Méta représentation EXPRESS des schémas XML pour la validation de documents XML

Jérôme Chochon, Eric Sardet, Guy Pierra
Laboratoire d'Informatique Scientifique et Industrielle
Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
1, avenue Clément Ader, 86961 Futuroscope
chochon@ensma.fr, sardet@ensma.fr, pierra@ensma.fr

Résumé

*Initialement utilisé comme format d'échange pour les documents, le langage XML (*eXtensible Markup Language*) tend à se développer dans le domaine des bases de données. Ceci impose de contrôler l'intégrité de l'information véhiculée. Pour satisfaire cette contrainte, diverses variantes de modèles de documents regroupés sous le nom de schémas XML ont été définies. Sur les nombreux formalismes ainsi proposés au W3C, le langage lui-même appelé 'XML Schema', grâce à ses concepts proches des formalismes de modélisation, possède de fortes chances de s'imposer. Cependant, aucun outil ne permet aujourd'hui de valider totalement ces schémas, ni la conformité des documents par rapport à ces schémas. La solution proposée dans ce papier est la mise en œuvre d'un tel processus de validation dans un environnement adapté, celui offert par les formalismes de spécification de données, et en particulier le langage EXPRESS.*

Mots clés

XML, XML Schema, EXPRESS, méta modélisation, validation de documents

1 Introduction

L'utilisation de la représentation informatique des données techniques tout au long de leur cycle de vie -conception, production, maintenance- impose de définir une représentation traitable par machine et un format d'échange entre systèmes. Dans ce but, deux approches différentes se sont développées au cours de ces dernières années.

La première correspond aux échanges entre bases de données. La cohérence et l'intégrité de l'information sont mis en avant au travers des notions de typage fort, de cardinalité des associations et des contraintes sur les données telles que l'unicité ou les contraintes ensemblistes. Cette sémantique est exprimée par les formalismes relationnels tels que E/R [1], NIAM [2], et les formalismes orientés objet, comme OMT [3], UML [4] et surtout EXPRESS [5] [6], utilisé dans le domaine technique.

La seconde correspond plus à des échanges d'informations entre humains. Elle utilise une présentation des données de type documentaire afin qu'elles soient compréhensibles par l'homme en s'appuyant sur deux concepts fondamentaux : la description explicite de l'organisation structurelle des documents d'une part, et la séparation de la structure et de la présentation d'autre part. Les langages tels que SGML [7], HTML [8] et XML [9] illustrent parfaitement cette approche.

Actuellement, les applications tendent à utiliser XML pour l'échange d'informations entre bases de données informatiques. Il est donc nécessaire d'assurer l'intégrité de cette information (exemple : le typage des données) afin d'assurer la qualité des données échangées. Ainsi, on a vu apparaître une nouvelle approche pour définir des structures documentaires utilisables pour échanger des informations entre bases de données : les différentes variantes de schémas XML (XML-Data [13], DCD [2],...) et plus particulièrement la version XML Schema [10] [11] [12].

L'objectif de ce papier est de proposer une approche pour la validation de documents XML conformément à leur schéma XML sous-jacent. En effet, il n'existe pas aujourd'hui d'outils de validation implémentant totalement la spécification XML Schema. Il est donc impossible de s'assurer de l'intégrité des données dans un monde purement XML. Nous proposons donc de mettre en œuvre une solution s'appuyant sur la puissance d'expression des langages de spécification de données et en particulier du langage EXPRESS. L'intérêt est de bénéficier des outils de validation de modèles de données disponibles, et, à terme, de pouvoir enrichir le pouvoir d'expression des XML Schema.

Le contenu de cet article est le suivant. Dans un premier temps, nous présentons les deux langages utilisés, XML Schema et EXPRESS. Une deuxième partie exposera l'intégration des univers XML Schema et EXPRESS et l'illustrera au travers de la présentation d'une implémentation d'un outil de validation.

2 Présentation des langages

L'approche proposée consiste à intégrer les concepts d'un langage (orienté modèle) de description de structures documentaires, XML Schema, dans un environnement de spécification de données, et plus précisément le langage EXPRESS.

2.1 Les langages XML et XML Schema

Les DTD permettent de définir la structure d'un document comme par exemple étant constitué de chapitres, eux-mêmes constitués de sections, elles-mêmes constituées de paragraphes... En

voulant étendre XML comme format d'échange entre bases de données, nous sommes confrontés à la notion d'intégrité de l'information. Les modèles sous-jacents ne doivent pas seulement définir une structure documentaire mais doivent aussi respecter des contraintes sémantiques fortes. Plusieurs langages ont été proposés : XML-Data [13], DCD [2], SOX [15], XML Schema [10] [11] [12], Schematron [15],...

Soutenu par le W3C, le langage XML Schema tend à devenir la référence. Avant de présenter ce langage, nous allons introduire la notion de balisage.

2.1.1 Les langages de balisage

La transmission écrite d'informations pour l'être humain est représentée par une suite de mots, composés de caractères, respectant une certaine structure grammaticale définie par le langage employé. La compréhension par le récepteur se fait par référence à ce même langage. Par contre, si nous transférons sans traitement particulier cette phrase entre deux ordinateurs, le récepteur ne pourra interpréter l'information que comme étant une chaîne de caractères. Les langages de balisage permettent de marquer les chaînes de caractères selon un principe de parenthésage (ou de balisage), ce qui en facilite le traitement. L'Illustration 1 présente la description d'une personne telle qu'elle peut être saisie dans un document texte.

Dupont Francis 23 ans 86000 Poitiers

Illustration 1 : Description d'une personne

En appliquant des balises (Illustration 2), l'information peut maintenant être identifiée et traitée par un programme. Ce document respecte les règles définies par SGML.

```
<PERSONNE>
   <SON_NOM>Dupont</SON_NOM>
   <SON_PRENOM>Francis</SON_PRENOM>
   <SON_AGE>23</SON_AGE>
   <SON_ADRESSE>
      <SON_CP>86000</SON_CP>
      <SA_VILLE>Poitiers</SA_VILLE>
   </SON_ADRESSE>
</PERSONNE>
```

Illustration 2 : Exemple 1 sous forme XML

Normalisé standard ISO 8879 en 1986, SGML [7] (*Standard Generalized Markup Language)* modélise, au travers de fichiers textes, les documents en séparant les données et le style. Les données sont stockées dans un document et plusieurs documents de style, appelé feuilles de style, permettent autant de représentations souhaitées. Les balises utilisées et leurs règles d'enchaînement sont définies dans une DTD (Définition de Type de Document). Ce troisième document agit comme un modèle dont les documents sont les instances. Mais, la complexité du langage l'a restreint dans sa diffusion.

Au début des années 1990, pour permettre l'échange d'informations via le Web, une application spécifique de SGML est créée. Nommée HTML [8], il s'agit plus particulièrement d'une DTD SGML. Les documents HTML n'ont cependant qu'une orientation présentation, rien ne permettant de qualifier sémantiquement l'information. De plus, il est impossible de créer de nouvelles balises.

Pour permettre aux utilisateurs d'échanger des documents orientés données, et non plus orientés présentation, le W3C, organisme gérant les technologies liées à Internet, s'est concentré vers une restriction du langage SGML portant le nom d'XML [9]. Il est ainsi possible de définir de nouveaux ensembles de balises (ou applications XML) pour l'échange de données sur le web.

2.1.2 Présentation du langage XML Schema

Un schéma désigne un modèle et le document XML est considéré comme une instance de ce modèle. Les concepts représentés dans un document XML sont :

? Les balises : présentes par paires -une entrante (**<BALISE>**) et une fermante (**</BALISE>**), elles sont identifiées par un nom, éventuellement des attributs et possèdent un contenu qui peut être une chaîne de caractères ou d'autres balises. Certaines sont déclarées comme vides (**<BALISE_VIDE/>**) ;

? Les attributs : ils sont inclus dans la balise et la qualifient (**<BALISE ATTRIBUT=**'valeur'**>**).

Un schéma XML Schema permet de modéliser ces concepts.

2.1.3 Concepts de base

Le modèle d'une balise se nomme un élément (mot clé **ELEMENT**). Celui-ci est qualifié par un nom et associé à un type qui définit le contenu de la balise résultante. Un élément ne contenant pas d'autres éléments ou attributs est déclaré de *type simple*. Dès lors qu'il possède des attributs ou d'autres éléments -voire les deux-, il est considéré comme étant de *type complexe.*

Le modèle d'un attribut est défini par le mot clé **ATTRIBUTE** en XML Schema, et caractérisé par son nom, son type (qui ne peut être que simple) et des contraintes d'occurrences exprimées par les attributs optionnels **USE** et **VALUE**. Les attributs peuvent être regroupés pour être utilisés par plusieurs éléments. Le groupe est identifié par un nom (**ATTRIBUTEGROUP NAME**) lors de sa déclaration puis référencé (**ATTRIBUTEGROUP REF**) dans les éléments qui vont les utiliser.

Ces deux concepts, élément et attribut, représentent les seuls objets instanciables dans XML Schema.

XML Schema comporte une bibliothèque de types prédéfinis. Ils peuvent être réutilisés ou spécialisés pour restreindre le domaine de valeurs au travers de *facettes* (**MININCLUSIVE**, **MAXINCLUSIVE**, **LENGTH**, **ENUMERATION**...). De plus, il est aussi possible d'unir des domaines (mot clé **UNIONMEMBER TYPES**).

Le type complexe est similaire au type RECORD de certains langages de programmation ou aux classes des langages à objets. Il s'agit d'un type comprenant d'autres propriétés telles que des attributs ou des éléments (appelés sous-éléments dans ce cas). Cependant, par rapport au RECORD, le type complexe de XML Schema rajoute la notion d'ordre au moyen de connecteurs.

? Le connecteur **SEQUENCE** définit une suite d'éléments. Ces sous-éléments doivent apparaître dans le document XML dans le même ordre que la déclaration ;

? Le connecteur **CHOICE** définit un choix d'apparition. Si trois éléments sont déclarés sous le connecteur, seul l'un des trois doit apparaître dans le document XML ;

? Le connecteur **ALL** impose aux éléments d'apparaître dans le document XML une fois au maximum, mais n'impose aucun ordre.

Les sous-éléments d'un type complexe peuvent être enrichis d'occurrences pour définir le nombre de fois où ils doivent apparaître dans le document XML. Les attributs **MINOCCURS** et **MAXOCCURS** viennent compléter ainsi leur définition.

L'Illustration 3 montre le schéma correspondant à l'exemple de document XML représenté dans Illustration 2.

```
<ELEMENT    NAME='PERSONNE'
            TYPE='T_PERSONNE'/>
<COMPLEXTYPE NAME='T_PERSONNE'>
  <SEQUENCE>
    <ELEMENT  NAME='son_nom' TYPE='STRING'/>
    <ELEMENT  NAME='son_prenom' TYPE='STRING'/>
    <ELEMENT  NAME='son_age' TYPE='INTEGER'/>
    <ELEMENT  NAME='son_adresse' TYPE='T_ADRESSE'/>
  </SEQUENCE>
</COMPLEXTYPE>
<COMPLEXTYPE NAME='T_ADRESSE'>
  <SEQUENCE>
    <ELEMENT  NAME='son_CP' TYPE='INTEGER'/>
    <ELEMENT  NAME='sa_ville' TYPE='STRING'/>
  </SEQUENCE>
</COMPLEXTYPE>
```

Illustration 3 : Schéma XML Schema définissant une personne

2.1.4 Concepts avancés

XML Schema supporte l'héritage (ou dérivation). Celui-ci s'applique aux types. Il peut-être utilisé par restriction (utilisation de facettes pour les types simples ou redéfinition du type d'un sous-élément ou d'un attribut pour les types complexes) ou par extension (ajout de sous-éléments et/ou des attributs pour les deux types). De plus, l'héritage est contrôlé par deux attributs : **FINAL** et **BLOCK.**

- **FINAL** interdit de nouvelles dérivations dans un schéma ;
- **BLOCK** autorise la création de type par dérivation mais empêche leur utilisation dans le cadre d'une relation polymorphe.

Ces attributs prennent les valeurs de **EXTENSION**, **RESTRICTION** ou **ALL** suivant que l'on souhaite contrôler la dérivation par extension, par restriction ou pour les deux.

De plus, XML Schema permet d'exprimer des contraintes d'unicité ou des contraintes référentielles.

- La contrainte *d'unicité* (mot clé **UNIQUE**) permet d'assurer que la valeur d'un attribut, quel que soit son type, est unique au sein d'un document XML ;
- La contrainte *référentielle* (mot clé **KEY** et **KEYREF**) autorise la création d'une clé étrangère au sein d'un type complexe pour référencer via un attribut un autre élément.

2.1.5 Outils

De nombreux outils permettent l'édition de documents XML. Par contre, très peu d'outils permettent de valider le document par rapport à son schéma. Notons enfin que ces outils ne sont que des implémentations partielles de la spécification XML Schema.

2.1.6 Conclusion

Le langage XML permet d'échanger des fichiers de données fortement structurés et compréhensibles pour l'être humain. Les schémas viennent en définir cette structure et appliquer des contraintes aux données. Cependant, aucun outil ne prend en charge la validation de façon complète (et donc exploitable) des documents par rapport à leur schéma. L'idée est donc de transférer le document et son schéma dans un autre langage disposant d'outils de vérification pour le valider. Dans notre cas, nous utiliserons le langage EXPRESS.

2.2 Le Langage EXPRESS

Ce langage fut développé dans les années 80 dans le cadre du projet STEP (*Standard for the Exchange of Product model and data*). Son objectif initial est la normalisation des modèles de données pour pouvoir les utiliser sur des plate-formes hétérogènes et surtout pouvoir les échanger et les réinterpréter sans difficultés. Le langage EXPRESS est utilisé pour exprimer formellement ces données.

Il s'agit avant tout d'un formalisme textuel associé à une version graphique nommée EXPRESS-G. Le format textuel en fait toute sa puissance d'expression par rapport aux autres formalismes : il est compilable et génère des modèles de données pour les langages C++, JAVA,...

2.2.1 Les éléments de base

Le langage EXPRESS [5] [6] dispose d'éléments de base servant à la modélisation d'un univers du discours appelé '**SCHEMA**' : les entités, les attributs et les types.

Une entité désigne un élément identifiable dans le domaine étudié et possédant une existence autonome. Des relations sont exprimées entre ces entités : l'héritage et l'association au travers d'attributs. La notion d'abstraction d'une entité n'autorise l'instanciation qu'au travers de ses sous-entités.

Chaque entité, outre son nom, est décrite par un ou plusieurs d'attributs. Leurs domaines de valeurs vont être des domaines simples (booléens, réels, entiers, chaînes de caractères), des collections (agrégats), des entités (on rejoint alors le terme d'association) ou des domaines construits à partir d'un type utilisateur. Ces derniers sont créés à partir d'autres types par restriction de leurs domaines (application d'une règle locale), union (types sélectifs **SELECT**) ou énumération de valeurs (types énumérés **ENUMERATION**). Ils possèdent plusieurs statuts dans leurs déclarations.

- Libre: indépendant vis à vis des autres attributs et sa valeur (obligatoire) dépend d'une saisie utilisateur ;
- Optionnel (**OPTIONAL**) : la présence de valeur n'est pas obligatoire. Un attribut déclaré optionnel dans une super classe, reste optionnel dans toutes les sous-classes sauf s'il est redéfini ;
- Dérivé (**DERIVE**) : la valeur de l'attribut est calculée par une fonction. Il est hérité par toutes les sous-classes de la classe où il est déclaré (Illustration 4). Le processus de validation de document XML par rapport à son schéma décrit en XML Schema va s'appuyer fortement sur ce concept.

2.2.2 Les contraintes

Les contraintes peuvent être appréhendées selon deux grandes familles : les contraintes locales qui s'appliquent uniquement sur chacun des instances de l'entité d'où elles sont issues (**WHERE**) ou les contraintes globales qui apportent une vérification globale sur l'ensemble des instances d'un type donné (**UNIQUE**, **INVERSE** et **RULE**).

Les contraintes locales (**WHERE**) sont définies au travers de prédicats auxquels chaque instance de l'entité où elles sont déclarées doit obéir. Ces prédicats permettent par exemple de limiter la valeur d'un attribut en définissant un domaine de valeurs ou encore de refuser la valuation d'un attribut selon certains critères.

La contrainte d'unicité (**UNIQUE**) contrôle l'ensemble d'une population d'instances issues d'une même entité pour s'assurer que l'attribut auquel s'applique la contrainte possède une valeur unique sur toute la population.

La contrainte de cardinalité (**INVERSE**), appelée aussi contrainte d'existence, permet de regrouper au sein d'un agrégat les entités d'un certain type qui référencent une entité donnée dans un certain rôle. L'attribut inverse exprime l'association entre une entité sujet (celle où l'attribut est déclaré) et des entités référençant l'entité sujet par un attribut.

Les règles globales (**RULE**) ne sont pas déclarées au sein des entités mais sont définies séparément. Elles permettent de vérifier des règles qui s'appliquent à un ensemble d'instances d'un type donné.

Pour faciliter la manipulation des données dans la population d'instances, des fonctions (Illustration 4) et des procédures peuvent être décrites (mot clés **FUNCTION** et **PROCEDURE**) au travers d'un langage de description s'apparentant au langage PASCAL. Le langage EXPRESS propose aussi tout un ensemble de fonctions prédéfinies : **QUERY** (requête sur les instances), **SIZEOF** (taille d'une collection),...

```
ENTITY Personne;
   son_nom : STRING;
   son_prenom : STRING;
   son_age : INTEGER;
   son_adresse : Adresse;
DERIVE
   son_ID : STRING:=donneID(SELF);
END_ENTITY;

ENTITY Adresse;
   son_CP : INTEGER;
   sa_ville : STRING;
END_ENTITY;

FUNCTION donneID(laPersonne : Personne):STRING;
LOCAL
   l_ID : STRING;
   le_prenom : STRING :=laPersonne.son_prenom;
   le_nom : STRING :=laPersonne.son_nom;
END_LOCAL;
   l_ID:=le_nom[1]+le_prenom[1];
   RETURN(l_ID);
END_FUNCTION;
```

Illustration 4 : Schéma EXPRESS définissant une personne

2.2.3 Le fichier d'échange

Une population de données conforme à un modèle de données EXPRESS est représentée au sein d'un fichier physique (Illustration 5). Celui-ci est un fichier texte répondant à une syntaxe stricte [ISO10303-21] précisant en particulier les règles de traduction de définitions EXPRESS pour décrire des instances.

```
#1=PERSONNE('Dupont','Francis',23);
#2=ADRESSE(86000,'Poitiers');
```

Illustration 5 : Fichier d'instances d'une personne

Une instance fait référence à l'entité dont elle est issue par son nom et un identifiant (#i), suivi d'un nombre unique et contient la suite de valeurs de ses attributs entre parenthèses. Chaque attribut respecte son type défini dans le modèle. Les attributs dérivés, tout comme les attributs inverses sont calculés par le système receveur et ne sont pas présents dans le fichier. Un agrégat se présente sous la forme d'une liste d'attributs comprise entre parenthèses. Un attribut déclaré optionnel ne possédant aucune valeur est représenté par le symbole $.

Un tel fichier permet ainsi à deux systèmes informatiques d'échanger sans aucune ambiguïté des populations d'instances conformes à un modèle EXPRESS.

3 Intégration d'un modèle XML Schema dans un environnement EXPRESS

Le but est de modéliser les concepts XML Schema dans un langage de modélisation de données, EXPRESS, pour utiliser les outils afin de valider un document XML conforme à son schéma.

3.1 Mapping direct

Cette approche consiste à exprimer chacun des concepts XML Schema en EXPRESS, par une transformation systématique. En d'autres termes, il faut définir une représentation EXPRESS aux notions d'**ELEMENT**, d'**ATTRIBUTE**, de **SIMPLETYPE**, de **COMPLEXTYPE**.

Ce problème a déjà été posé et étudié, soit au travers de la transformation de documents SGML et de ses DTD en EXPRESS [17], soit au travers de la transformation de schémas XML Schema [18] et de DTD [19] vers un langage orienté objets ou encore vers des bases de données relationnelles.

Les conclusions sont les suivantes. La représentation directe des types prédéfinis de XML Schema ne pose aucun problème. Cependant, le modèle de contenu d'un type complexe, et surtout la souplesse qu'il apporte tout en assurant une rigueur sur l'ordre d'apparition des balises et, éventuellement, leur répétition multiple sont difficilement représentables directement.

Pour le langage EXPRESS, les problèmes rencontrés restent les mêmes. L'**ELEMENT** est l'objet identifiable et ayant une existence propre en XML Schema ; sa représentation EXPRESS est donc l'**ENTITY**. Les types simples ont leur représentation directe (**TYPE**), mais la relation d'héritage n'est plus exprimable. La remarque concernant le modèle de contenu d'un type complexe est toujours valable, tout en sachant que le type complexe n'existe pas en EXPRESS. Par contre, les contraintes telles que les restrictions des domaines de valeurs et l'unicité sont prises en charge, sachant que le pouvoir d'expression d'EXPRESS est supérieur à celui du langage XML Schema.

Les limites du mapping direct nous conduisent à envisager une seconde approche, plus abstraite : la méta représentation des concepts XML Schema en EXPRESS.

3.2 Principe de la Méta modélisation

Une transformation directe, systématique et automatique d'un schéma XML dans un modèle EXPRESS s'avère difficilement envisageable. Cependant une alternative existe et va consister à représenter non plus le schéma XML directement, mais plutôt à s'appuyer sur sa méta représentation. Un schéma XML particulier sera alors représenté non pas par un modèle EXPRESS particulier mais par un ensemble d'instances d'un (méta-) modèle EXPRESS général.

3.2.1 Principe

Les concepts XML Schema sont définis au niveau de la grammaire du langage, ou en termes documentaires, au niveau de sa DTD. L'idée est donc de non plus définir un modèle EXPRESS équivalent à un schéma XML Schema particulier, mais plutôt de définir un (méta-) modèle EXPRESS des concepts de XML Schema (Illustration 6). L'instanciation en EXPRESS de ce méta modèle définira un modèle XML Schema particulier.

De plus, si l'on veut valider un document XML conformément à son XML Schema sous-jacent, on doit rendre disponible dans un environnement EXPRESS le document XML. Pour cela nous proposons de définir un modèle de document XML en terme d'un modèle EXPRESS (Illustration 6).

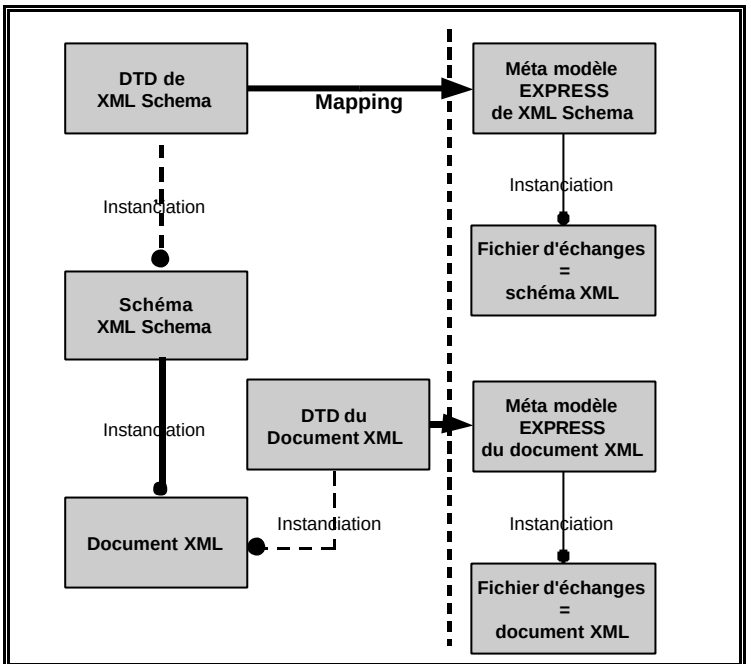

Illustration 6 : Représentation d'un modèle EXPRESS d'un document XML

L'ensemble des données étant maintenant disponible dans l'environnement EXPRESS, valider un document XML par rapport à son schéma XML Schema nécessite :

? De s'assurer que le schéma XML Schema est lui-même valide ;

? De définir une fonction EXPRESS d'évaluation de la validité d'un document XML.

3.2.2 Modélisation d'un schéma XML

Un schéma définit des éléments et des types, simples et complexes. Chaque **`ELEMENT`** ou type (**`SIMPLETYPE`**, **`COMPLEXTYPE`**) est qualifié par un nom `its_name`. L'**`ELEMENT`** est associé à un type via un attribut (`its_type`). L'Illustration 7 montre une représentation simplifiée en EXPRESS-G de la méta modélisation d'un **`ELEMENT`** XML Schema. Les boites représentent les entités. Celles possédant une barre verticale sur la droite représentent des types atomiques du langage EXPRESS. Chaque trait gras symbolise l'héritage entre deux classes (orienté par un rond). Les traits fins qualifiés par un nom représentent une association.

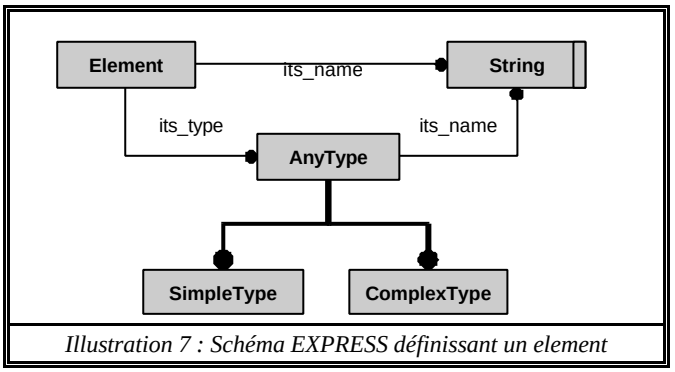

Illustration 7 : Schéma EXPRESS définissant un element

Les types simples sont quant à eux décrits par un contenu (`its_content`), indiquant leur type de dérivation (Illustration 8). Trois dérivations sont proposées : restriction (redéfinition d'un domaine de valeurs), union (union de domaines de valeurs) ou liste (suite ordonnée de valeurs). A noter que le type restriction, sans ajout de facettes, agit seulement comme une redéfinition du type.

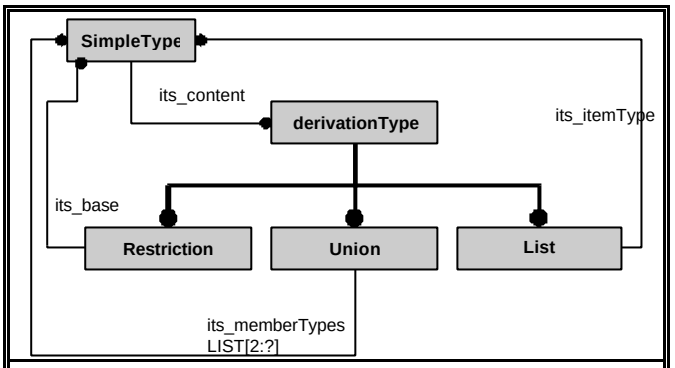

Illustration 8 : Schéma EXPRESS des types simples

Enfin, les types complexes sont décrits par un attribut `its_content`. Le **`SIMPLECONTENT`** et **`COMPLEXCONTENT`** n'interviennent que lors de la dérivation. Dans ce cas, **`SIMPLECONTENT`** indique que le type de base est de type simple et qu'aucun sous-élément ne sera ajouté. Dans tous les autres cas (ajouts d'**`ELEMENT`** ou redéfinition du type de certain **`ELEMENT`**), **`COMPLEXCONTENT`** est employé. Pour définir un modèle de contenu sans dérivation, les connecteurs **`CHOICE`**, **`SEQUENCE`** ou **`ALL`** définissent l'ordre d'apparition des sous-éléments (Illustration 9).

Sur l'illustration, les liens en pointillés indiquent une association de nature optionnelle.

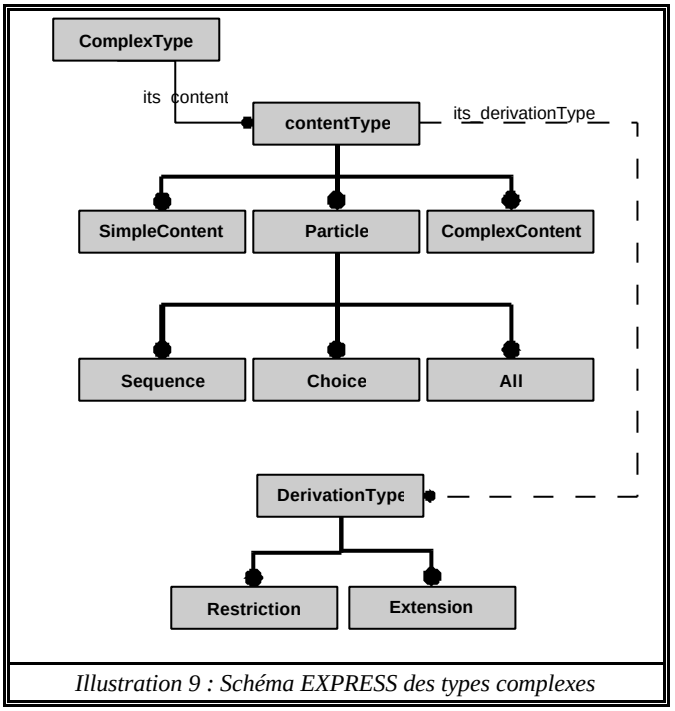

Illustration 9 : Schéma EXPRESS des types complexes

3.2.3 Modélisation d'un document XML

L'objectif est de modéliser un document XML en termes d'un modèle EXPRESS, ceci à partir des concepts sous-jacents aux documents XML. Un document XML est défini comme un arbre où les feuilles représentent les éléments d'informations (des chaînes de caractères) et où les nœuds définissent la structure du document. Ainsi, un **`DOCUMENT`** se caractérisera par une **`BALISE`** racine, associée par la propriété `its_root`. Cette **`BALISE`** contiendra d'autres **`BALISE`**s ou simplement une chaîne de caractères (**`STRING`**). De plus, elle sera qualifiée par une propriété, `its_name` qui définit le nom de la balise. La propriété `its_content` sera une liste de types utilisateurs **`BS`**, un type **`SELECT`**

(union de types EXPRESS) créé pour unir l'entité **BALISE** et le type chaîne de caractères (Illustration 10).

Dans l'illustration, les boites possédant des bords en pointillés indiquent un type utilisateur. De plus, la barre verticale en pointillés à droite indique que le type utilisateur (ici **BS**) est un type **SELECT** (ou union de types).

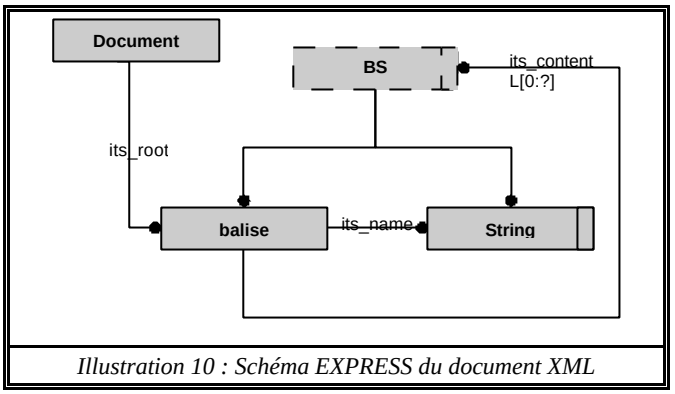

Illustration 10 : Schéma EXPRESS du document XML

La représentation des concepts XML Schema et des documents XML dans un environnement EXPRESS étant achevée, il faut maintenant définir un processus de validation de documents XML par rapport à leur modèle XML Schema sous-jacent.

3.3 Implémentation d'un outil de validation

Cette application consiste à prendre en entrée un document XML, accompagné de son schéma XML Schema, puis vérifier si ce document XML est valide par rapport à son schéma (Illustration 11).

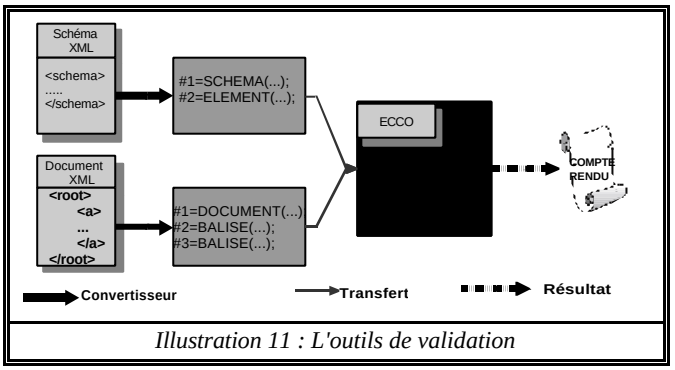

Illustration 11 : L'outils de validation

Le schéma XML et le document XML sont tout d'abord convertis en fichier d'échanges EXPRESS conformes respectivement au modèle EXPRESS d'un schéma XML et au modèle EXPRESS d'un document XML. Cette étape de transformation ne pose pas de difficultés, puisque les modèles sources (XML Schema) et cible (modèle EXPRESS) représentent les mêmes concepts. Il s'agira donc essentiellement d'une transformation syntaxique. Par la suite, ces deux fichiers sont importés dans un outil de vérification de modèle EXPRESS, ECCO [20], puis validés en fonction de leurs modèles EXPRESS. En créant des fonctions de contrôle qui permettent de respecter les critères de validation d'un document XML par rapport à son schéma, l'environnement ECCO produira un compte rendu sur la validité du document.

3.3.1 Méthodes de validation d'un contenu

L'outil doit permettre dans un premier temps de valider l'ordre d'enchaînement des balises. Les quatre critères suivants se doivent d'être respectés pour qu'un document XML soit valide par rapport à sa définition.

- Le document doit être bien formé, c'est à dire respecter les contraintes syntaxiques du langage XML : par exemple, tout balisage du document doit être composé d'une balise de début et d'une balise de fin. Cette vérification est effectuée en amont lors de la conversion du document XML en une population d'instances EXPRESS ;
- Chaque balise doit posséder un modèle : il faut rechercher dans le schéma associé au document si un **ELEMENT** possédant le même nom existe. Si aucun modèle n'est présent, alors la balise n'est pas valide.
- Le contenu d'une balise doit être valide par rapport au modèle de contenu défini dans le XML Schema associé. Si la balise contient seulement une chaîne de caractères, celle-ci doit être conforme au type défini dans le modèle (comme **SIMPLETYPE**). Par contre, si elle contient d'autres balises, alors l'ordre d'apparition de ces sous balises doit être vérifié par rapport au modèle de contenu défini dans le modèle (comme **COMPLEXTYPE**). Les différents ordres d'apparition des balises seront modélisés sous une forme en permettant la validation. Ce point sera développé dans la section suivante.
- Enfin, les sous balises de la balise courante doivent être elles aussi valides.

La description EXPRESS d'une balise est donc complétée d'un attribut dérivé qui définit son modèle de contenu, puis de trois autres attributs dérivés de type **BOOLEAN** pour valider les trois conditions de validité, et enfin d'un dernier attribut dérivé indiquant que la balise est valide ou non. Des fonctions, non décrites dans l'exemple (Illustration 12), valuent ces nouveaux attributs.

```
ENTITY Balise;
   its_name : STRING;
   its_content : LIST [0:?] OF BS;
DERIVE
   its_model:ELEMENT :=giveModel(SELF);
   have_model:BOOLEAN:=checkModel(SELF);
   is_Conform:BOOLEAN:=checkConformity(SELF);
   have_Valid_Children:BOOLEAN :=checkChildren(SELF);
   is_valid : boolean:=checkValidity(SELF);
END_ENTITY;
```

Illustration 12 : Définition d'une balise en EXPRESS

3.3.2 Représentation du modèle de contenu

Une **BALISE** est liée à l'**ELEMENT** qui défini son modèle, au moyen d'un attribut dérivé (`its_model`). L'ELEMENT est caractérisé par un **ANYTYPE**. Dans le cas d'un type complexe, le premier souci est d'en définir une représentation de son modèle de contenu, c'est à dire de tous les enchaînements de balises possibles dans un document XML conformément à sa définition.

Prenons l'exemple (Illustration 13) d'un **ELEMENT** `root` de type complexe possédant un modèle de contenu. Ce dernier est défini comme un choix entre deux séquences possibles : `a` puis `b`, ou `c` puis `d`. Notons que l'élément `d` est seulement référencé : cela signifie que sa description est externe à son utilisation. Il est décrit par un modèle de contenu définissant à nouveau un choix entre deux séquences de balises possibles : `z`, ou `y` puis `w`.

```
<ELEMENT NAME='root'>
  <COMPLEXTYPE>
    <SEQUENCE>
      <ELEMENT NAME='x' TYPE='..'/>
      <CHOICE>
        <SEQUENCE>
          <ELEMENT NAME='a' TYPE='..'/>
          <ELEMENT NAME='b' TYPE='..'/>
        </SEQUENCE>
        <SEQUENCE>
          <ELEMENT NAME='c' TYPE='..'/>
          <ELEMENT REF='d'>
        </SEQUENCE>
      </CHOICE>
      <ELEMENT NAME='f' TYPE='..'/>
    </SEQUENCE>
  </COMPLEXTYPE>
</ELEMENT>
<ELEMENT NAME='D'>
  <COMPLEXTYPE>
    <CHOICE>
      <ELEMENT NAME='Z' TYPE='..'/>
      <SEQUENCE>
        <ELEMENT NAME='Y' TYPE='..'/>
        <ELEMENT NAME='W' TYPE='..'/>
      </SEQUENCE>
    </CHOICE>
  </COMPLEXTYPE>
</ELEMENT>
```

Illustration 13 : Exemple de schémas pour valider l'ordre

L'Illustration 14 montre deux documents XML valides conformément au schéma précédent.

```
<ROOT>
  <X></X>
  <A></A>
  <B></B>
  <F></F>
</ROOT>
```

```
<ROOT>
  <X></X>
  <C></C>
  <D>
    <Y></Y>
    <W></W>
  </D>
  <F></F>
</ROOT>
```

Illustration 14 : Instances possibles (Illustration 14)

L'algorithme de validation nécessite donc une représentation de l'ensemble des séquences de balises possibles conformément à la définition de leur modèle de contenu. Pour cela, nous proposons d'utiliser un automate à états finis.

3.3.3 Utilisation d'un automate

Un automate à états finis est défini de la façon suivante :

- Un alphabet A ;
- Un ensemble fini d'états E ;
- Une relation de transition ExAxE ;
- D'un état initial I appartenant à E ;
- d'un ensemble d'états finaux F inclus dans E.

Une transition (*e, a, e'*) dite de l'état *e* vers l'état *e'* et étiquetée par le symbole *a* est notée :

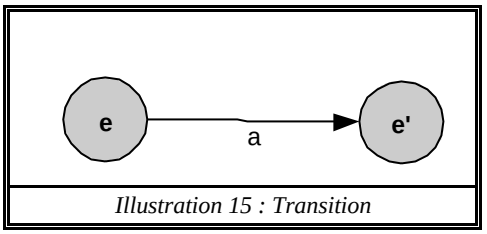

Illustration 15 : Transition

Un calcul de cet automate est une suite de transition. Ce calcul est réussi si le dernier état e_n est un état final. On dit alors que le mot a_1 a_2... a_n est reconnu par l'automate.

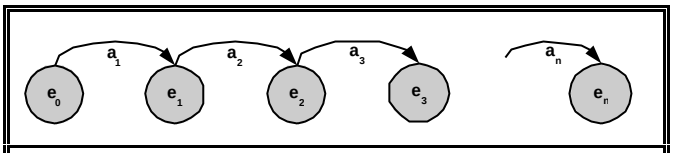

Illustration 16 : Suite de transitions

L'automate, les états et les transitions sont donc traduits en langage EXPRESS.

```
ENTITY Automate;
  its_states : LIST OF Etat;
  its_transition : LIST OF Transition;
END_ENTITY;

ENTITY Etat;
  its_name : INTEGER;
INVERSE
  ses_transitions_entrantes : SET OF Transition FOR son_etat_final;
  ses_transitions_sortantes : SET OF Transition FOR son_etat_initail;
END_ENTITY;

ENTITY Transition;
  its_name : STRING;
  its_automate : Automate;
  its_first_state : Etat;
  its_last_state : Etat;
END_ENTITY;
```

Illustration 17 : Représentation d'un automate en EXPRESS

Il suffit maintenant de poser le problème de la validation d'un **ELEMENT** en termes d'un automate. Nous sommes dans un état d'attente et seule l'arrivée de la balise vraiment attendue (l'étiquette) permet de passer dans l'état suivant. Pour représenter cette condition, il suffit de modéliser deux états, le premier étant celui dans lequel on est et le deuxième celui que l'on souhaite obtenir. La transition porte comme étiquette le nom de l'**ELEMENT** défini dans le modèle de contenu. Si la balise rencontrée possède le même nom alors l'état suivant est activé, sinon l'état renvoyé est un état indéterminé.

La **SEQUENCE** et le **CHOICE** peuvent être représentés selon le même principe. La séquence est simplement représenté par une suite de transition et le choix par *n* transitions différentes pour atteindre le même état.

Il est maintenant possible de donner une représentation (Illustration 18) du modèle de contenu de l'exemple traité précédemment (Illustration 13).

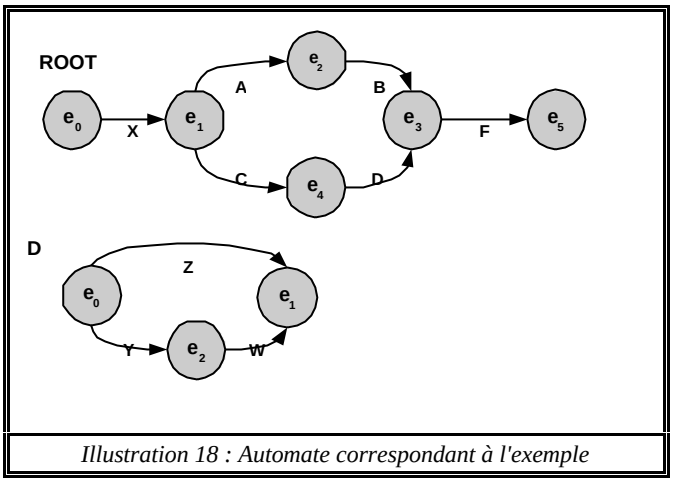

Illustration 18 : Automate correspondant à l'exemple

En arrivant sur une balise contenant des sous-balises, la liste des noms des sous-balises est comparée aux différents chemins possibles. Dès lors qu'un nom de sous-balise n'est pas présent dans les transitions attendues, un état indéterminé est renvoyé et la balise n'est pas validée.

Il reste néanmoins la question du déterminisme de l'automate. Il faut en effet s'assurer qu'un état ne disposera pas de deux transitions sortantes portant la même étiquette. La spécification du langage XML Schema interdit ce genre d'ambiguïté. Une simple contrainte EXPRESS pourrait donc être définie, stipulant que les noms des transitions sortantes d'un état doivent être uniques.

3.3.4 Modélisation des occurrences

Lors de la déclaration d'un modèle de contenu, les **ELEMENT**s peuvent être qualifiés par des indicateurs d'occurences **MINOCCURS** et **MAXOCCURS**. On spécifie ainsi le nombre de fois que cet élément peut être instancié. Il est possible de redéfinir l'**ELEMENT** A avec **MINOCCURS=**3 et **MAXOCCURS=**5. Selon le chemin choisi, le nombre de A imposé sera soit de 3, soit 4, soit 5 (Illustration 19).

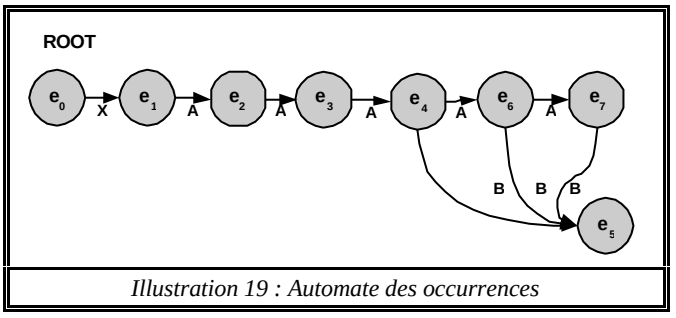

Illustration 19 : Automate des occurrences

3.3.5 Modélisation de l'héritage

L'étude de l'héritage en XML Schema porte sur deux parties : sa représentation dans les modèles et son utilisation dans un document XML.

Héritage au sein du schéma

Dans le cas de la dérivation (l'héritage) par extension, un type complexe est créé à partir d'un autre type de base, et des **ELEMENT**s sont ajoutés dans son modèle. L'automate résultant sera défini comme la séquence de l'automate du type de base et de l'automate correspondant à l'extension. (Illustration 20).

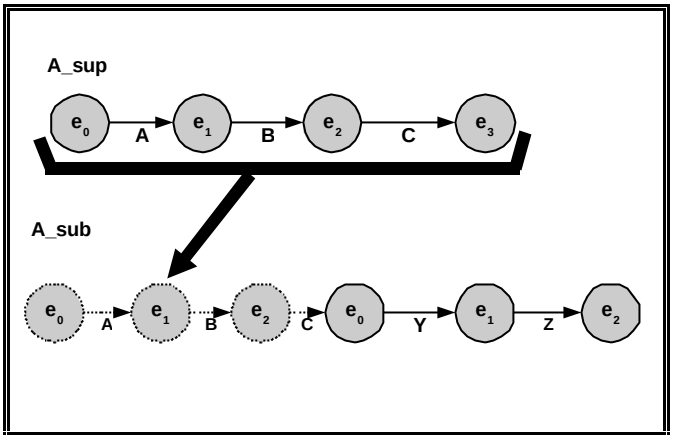

Illustration 20 : Automate correspondant à la dérivation

Ainsi la fonction de validation correspondante cherchera l'automate du type père lors de la validation du fils et comparera le contenu à la séquence des deux automates.

La dérivation par restriction n'est actuellement pas supportée par l'approche proposée mais ne semble pas poser de difficultés particulières.

Héritage au sein du document

L'utilisation d'un type dérivé se fait de manière explicite dans un document XML. La balise possède un attribut **TYPE** qui vient spécifier le type utilisé. Si on omet cet attribut, le type utilisé est celui déclaré dans le modèle. Par contre, il est possible d'utiliser une spécialisation de celui déclaré (relation polymorphe).

En reprenant l'exemple de définition d'une personne en XML (Illustration 2), l'adresse peut être de type française ou américaine. Dans ce cas, la balise **SON_ADRESSE** est éditée pour définir le type d'adresse que l'on souhaite utiliser (Illustration 21).

```
<PERSONNE>
   <SON_NOM>Dupont</SON_NOM>
   <SON_PRENOM>Francis</SON_PRENOM>
   <SON_AGE>23</SON_AGE>
   <SON_ADRESSE TYPE='t_adresseFrance'>
      <SON_CP>86000</SON_CP>
      <SA_VILLE>Poitiers</SA_VILLE>
   </SON_ADRESSE>
</PERSONNE>
<PERSONNE>
   <SON_NOM>Dupont</SON_NOM>
   <SON_PRENOM>Francis</SON_PRENOM>
   <SON_AGE>23</SON_AGE>
   <SON_ADRESSE TYPE='t_adresseUSA'>
      <SA_VILLE>Sunnyvale</SA_VILLE>
      <SON_ETAT>CA</SON_ETAT>
      <SON_ZIPCODE>94089</SON_ZIPCODE>
   </SON_ADRESSE>
</PERSONNE>
```

Illustration 21 : Héritage dans le document XML

Pour contrôler ce contenu, la méthode utilisée est la suivante : la balise **SON_ADRESSE** est reliée à son modèle (un **ELEMENT**) par un attribut dérivé `its_model`. Il est alors possible de trouver le type correspondant et donc son automate associé. En rajoutant un attribut dérivé (`its_derived_type`) à l'entité **COMPLEXTYPE**, on référence l'ensemble des types fils qu'il est possible d'utiliser. Il ne reste alors plus qu'à consulter l'attribut type de la balise et de vérifier s'il est licite. La validation s'opère alors avec l'automate du type dérivé. Sinon, le contenu de la balise est déclaré comme non valide.

3.3.6 Validation d'un document

Enfin, il reste à mettre en œuvre une méthode de validation. En effet, le schéma et le document étant modélisés, il est nécessaire de créer une fonction de validation qui va consister en un parcours des automates générés.

```
ENTITY Element;
  ...
  its_type : Anytype;
  ...
END_ENTITY;
ENTITY ComplexType
SUBTYPE OF (Anytype);
  ...
DERIVE
  its_representation:Automate :=computeAutomate (SELF);
END_ENTITY;
```

Illustration 22 : Modélisation au niveau schéma

```
ENTITY Balise;
  ...
  its_content : LIST [0:?] OF BS;
DERIVE
  its_model:ELEMENT:=giveModel (SELF);
...
  is_conform:BOOLEAN:=checkConformity(SELF);
END_ENTITY;
```

Illustration 23 : Modélisation au niveau document XML

La fonction `computeAutomate` de l'entité **`COMPLEXTYPE`** donne une représentation sous forme d'automate au modèle de contenu. Chaque **`BALISE`** est reliée à son modèle **`ELEMENT`** via l'attribut `its_model`, et donc à la définition de son modèle de contenu (**`COMPLEXTYPE`**), via l'attribut `its_type`. Il suffit maintenant de parcourir le contenu de la balise et de valider les différents mots (étiquettes) de l'automate jusqu'à atteindre, en cas de succès, un état final. La fonction `checkConformity` permet la mise en œuvre de ce parcours.

Notons enfin que la validation d'un document XML ne nécessite pas le parcours de la structure arborescente du document. La validation s'opère localement et automatiquement au niveau de chacune des balises par le biais des attributs dérivés. Finalement, une procédure de plus haut niveau est en charge de collecter les vérifications locales et de produire un compte rendu.

4 Conclusion

Avec la tendance actuelle à utiliser des descriptions de type documentaire (SGML, HTML et XML) pour les échanges entre bases de données, le besoin se fait sentir de non plus seulement structurer l'information mais également de modéliser et de contraindre les éléments d'information échangés. C'est l'objectif des formalismes tels que XML Schema. Ces formalismes se rapprochent des langages de spécification de données, tels que EXPRESS, mais ils ne possèdent pas actuellement d'implémentations réelles permettant de valider la correction des informations échangées.

Ainsi, l'objectif de ce papier était d'étudier la possible intégration de l'approche documentaire, représentée par XML et XML Schema, et l'approche spécification de données, représentée par EXPRESS. L'intégration proposée, réalisée au niveau méta, a permis l'implémentation d'un environnement de validation de documents XML dont la structure était définie par des XML Schema. Notre approche permet l'utilisation des outils existant pour la manipulation de spécification de données EXPRESS. L'avantage de cette approche, qui utilise la méta représentation des constructions de XML Schema, est de pouvoir représenter (et donc manipuler)

totalement tous les concepts véhiculés par XML Schema. De plus, après avoir défini un modèle EXPRESS pour représenter des documents XML, nous avons élaboré une architecture de validation basée sur les automates à états finis. Celle-ci nous permet à moindre coût de valider complètement un document XML par rapport à son schéma. Notons qu'elle utilise un mécanisme puissant du langage EXPRESS, les attributs dérivés, qui allègent considérablement la charge d'implémentation.

A terme, l'approche proposée pourrait être utilisée pour enrichir XML Schema à l'aide de constructions provenant du langage EXPRESS, dont le pouvoir d'expression est bien plus important.

5 Références

[1] P. Chen
The Entity Relationship Model – Toward a Unified View of Data; ACM Transactions on Database Systems, Volume 1, N°1 (Mars 1976)

[2] G.M. Nijssen
An architecture for knowledge base systems; Proc. SPOT-2 conf., Stockholm, 1981

[3] J. Rumbaugh, M. Blaha, W Premerlani, F. Eddy, W. Lorensen
Object Oriented Modelling and Design; Prentice-Hall International Editions, ISBN 0-13-630054-5, 1991

[4] I. Jacobson, G. Booch, J. Rumbaugh
The Unified Software Development Process; Addison-Wesley Eds, 1999

[5] D. Schenk, P. Wilson
Information Modelling The EXPRESS Way, Oxford University Press, 1994

[6] ISO 10303-11
Industrial automation systems and integration -Product data representation and exchange - Part 11: Description Methods: The EXPRESS Language reference manual; 1994

[7] J.H. Coombs, A.H. Renear, S.J. DeRose
Markup Systems and the future of Scholarly Text Processing; Communications of the Association for Computing Machinery, 1987

[8] D. Ragget, A. Le Hors, I. Jacobs
HTML 4.01 Specification; W3C Recommendation, 24 Décembre 1999
http://www.w3.org/TR/html4/

[9] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler
Extensible Markup Language (XML) 1.0 (Second Edition); W3C Recommendation 6 octobre 2000
http://www.w3c.org/TR/2000/REC-xml-20001006

[10] D.C. Fallside
XML Schema Part 0: Primer; W3C Recommendation, 2 Mai 2001
http://www.w3.org/TR/xmlschema-0/

[11] S. Thompson, D. Beech, M. Maloney, N. Mendelsohn
XML Schema Part 1: Structures; W3C Recommendation, 2 Mai 2001
http://www.w3.org/TR/xmlschema-1/

[12] P. V. Biron, A. Malhotra
XML Schema Part 2: Datatypes; W3C Recommendation, 2 Mai 2001
http://www.w3.org/TR/xmlschema-2/

[13] A. Layman, E. Jung, E. Maler, H.S. Thompson, J. Paoli, J. Tigue, N.H. Mikula, S. De Rose *XML-Data*; W3C Note, 5 Janvier 1998
http://www.w3c.org/TR/1998/NOTE-XML-data/

[14] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler
Extensible Markup Language (XML) 1.0 (Second Edition); W3C Recommendation 6 octobre 2000
http://www.w3c.org/TR/2000/REC-xml-20001006

[15] A. Davidson, M. Fuchs, M. Hedin, M. Jain, J. Koistinen, C. Lloyd, M. Maloney, K. Schwarzhof
Schema for Object-oriented XML 2.0, W3C Note, 30 Juillet 1999
http://www.w3.org/TR/NOTE-SOX/

[16] R. Jeliffe
The Schematron: An XML Structure Validation Language using Patterns in Trees; Mai 2000
http://www.ascc.net/xml/resource/schematron/

[17] J. Bicarregui, B. Matthews
Integrating EXPRESS and SGML for Document Modelling in Control Systems Design; 6 Septembre 1995

[18] R. Bourret
Mapping W3C Schemas to Object Schemas, to Relational Schemas; Mars 2001
http://www.rpbourret.com/xml/SchemaMap.htm

[19] R. Bourret
Mapping DTDs to Databases; published on XML.com, 9 Mai 2001
http://www.xml.com/pub/a/2001/05/09/dtdtodbs.html

[20] G. Staub, M. Maier
ECCO Tool Kit -An Environment for the Evaluation of EXPRESS Models and the Development of STEP based IT Applications; User Reference Manual, 1997
http://www-rpk.mach.uni-karlsruhe.de/kometenzen/Ecco/Docs.userman-1_7.ps.gz