/><cell cellNo="1" value="0"/><cell cellNo="2
            " value="0"/>
    </row>
    <row rowNo="1">
        <cell cellNo="0" value="0"/><cell cellNo="1" value="1"/><cell cellNo="2
            " value="0"/>
    </row>
    <row rowNo="2">
        <cell cellNo="0" value="0"/><cell cellNo="1" value="0"/><cell cellNo="2
            " value="1"/>
    </row>
</matrix>
```

Figure VIII.2: Document XML contraint par le schéma XML `Matrix.xsd`. Une matrice 3x3 est instanciée. Elle contient 3 lignes (row) numérotées de 0 à 2, contenant chacune 3 cellules (cell) numérotées de 0 à 2 également, contenant les valeurs des coefficients de la matrice.

Où déclarer une contrainte d'unicité

Les éléments `xsd:unique` ou `xsd:key` doivent être *éléments fils* d'un l'élément ; cet élément contient ceux sur lesquel va s'appliquer la contrainte d'unicité.
Dans l'exemple donné figure VIII.1, on voit effectivement que la contrainte `cellNoKey` s'appliquant sur les cellules d'une ligne, est élément fils de l'élement `row : mat:Row` qui contient les éléments de type `mat:Cell` sur lesquels s'applique cette contrainte. De même, la contrainte `rowNoKey` est élément fils de l'élément `matrix : mat:Matrix` qui contient les éléments de type `mat:Row` sur lesquels s'applique cette contrainte.

Attributs de la contrainte

Les éléments `xsd:key` ou `xsd:unique` contiennent :

- un attribut `id` qui est optionel. Cet identifiant de clef est unique.
- un attribut `name` qui est requis
- deux types d'éléments:
  - `xsd:selector` qui est requis. Une seule occurence doit être présente.
  - `xsd:field`. Au moins une occurence doit être présente, mais il peut y en avoir plusieurs.

  qui ont chacun un seul élément `xpath`.

Ainsi, les éléments définissant une contrainte d'unicité s'écrivent :

```
        <xsd:unique name="nom_de_la_clef" id="identifiant_de_la_clef">
            <xsd:selector xpath="éléments sur lesquels porte la contrainte" />
            <xsd:field xpath="éléments ou attributs spécifiant l'unicité" />
            <xsd:field xpath="autres éléments/attributs ...." />
            ...
        </xsd:unique>
```

ou

```
        <xsd:key name="nom_de_la_clef" id="identifiant_de_la_clef">
            <xsd:selector xpath="éléments sur lesquels porte la contrainte" />
            <xsd:field xpath="éléments ou attributs spécifiant l'unicité" />
            <xsd:field xpath="autres éléments/attributs ...." />
            ...
        </xsd:unique>
```

Figure VIII.3: Structure d'une clef d'unicité.

L'attribut `name` est utilisé uniquement par les contraintes d'existance `xsd:keyref` (voir la section 2. *Contraintes d'existence* plus bas). Si seule l'unicité est appliquée, alors cet attribut peut être ignoré (mais doit être présent avec une valeur de nom quelconque).

L'élément `xsd:selector`, via un chemin XPath spécifié dans son attribut `path`, spécifie sur quels éléments s'applique la contrainte d'unicité. Ces éléments, de même type, peuvent être multiples : il peut s'agir d'un seul élément comme d'une liste d'élément. Par exemple, dans la figure VIII.1, le sélecteur de la clef `cellNoKey` s'applique à la liste des noeuds (éléments ici) obtenue par le chemin xpath `mat:cell`, autrement dit toutes les cellules contenues dans une ligne.
L'élément `xsd:field` permet, via son chemin XPath spécifié dans son attribut `path`, de spécifier la valeur qui doit être rendue unique. Dans le même exemple de la clef `cellNoKey` (figure VIII.1), c'est l'attribut `cellNo` des cellules d'une ligne qui est ciblé et qui doit être unique. Plusieurs champs `xsd:field` peuvent être présents pour forcer l'unicité de plusieurs valeurs en même temps (voir la section dédiée ci dessous).

1.b `xsd:unique` et `xsd:key`

Ces deux éléments permettent de définir une contrainte d'unicité sur des valeurs. La différence entre le deux est un critère d'existence:

- une clef de type `xsd:key` est unique mais exige que le champs (`field`) existe pour chaque résultat du sélecteur dans le schéma et dans le document XML. La réciproque est que chaque élément dans le selecteur doit avoir une clef.
- ce n'est pas requis pour une clef de type `xsd:unique` : elle ne nécessite pas que le sélecteur contienne le champ à contraindre.

Dans le document d'exemple, figure VIII.1, les deux contraintes sont définies avec `xsd:unique`. Si maintenant on utilise un élément `xsd:key` pour définir la contrainte sur les numéros de cellules (`cellNo`), comme dans la figure VIII.4, la présence de l'attribut `cellNo` dans les éléments `mat:Cell` sera obligatoire (en supposant que le schéma, pour cet attribut, ne spécifie pas `use="required"` comme dans la figure VIII.1).

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- SHEMA 1 -->
<xsd:schema version="1.0"
            targetNamespace="https://www.timc.imag.fr/matrix"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:mat="https://www.timc.imag.fr/matrix"
            elementFormDefault="qualified">

    <!-- Déclaration de l'élément racine 'matrix:Matrix' -->
    <xsd:element name="matrix" type="mat:Matrix">
        <xsd:unique name="rowNoKey">
            <xsd:selector xpath="mat:row" />
            <xsd:field xpath="@rowNo" />
        </xsd:unique>
    </xsd:element>

    <!-- Définition du type complexe 'Cell' (cellules, c'est à dire les cases
        de la matrice) -->
    <xsd:complexType name="Cell">
```

```
                <xsd:attribute name="cellNo" type="xsd:int""/>
                <xsd:attribute name="value" type="xsd:int" use="required"/>
    </xsd:complexType>

    <!-- Définition du type complexe 'Row' (ligne de la matrice) -->
    <xsd:complexType name="Row">
        <xsd:sequence>
            <xsd:element name="cell" type="mat:Cell" maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:attribute name="rowNo" type="xsd:int"/>
    </xsd:complexType>

    <!-- Définition du type complexe 'Matrix' (une matrice) -->
    <xsd:complexType name="Matrix">
        <xsd:sequence>
            <xsd:element name="row" type="mat:Row" maxOccurs="unbounded">
                <xsd:key name="cellNoKey">
                    <xsd:selector xpath="mat:cell" />
                    <xsd:field xpath="@cellNo" />
                </xsd:key>
            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="matName" type="xsd:string"/>
    </xsd:complexType>

</xsd:schema>
```

Figure VIII.4: Utilisation de `xsd:key` au lieu de `xsd:unique` pour définir la contrainte d'unicité sur les numéros de cellules.

Dans ce cas, tous les éléments `mat:cell` présents dans le document XML doivent contenir un attribut `cellNo`, comme montré dans l'instance VIII.5. Par contre, les attributs `rowNo` des éléments `mat:row` ne sont pas requis : il n'est pas spécifié `use='required'` et la contrainte d'unicité qui s'applique sur eux est définie avec un élément `xsd:unique` qui n'impose pas de conditions d'existance.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- INSTANCE 2 : une matrice 3x3 -->
<matrix
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns='https://www.timc.imag.fr/matrix'
    xsi:schemaLocation='https://www.timc.imag.fr/matrix Matrix.xsd'
    matName="m0">
    <row>
        <cell cellNo="0" value="1"/><cell cellNo="1" value="0"/><cell cellNo="2
            " value="0"/>
    </row>
    <row>
        <cell cellNo="0" value="0"/><cell cellNo="1" value="1"/><cell cellNo="2
            " value="0"/>
    </row>
    <row>
        <cell cellNo="0" value="0"/><cell cellNo="1" value="0"/><cell cellNo="2
            " value="1"/>
    </row>
</matrix>
```

Figure VIII.5: Document XML contraint par le schéma XML `Matrix.xsd`. Il est semblable à celui montré figureVIII.2 au détail que les attributs `rowNo`, non requis, ne sont pas présents ; leur unicité était spécifiée, pas leur existence. Ce document est tout à fait valide.

En revanche, le document présenté figure VIII.6 **n'est pas valide** car il manque un attribut `cellNo` dont l'existence est imposée par la contrainte définie avec un élément `xsd:key`.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- INSTANCE 3 : une matrice 3x3 -->
<matrix
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns='https://www.timc.imag.fr/matrix'
    xsi:schemaLocation='https://www.timc.imag.fr/matrix Matrix.xsd'
    matName="m0">
    <row>
        <cell value="1"/><cell cellNo="1" value="0"/><cell cellNo="2" value="0"
            />
    </row>
    <row>
        <cell cellNo="0" value="0"/><cell cellNo="1" value="1"/><cell cellNo="2
            " value="0"/>
    </row>
    <row>
        <cell cellNo="0" value="0"/><cell cellNo="1" value="0"/><cell cellNo="2
            " value="1"/>
    </row>
</matrix>
```

Figure VIII.6: Document XML contraint par le schéma XML `Matrix.xsd`. Il est semblable à celui montré figureVIII.5 mais cette fois, **il manque un attribut `cellNo`** requis (celui de la première cellule). **Ce document n'est pas valide**.

1.c Portée de la contrainte d'unicité

De la même manière qu'une variable ou un attribut de classe en Java dispose d'une portée, une contrainte d'unicité a une portée : la contrainte d'unicité ne pourra s'appliquer que sur les valeurs contenues dans l'élément

Revenons à notre exemple de matrice (figure VIII.2) : dans une même ligne, les cellules doivent avoir des numéros différents, mais des cellules de lignes différentes peuvent avoir des numéros égaux car la portée d'unicité sur les numéros de cellules ne s'applique que pour chaque ligne. Pour s'en convaincre, il suffit de voir que cette contrainte d'unicité pour les numéros de lignes est définie lors de la déclaration de l'élément `row : mat:Row` : autrement dit, pour chaque occurence d'élément `row : mat:Row` qui sera créée dans la matrice, une nouvelle contrainte d'unicité pour les numéros de cellules sera définie.

1.d Contrainte d'unicité sur plusieurs valeurs

Dans une définition de contrainte d'unicité, il peut y avoir plusieurs éléments `xsd:field`. Cela permet de définir une clef d'unicité en utilisant plusieurs valeurs à la fois.
Attention ! Deux valeurs sont considérées différentes si elles diffèrent sur au moins 1 champ. Il n'est donc pas nécessaire que les deux valeurs soient uniques pour définir l'unicité. C'est la combinaison des deux valeurs qui fait l'unicité. Par exemple, si l'on considère les deux champs *nom* et *prénom* qui définissent l'identité d'une personne, la combinaison *{ Nicolas, Glade}* diffère de la combinaison *{ Nicolas, Cage}*.

Un exemple plus détaillé est donné figure VIII.7. Dans ce schema XML, on modélise une liste d'opérations matricielles. Ces opérations sont de type 'addition' (add), soustraction (sub), multiplication (mul). Une opération `Operation` est donc définie par un type d'opération (attribut `opType`), mais possède aussi deux autres attributs contribuant à l'identifier de façon unique, son identifiant `opID` et son nom `opName`. De plus, chaque opération se réfère, sous forme d'attributs, à deux matrices par leur nom dans deux attributs supplémentaires, `left` et `right`, qui indiquent la matrice à gauche de l'opérateur et la matrice à droite. Enfin, une opération spécifie le nom de la matrice résultante de l'opération dans un dernier attribut `result`. Au total, une opération a 6 attributs.
On remarque qu'une clef unique nommée `matUnique` est définie comme élément fils de l'élément `matrices`, une liste de matrices. Elle s'applique sur tous les éléments `mop:matrix` contenus dans cette liste, rendant ainsi chaque matrice unique par son nom `matName`.
De plus, une clef unique nommée `opUnique` est définie comme élément fils de l'élément `operations`.

Cette clef s'applique sur l'ensemble des opérations (`mop:operation`) et son unicité est définie grâce à trois champs : l'identifiant de l'opération `opID`, son nom `opName` et le type de l'opération `opType`. L'usage de ces trois valeurs à la fois permet de garantir l'unicité d'une opération. Il est tout à fait possible d'avoir 2 opérations de type multiplication (*mul*) mais avec des identifiants différents ou des noms différents, comme il est possible d'avoir 2 opérations de type différent (par exemple multiplication et soustraction), mais d'identifiants et/ou de noms égaux (voir par exemple l'instance de document XML figure VIII.8.

```
<?xml version="1.0"?>
<xsd:schema version="1.0"
            targetNamespace="https://www.timc.imag.fr/matrix"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:mat="https://www.timc.imag.fr/matrix"
            elementFormDefault="qualified">
    <xsd:include schemaLocation="./Matrix.xsd"/>

    <!-- Elément racine 'matrixComputation' -->
    <xsd:element name="matrixComputations">
        <xsd:complexType>
            <xsd:sequence>

                <!-- liste de matrices -->
                <xsd:element name="matrices" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="matrix" type="mat:Matrix"
                                minOccurs="1" maxOccurs="unbounded"/>
                        </xsd:sequence>
                    </xsd:complexType>
                    <!-- Unicité des matrices -->
                    <xsd:key name="matUnique">
                        <xsd:selector xpath="mat:matrix"/>
                        <xsd:field xpath="@matName"/>
                    </xsd:key>
                </xsd:element>

                <!-- liste d'operations -->
                <xsd:element name="operations" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="operation" type="mat:Operation"
                                minOccurs="1" maxOccurs="unbounded"/>
                        </xsd:sequence>
                    </xsd:complexType>
                    <!-- Unicité des opérations -->
                    <xsd:key name="opUnique">
                        <xsd:selector xpath="mat:operation" />
                        <xsd:field xpath="@opID" />
                        <xsd:field xpath="@opName" />
                        <xsd:field xpath="@opType" />
                    </xsd:key>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <!-- Définition du type complexe 'Operation' (une opération matricielle) -->
    <xsd:complexType name="Operation">
        <xsd:attribute name="opID" type="xsd:int" use="required"/>
        <xsd:attribute name="opName" type="xsd:string" use="required"/>
        <xsd:attribute name="opType" type="mat:MatOpType" use="required"/>
        <xsd:attribute name="left" type="xsd:string" use="required"/>
        <xsd:attribute name="right" type="xsd:string" use="required"/>
```

```
        <xsd:attribute name="result" type="xsd:string" use="required"/>
    </xsd:complexType>

    <!-- Définition du type complexe 'MatOpType' (types possibles pour une opération matricielle) -->
    <xsd:simpleType name="MatOpType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="add"/>
            <xsd:enumeration value="sub"/>
            <xsd:enumeration value="mul"/>
        </xsd:restriction>
    </xsd:simpleType>

</xsd:schema>
```

Figure VIII.7: Schema XML (`MatrixOperations.xsd`) modélisant une liste d'opérations matricielles entre deux matrices.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- INSTANCE 4 : des opérations sur des matrices -->
<matrixComputations
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns='https://www.timc.imag.fr/matrix'
    xsi:schemaLocation='https://www.timc.imag.fr/matrix MatrixOperations.xsd'>
    <matrices>
        <matrix matName="translate">
            <row rowNo="0">
                <cell cellNo="0" value="1"/><cell cellNo="1" value="0"/><cell cellNo="2" value="2"/>
            </row>
            <row rowNo="1">
                <cell cellNo="0" value="0"/><cell cellNo="1" value="1"/><cell cellNo="2" value="-1"/>
            </row>
            <row>
                <cell cellNo="0" value="0"/><cell cellNo="1" value="0"/><cell cellNo="2" value="1"/>
            </row>
        </matrix>
        <matrix matName="colVect1">
            <row rowNo="0">
                <cell cellNo="0" value="1"/>
            </row>
            <row rowNo="1">
                <cell cellNo="0" value="0"/>
            </row>
            <row>
                <cell cellNo="0" value="1"/>
            </row>
        </matrix>
        <matrix matName="rowVect">
            <row rowNo="0">
                <cell cellNo="0" value="1"/><cell cellNo="1" value="-1"/><cell cellNo="2" value="2"/>
            </row>
        </matrix>
    </matrices>
    <operations>
        <operation opID="1" opName="translation" opType="mul" left="translate" right="colVect1" result="colVect2"/>
        <operation opID="2" opName="scalarProduct" opType="mul" left="rowVect" right="colVect1" result="scalar"/>
```

```
39      </operations>
40  </matrixComputations>
```

Figure VIII.8: Document XML contraint par le schéma XML `MatrixOperations.xsd`.

1.e Les expression XPath des sélecteurs et des champs

Notez que les expressions XPath des sélecteurs et des champs (`xsd:selector` et `xsd:field`) sont restreintes:

- Avec le sélecteur, on ne peut sélectionner que des éléments descendants de l'élément dans lequel la clefs d'unicité a été déclarée.
- Le chemin XPath défini dans le champ est relatif aux éléments sélectionnés dans le sélecteur. Ce sont uniquement des éléments ou attributs descendants.
- Les seuls opérateurs disponibles sont l'opérateur de chemin `/` et l'opérateur d'union `|`.
- Les axes autorisés sont les axes enfants `/`, d'attribut `@` et l'axe descendant `.//`
- Aucun filtre n'est autorisé.

Attention à bien respecter les préfixes associés aux espaces de nom dans ces chemins XPath !

2 Contraintes d'existence

Une contrainte d'existence spécifie qu'il doit exister un élément ou attribut d'une propriété fixée dans un élément donné. Pour être plus précis, une contrainte d'existence permet de lier une contrainte d'unicité (ayant des propriétés de contrainte d'existence dans le cas des éléments `xsd:key`) à une valeur dont on requiert l'existence.

C'est l'élément `xsd:keyref` qui permet de déclarer et définir une contrainte d'existence.

L'exemple typique est celui d'une commande faite en ligne à partir d'un catalogue. Une contrainte d'existence peut imposer le fait qu'un produit commandé existe dans le catalogue en liant la référence du produit commandé à l'existence de sa référence dans le catalogue.

2.a Où déclarer une contrainte d'existence

Les éléments `xsd:keyref` doivent être *éléments fils* d'un élément ; cet élément contient ceux sur lesquel va s'appliquer la contrainte d'existence. Comme pour les contraintes d'unicité, la portée de la contrainte d'existence est définie à partir du niveau où elle est déclarée.

2.b Attributs de la contrainte

Les éléments `xsd:keyref` contiennent :

- un attribut `name` qui est requis
- un attribut `refer` qui est requis.
- deux types d'éléments:
  - `xsd:selector` qui est requis. Une seule occurence doit être présente.
  - `xsd:field`. Au moins une occurence doit être présente, mais il peut y en avoir plusieurs.

  qui ont chacun un seul élément `xpath`.

Ainsi, les éléments définissant une contrainte d'existence s'écrivent :

```
<xsd:keyref name="nom_de_la_clef" refer="nom_clef_d'unicité">
    <xsd:selector xpath="éléments sur lesquels porte l'existence" />
    <xsd:field xpath="éléments ou attributs spécifiant l'existence" />
    <xsd:field xpath="autres éléments/attributs ...." />
    ...
</xsd:unique>
```

Figure VIII.9: Structure d'une clef d'existence.

L'attribut `name` donne son nom à la contrainte d'existence.

La valeur de l'attribut `refer` contient le nom (attribut `name`) d'une clef d'unicité associée.

L'élément `xsd:selector`, via un chemin XPath spécifié dans son attribut `path`, spécifie sur quels éléments s'applique la contrainte d'existence. Ces éléments, de même type, peuvent être multiples : il peut s'agir d'un seul élément comme d'une liste d'élément.
L'élément `xsd:field` permet, via son chemin XPath spécifié dans son attribut `path`, de spécifier la valeur servant de clef, dont on doit vérifier l'existence par rapport à une clef d'unicité. Plusieurs champs `xsd:field` peuvent être présents pour forcer la vérification de l'existence de plusieurs valeurs en même temps.

La contrainte d'existence implique que pour chaque élément sélectionné (`selector`), il existe un élément sélectionné par la contrainte d'unicité qui a la même valeur.

Attention, la cardinalité des champs `field` doit être la même entre la clef d'existence et la clef unique à laquelle elle se réfère !

Attention, les clefs uniques générées appartiennent à l'espace de nom dans lequel elles ont été définies. Si, dans le schéma XML dans lequel elles sont définies les éléments sont préfixés, alors la clef unique à laquelle la clef d'existence se réfère (avec `refer`) doit être préfixée !

Attention, la contrainte d'unicité à laquelle se réfère la contrainte d'existence doit être contenue dans le même élément ou dans un de ses descendants.

2.c Un exemple complet

Reprenons le schéma XML spécifiant les opérations sur les matrices. Nous avons vu qu'une opération possédait 2 attributs `left` et `right` permettant de se référer au nom des matrices. En réalité, dans ce schéma montré figure VIII.7, rien ne garanti qu'une matrice dont le nom est donné dans les attributs `left` et `right` d'une opération existe réellement dans la liste des matrices instanciées ! Il faut donc ajouter une contrainte d'existence liant les valeurs de chacun des attributs `left` et `right` à l'existence des matrices, donc à leur nom donné par l'attribut `matName` présent dans chaque matrice.

Considérons d'abord la partie de schéma XML donnée ci-dessous dans la figure VIII.10. Supposons que nous disposions d'une clef d'unicité nommée `matUnique` qui garanti l'unicité des noms des matrices présentes dans la listes de matrices `mat:matrices`, alors nous pouvons nous servir de cette clef pour

vérifier l'existence du nom d'une matrice.
Il suffit alors de définir deux clefs d'existence que nous nommerons `leftMatExist` et `rightMatExist`. Chacune de ces clefs se réfère à la clef d'unicité `mat:matUnique` (attention au préfixe ! voir note ci-dessus) qui lui permet de connaitre l'existence du nom d'une matrice. Leur sélecteur applique la contrainte d'existence aux opérations `mat:operation` sur les valeurs `@left` et `@right` respectivement.

Il n'est cependant pas nécessaire de définir une clef d'existence pour vérifier que le nom de la matrice résultat existe dans la liste des matrices ; celle-ci, en tant que résultat, n'existe pas nécessairement encore.

L'exemple complet montrant l'inclusion de ces contraintes dans le schéma XML est donné plus bas, figure VIII.11.

On remarque (1) que la contrainte d'unicité `matUnique` est définie dans `mat:matrices`, un élément descendant de `mat:matrixComputations` dans lequel se trouvent définies les contraintes d'existence `leftMatExist` et `rightMatExist` et (2) l'usage du préfixe `mat` pour se référer à la contrainte d'unicité `mat:matUnique` dans les contraintes d'existence.

```
1        <!-- Unicité des matrices -->
2        <xsd:key name="matUnique">
3            <xsd:selector xpath="mat:matrix"/>
4            <xsd:field xpath="@matName"/>
5        </xsd:key>
```

```
1        <!-- Existence des matrices gauche et droite appelées dans les opé
             rations -->
2        <xsd:keyref name="leftMatExist" refer="mat:matUnique">
3            <xsd:selector xpath="mat:operations/mat:operation"/>
4            <xsd:field xpath="@left"/>
5        </xsd:keyref>
6        <xsd:keyref name="rightMatExist" refer="mat:matUnique">
7            <xsd:selector xpath="mat:operations/mat:operation"/>
8            <xsd:field xpath="@right"/>
9        </xsd:keyref>
```

Figure VIII.10: Définition de deux clefs d'existence se référant à la clef d'unicité `matUnique`.

```
1  <?xml version="1.0"?>
2  <xsd:schema version="1.0"
3              targetNamespace="https://www.timc.imag.fr/matrix"
4              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5              xmlns:mat="https://www.timc.imag.fr/matrix"
6              elementFormDefault="qualified">
7      <xsd:include schemaLocation="./Matrix.xsd"/>
8
9      <!-- Elément racine 'matrixComputation' -->
10     <xsd:element name="matrixComputations">
11         <xsd:complexType>
12             <xsd:sequence>
13
14             <!-- liste de matrices -->
15                 <xsd:element name="matrices" maxOccurs="unbounded">
16                     <xsd:complexType>
17                         <xsd:sequence>
18                             <xsd:element name="matrix" type="mat:Matrix"
                                   minOccurs="1" maxOccurs="unbounded"/>
```

```
19                          </xsd:sequence>
20                      </xsd:complexType>
21                      <!-- Unicité des matrices -->
22                      <xsd:key name="matUnique">
23                          <xsd:selector xpath="mat:matrix"/>
24                          <xsd:field xpath="@matName"/>
25                      </xsd:key>
26                  </xsd:element>
27
28              <!-- liste d'operations -->
29                  <xsd:element name="operations" maxOccurs="unbounded">
30                  <!-- code inchangé (voir précédent Schema XML) -->
31                  </xsd:element>
32
33              </xsd:sequence>
34          </xsd:complexType>
35
36          <!-- Existence des matrices gauche et droite appelées dans les opé
                rations -->
37          <xsd:keyref name="leftMatExist" refer="mat:matUnique">
38              <xsd:selector xpath="mat:operations/mat:operation"/>
39              <xsd:field xpath="@left"/>
40          </xsd:keyref>
41          <xsd:keyref name="rightMatExist" refer="mat:matUnique">
42              <xsd:selector xpath="mat:operations/mat:operation"/>
43              <xsd:field xpath="@right"/>
44          </xsd:keyref>
45
46      </xsd:element>
47
48      <!-- Définition du type complexe 'Operation' (une opération matricielle) --
            >
49      <!-- code inchangé (voir précédent Schema XML) -->
50
51      <!-- Définition du type complexe 'MatOpType' (types possibles pour une opé
            ration matricielle) -->
52      <!-- code inchangé (voir précédent Schema XML) -->
53
54  </xsd:schema>
```

Figure VIII.11: Schema XML (`MatrixOperation.xsd`) *modifié* modélisant une liste d'opérations matricielles entre deux matrices et intégrant des clefs d'existence.