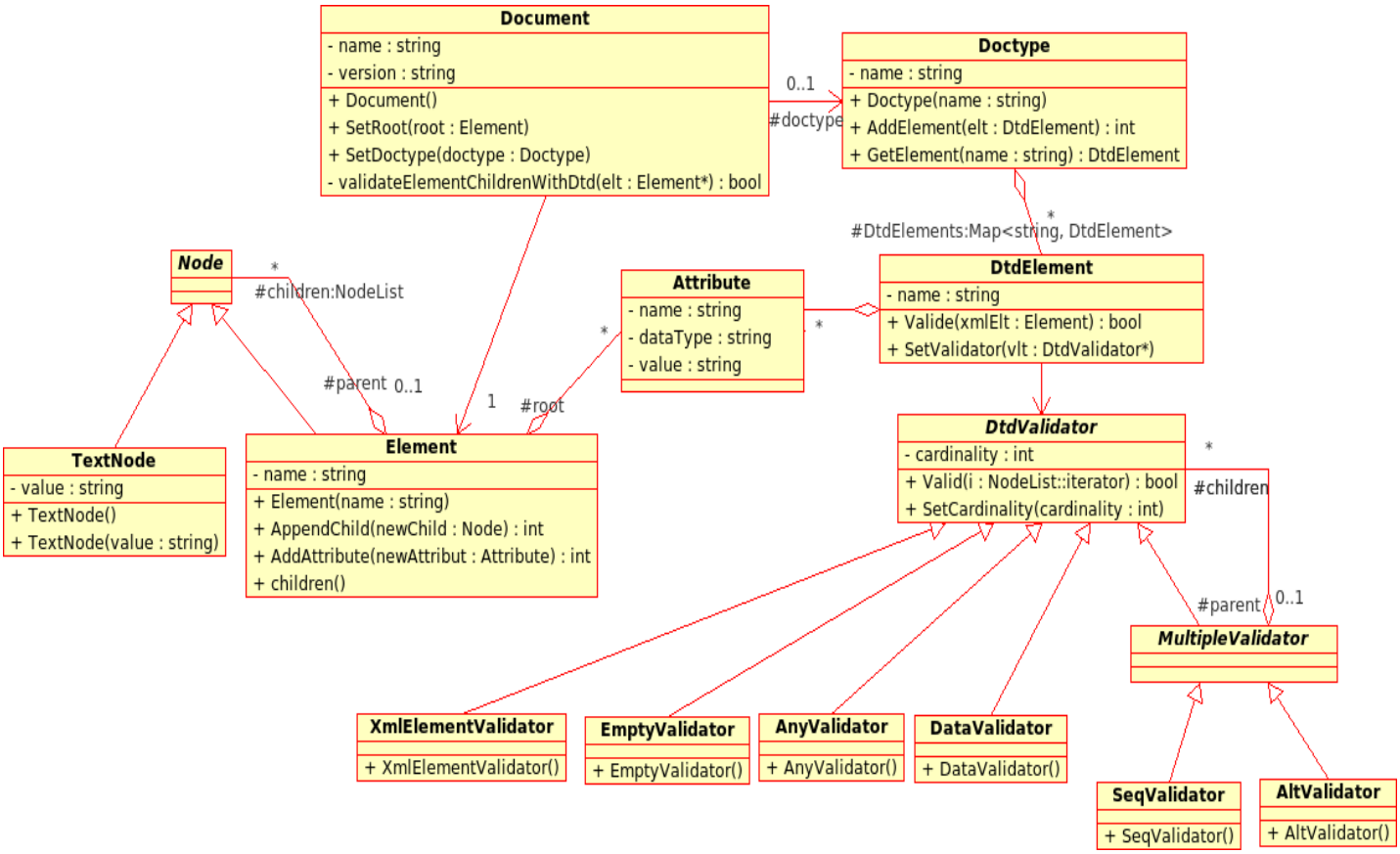
Projet LG

H4314

Paul ADENOT, Yoann BUCH,
Pierrick GRANDJER, Etienne GUERIN,
Martin RICHARD, Arturo MAYOR, Yi Quan ZHOU

Diagramme de classes



Algorithmes

Document

Valider_XML_Avec_DTD :

// Parcours de l'arbre xml en profondeur
// Pour chaque élément xml, on cherche dans Dtd la définition dtd élément correspondant
// et on essaie de valider cet élément xml avec le dtd élément

```
ValiderXmlAvecDtd(Element elt) renvoie booléen
{
    Chercher DtdElement dtdElt dans doctype par elt.nom;
    Si elt n'exite pas alors
       renvoie faux
    Sinon
       // Valider cet élément xml avec l'élément dtd correspondant
```

```
      Booléen résultat = dtdElt.valide(elt);
      Pour chaque element eltFils dans elt.fils faire
            Si ValiderXmlAvecDtd(eltFils) == faux alors
                  résultat = faux;
            Fin si;
      Fin pour;
    Fin si;
    Renvoie résultat;
}
```

DTD_Element

// Algorithme pour valider un élément xml avec le dtd élement correspondant
// L'élément dtd utilise le validateur qu'il possède pour la validation

```
DtdElement.valide(Element elt) renvoie booléan
{
    // Valider l'élément avec le validateur
    booléan résultat = validateur.valide(elt.fils);
    // Valider les attributs
    Si résultat == vrai alors
      Pour chaque attribut att dans elt faire
            Chercher elt.nom dans DtdElement.attlist;
            Si il n'existe pas alors
                  résultat = faux;
            Fin si;
      Fin pour;
    Fin si;
    Renvoie résultat;
}
```

DTD_Validators

XML_Element_Validator :

// Algorithme pour valider un simple xml élément avec un XmlElementValidator

```
Faire
    Si le noeud est un élément et si son nom est celui attendu par la
DTD
      Alors Avancer le curseur d'un cran
```

```
      valide = true
      Si la cardinalité attendue est 0_N ou 1_N
            // Alors on peut attendre plus d'occurences
            Alors continuer = true
      FinSi
    Sinon
      // On n'a pas trouvé l'élément attendu
      continuer = false
      Si cardinalité attendue est 0_N ou 0_1
            // Alors la DTD spécifie qu'on peut ne pas avoir
           l'élément
            // attendu (cardinalité 0), donc on valide tout de même
            Alors valide = true
      FinSi
    FinSi
Tant que continuer = true

Renvoyer Valide
```

Empty_Validator :

// Algorithme de la méthode valid de EmptyValidator, renvoie un booléen
// Cette méthode renvoie true seulement s'il n'y a pas d'éléments suivants
// c'est à dire si le curseur est à la fin de la séquence

```
Si le curseur est à la fin de la séquence
    Alors valide = true
Sinon valide = false
FinSi

Renvoyer valide
```

Any_Validator :

// Algorithme pour valider un élément xml type ANY avec AnyValidator, renvoie un booléen
// Lorsque ANY est attendu par la DTD, n'importe quel élément est valide

```
Mettre le curseur à la fin de la séquence
valide = true
```

```
Renvoyer valide
```

Data_Validator :

// Algorithme pour valider un xml élément type CDATA avec DataValidator, renvoie un booléen

```
Faire
    Si le noeud courant est de type Data
       Alors valide = true
       Avancer le curseur d'un cran
    Sinon // l'élément n'est pas celui attendu
       continuer = false
       Si la cardinalité attendue est 0_1 ou 0_N
            // Alors la DTD spécifie qu'on peut ne pas avoir l'élément
            // attendu (cardinalité 0), donc on valide tout de même
            Alors valide = true
       FinSi
    FinSi
Tant que continuer = true

Renvoyer Valide
```

Multiple_Validator

Seq_Validator :

// Algorithme pour valider un élément type sequence avec SeqValidator, renvoie un booléen

```
Faire
    Tant qu'il reste des éléments dans la séquence
  // Si l'élément est valide, l'avancement du curseur est effectué par
les méthodes concernées, il n'y a donc rien d'autre à faire ici que
passer à l'élément suivant dans la séquence.

       Si l'élément n'est pas valide
            Replacer le curseur au début de la séquence
            Si la cardinalité est 0_1 ou 0_N
```

```
                  // Alors on peut attendre plus d'occurences de
l'élément
                  Alors valide = true
            Sinon
                  valide = false
            FinSi
            Arrêter de lire la séquence
      Sinon
            Passer à l'élément suivant dans la séquence
      FinSi
    FinTantQue

    Si la cardinalité attendue est 0_N ou 1_N
      Alors continuer = true
      FinSi
Tant que continuer = true

Renvoyer valide
```

Alt_Validator :

// Algorithme pour valider un élément xml type alt avec AltValidator, renvoie un booléen

```
AltValidator.valide(ListeNoeud elt.fils) renvoie booléan
    booléan doitContinuer = faux;
    résultat = faux;
    Curseur cv : pointe sur début de la liste validateurs;
    Curseur ce : pointe sur elt.fils;
    Faire
      Tant qu'il reste des validateurs faire
            Essayer de valider elt.fils avec validateur courant;
            Si cv->valide(elt.fils) alors
                  résultat = vrai;
                  avancer le curseur ce;
                  Si cardinalité est 0..n ou 1..n alors
                        doitContinuer = vrai;
                        Sortir du boucle tant que;
                  Fin si;
            Sinon
                  Avancer le curseur cv;
      Fin faire;

      Si cv == fin liste validateurs alors
            doitContinuer = faux;
```

```
            Si cardinalité est 0..1 ou 0..n alors
                  résultat = vrai;
            Fin si;
      Fin si;

      Si doitContinuer == vrai alors
            Remettre cv au début de la liste validateurs;
      Fin si;

   Tant que
      doitContinuer == vrai;
```

Transformation XSL

Lors de la transformation, on part du principe que les différents fichiers sont valides au niveau syntaxe, respect DTD.

Algo1: Parsage XSLT

```
pour chaque <xsl:template match="X">
      créer xsltElem(X,Y)
fin pour
```

commentaires Algo1:

le 'X' correspond à quel type de noeud on se "réfère".
le 'Y' correspond aux textes/balises qui sont compris entre <xsl:template match="X"> de l'itération en cours de la boucle pour et la balise de fin </xsl:template> correspondante.

La méthode '`xsltElem(X,Y)`' crée en mémoire le style XSLT 'Y' que l'on veut pour le noeud 'X' considéré.

'Y' est une chaîne de caractères qui contient des "balises au sens xml" ou du texte "pur", ex:

```
<h2>
<xsl:text>
 Auteur:
</xsl:text>
```

Nous allons stocker les différents éléments de Y dans un tableau, avec les éléments insérés en bout de tableau (cf Algo 2).

Pour les créations `xsltElem(X,Y)`, il faudrait mettre les éléments dans un dictionnaire avec pour index X, pour pouvoir facilement accès au XSLT Y en fonction de X.

Algo 2: transformation squelette XML généré en HTML avec XSLT

pour noeud 'racine' , il faut tout d'abord chercher et appliquer dicoXSLT avec X= "/"
// on fait cette étape pour écrire l'entête du fichier xml: balises <html> <head> <title> etc.

```
pour chaque noeud,  chercher dans dicoXSLT en fonction type de noeud.
La recherche retourne un tableau xsltElem
      pour i = 0 à xsltElem.size()
            if (xsltElem.get(i).value() == "<xsl:apply-template>")
                  output(noeud.text())
                  // on écrit le "PCDATA" du noeud en cours
            elseif (xsltElem.get(i).value() == "<xsl=text>")
                  //ca correspond à du texte qu'on rajoute
                  output(xsltElem.get(++i).value());
                  //on écrit le texte, à rajouter, qui se trouve dans
                  la case suivante du tableau
                  output(noeud.text())
                  // on écrit le "PCDATA" du noeud en cours
                  ++i;
                  // on saute la case qui contient "</xsl=text>"
     fin pour
fin pour
```

commentaires Algo2:

'output' correspond à l'endroit où on écrit le code html (fichier, sortie standard, socket, etc.)
noeud correspond au noeud en cours quand on parcours le squelette XML de l'arbre qu'on a généré
Attention: il faut pouvoir distinguer les noeuds "rapport/titre" et "chapitre/titre" car dans un cas ça donne du `<h1>` et dans l'autre du `<h2>`.