8 SGML

SGML = Standard Generalized Markup Language

Standard international pour l'encodage des informations supplémentaires sur des textes.

SGML est un méta-langage: SGML ne spécifie pas concrètement quelles informations (quels tags) on peut ajouter mais SGML spécifie quelles formes de tags on peut utiliser et comment on peut définir l'ensemble de tags utilisés pour un certain type de document.

On distingue entre le texte et son markup (les informations supplémentaires).

Trois propriétés caractéristiques de SGML:

1. Son caractère déscriptif (non-procédural)
2. Son concepte de type de document
3. Son indépendance du système sous lequel le texte en question est annoté

Markup déscriptif:

Symboles (tags) pour marquer certaines parties d'un texte (p.e. titre, sous-titre, chapitre, ...) comme <titre> ou \title{}. Mais ces symboles ne disent rien sur la présentation de cette partie du texte.

Document type:

SGML permet de définir des types de documents. Dans la définition d'un type de document on spécifie le markup possible (c'est-a-dire la structure) pour ce certain type de document. Chaque document a un type.

Indépendance des systèmes:

Un des buts de SGML est d'assurer que des documents annotés en SGML devraient être lisibles pour des systèmes différents. Pour que des ensembles de caractères différents ne posent pas de problèmes, SGML offre un mécanisme pour définir des suites de caractères qui remplacent des caractères spéciaux (et spécifiques d'un certain système).

8.1 Structures en SGML

Des parties d'un texte qui sont marquées comme formant une unité s'appellent éléments. Chaque type d'élément a un nom element, et un tel élément est entouré par des étiquettes <element> et </element>.

Exemple 11

```
<sp>
   <speaker>HARPAGON</speaker>
   <stage>bas.<stage>
   <p>Tout va comme il faut.</p>
   <stage>(Haut.)<stage>
   <p>H&eacute; bien ! qu'est-ce, Frosine ?</p>
</sp>
```

é est une suite de caractères (une sorte de variable) pour un 'e' avec accent aigu. Des telles variables s'appellent entités (*entities*).

SGML ne spécifie pas les éléments et entités concrets, mais SGML est un meta-langage dans lequel on peut définir le markup possible (les éléments) d'un texte.

SGML permet de définir des types d'éléments à partir de leur contenu, c'est-à-dire leurs sous-élélements eventuels et le type de donnée qu'ils peuvent contenir.

Exemple 12 *Un élement* `<sp>` *(spoken paragraph) peut être défini comme*

- *contenant un sous-élément optionnel* `<speaker>`
- *suivi d'à moins un sous-élément* `<stage>` *ou* `<p>`.

8.2 Définition des éléments en SGML

Les définitions des éléments et des entités possible dans un certain type de document sont mises dans un DTD *document type definition*.

Exemple 13
```
<!ELEMENT sp          - -   (speaker?, (p | stage)+)              >
<!ELEMENT speaker     - O   (#PCDATA)                             >
<!ELEMENT p           - O   (#PCDATA)                             >
<!ELEMENT stage       - O   (#PCDATA)                             >
```

Une déclaration dans un DTD est entourée par `<` et `>`. Après le `<` ouvrant il faut un point d'exclamation suivi d'un mot clé, `ELEMENT` pour la déclaration d'un élément. Ensuite le nom de l'élement est donné suivi de deux symboles (des règles de réduction), suivi d'un modèle du contenu de l'élément. A la fin on met la parenthèse `>` fermante.

Règles de réduction

Les deux symboles spécifient si le tag ouvrant et le tag fermant pour l'élément sont obligatoires ou optionnels, le premier symbole concerne le tag ouvrant et le deuxième le tag fermant.

Un trait `-` veut dire que le tag en question est obligatoire tandis qu'un `O` veut dire qu'il est optionnel.

Modèle du contenu

La troisième partie (après le nom et les règles de réduction) est le modèle du contenu qui spécifie les élements permis à l'intérieur de l'élément en question. Il y a des mots réservés qu'on peut utiliser ici comme `#PCDATA` pour *parsed character data*, ce qui signifie une séquence de caractères.

Le `?` après une partie du modèle du contenu veut dire que cette partie est optionnel, le `+` signifie 1 ou plusieurs occurrences. Le `*` signifie 0 ou un nombre arbitraire d'occurrences.

Pour connecter deux parties d'un modèle du contenu, on a trois connecteurs:

- `|` qui signifie qu'une des deux parties doit apparaître,
- `,` qui signifie que les deux doivent apparaître dans l'ordre spécifié dans la déclaration et
- `&` qui signifie que les deux doivent apparaître dans un ordre arbitraire.

Dans Ex. 13, un élément `sp` consiste alors en un élément `speaker` optionnel suivi de 1 ou plusieurs occurrences d'un élément `p` ou `stage`.

Exemple 14 *Supposons qu'on veut assurer qu'un* `sp` *contient à moins un élément* `p` *en contenant un nombre arbitraire de* `p` *ou* `stage`*. Cela peut être fait comme suit:*

```
<!ELEMENT sp       - -   (speaker?, stage*, p, (p | stage)*)    >
```

Parfois on a des éléments qui peuvent apparaître à n'importe quel niveau á l'intérieur d'un certain élément. Par exemple, on peut imaginer d'avoir `stage` aussi á l'intérieur des paragraphes `p` qui occurrent dans un élément `sp`.

Une déclaration de `sp` correspondante:

```
<!ELEMENT sp             - -   (speaker?, p+)          +(stage)  >
<!ELEMENT stage          - -   (#PCDATA)                         >
```

Il s'agit d'une exception à la forme standard d'un modèle de contenu. Le `+` précédant `(stage)` indique qu'il s'agit d'une inclusion (*inclusion exception*). Cela veut dire que `stage` peut être un sous-élément direct de `sp` ou un sous-élément d'un de ces sous-éléments etc.

Le tag fermant de `<stage>` doit être obligatoire maintenant pour pouvoir reconnaître la fin d'un élément `stage`.

Exemple 15

```
<sp><speaker>Ines
    <stage>, la prenant dans ses bras</stage>
    <p>Mon pauvre petit! <stage>(A Garcin :)</stage> L'enqu&ecirc;te est
       finie. ...
</sp>
```

Pour interdire l'apparition d'un élément à n'importe quel niveau à l'intérieur d'un autre élément on fait une *exclusion exception* en mettant `-` au lieu de `+`.

Exemple 16 *Pour interdire* `stage` *à l'intérieur de* `speaker`*, on peut déclarer*

```
<!ELEMENT speaker        - -   (#PCDATA)             -(stage)  >
```

8.3 Définition des attributs en SGML

Un élément peut avoir des traits (*attributes*).

Exemple 17

```
<sp>
   <speaker who='Harpagon'>HARPAGON</speaker>
   <stage>bas.<stage>
   <p>Tout va comme il faut.</p>
   ...
</sp>
```

Les traits possibles pour un certain élément sont déclarés dans la DTD:

```
<!ATTLIST speaker
          who          CDATA          #IMPLIED  >
```

Une déclaration d'attributs est marquée par le mot clé `ATTLIST`. Elle contient le nom de l'élément suivi d'une déclaration de chaque attribut. Celle-ci contient le nom de l'attribut suivi du type de sa valeur suivi d'une partie qui spécifie ce qu'il faut faire si l'attribut n'est pas présent.

Le type de la valeur d'un attribut peut être

- un ensemble de valeurs possibles séparées par des traits verticaux, p.e. `(yes | no)`
- `CDATA`: une séquence de caractères (entre guillemets) qui peut contenir des tags mais dans ce cas, ces tags ne sont pas reconnus comme tags par le parser
- `ID`: un identificateur unique d'un élément
- `IDREF`: un pointeur vers un autre élément, c'est-à-dire la valeur d'un attribut du type `ID` d'un autre élément

- `NMTOKEN`: un strings de caractères alphanumériques
- `NUMBER`: la valeur se compose seulement des chiffres

La dernière partie de la déclaration d'un attribut peut être

- une valeur default
- `#REQUIRED` qui signifie que l'attribut est obligatoire. S'il n'est pas présent, le parser va sortir un message d'erreur.
- `#IMPLIED` qui signifie que l'attribut est optionnel (et peut être omis)
- `#CURRENT` qui signifie que, si la valeur n'est pas spécifiée, la dernère valeur de cet attribut va être reprise.

Exemple 18
```
<!ATTLIST poem
          id        ID                                #IMPLIED
          status    (draft | revised | published)     #CURRENT   >
```
Un `poem` *sans valeur pour* `status` *serait étiqueté avec le même* `status` *que le* `poem` *précédant.*

Parfois dans un texte on fait référence à une autre partie du texte (p.e. 'voir chapître 2'). Pour encoder cela en SGML on utilise l'identificateur `id` de l'élément auquel on veut référer. La référence est un élément à part qui n'a pas de contenu. Il est seulement un pointeur.

Exemple 19
```
<!ELEMENT poemref   - O  EMPTY                              >
<!ATTLIST poemref        target   IDREF   #REQUIRED         >
```

Exemple 20
```
<poem id=bateau status=published>
Le Bateau ivre
...
</poem>

Le bateau ivre de Rimbaud <poemref target=bateau> ...
```

Si le parser trouve une telle référence il teste si l'élément auquel elle réfère existe.

8.4 Entités en SGML

En SGML on peut définir des abbréviations (*entities*) pour certaines parties d'un texte. Cette partie peut être une suite de caractères ou un texte dans un fichier à part.

Exemple 21
```
<!ENTITY sgml "Standard Generalized Markup Language">
<!ENTITY mydtd SYSTEM "mydoc.dtd">
```

On réfère à une entité avec le nom de l'entité precédé d'un `&` et suivi d'un `;`.

Exemple 22
```
Le DTD &mydtd; est un DTD en &sgml;.
```

Entités permettent de donner des noms à des caractères spéciaux qui ne peuvent pas être représenté par chaque système.

Exemple 23

```
<!ENTITY eacute "é">


... un corpus annot&eacute; en SGML
```

Un texte annoté d'une telle façon peut être lit sous chaque système. Pour le parser, il faut seulement modifier la définition de l'entité dans la DTD.

On peut aussi utiliser des entités à l'intérieur d'un DTD. Ces entités s'appellent *parameter entities* et elles sont définies en utilisant `%` et une espace devant le nom. Quand on utilise une telle entité, son nom est précédé par `%` (sans espace) et suivi de `;`.

Exemple 24

```
<!ENTITY % m.token  'abbr | date | num | measure | name | term | time' >
<!ENTITY % a.global '
           id        ID         #IMPLIED
           lang      IDREF      #IMPLIED'              >

<!ELEMENT phrase     - O    (%m.token; | #PCDATA)*    >
<!ATTLIST phrase     %a.global;                        >
```

8.5 Résumé

Un texte annoté en SGML consiste en

- une déclaration du dialecte SGML utilisé (normalement pas visible pour l'utilisateur)
- un DTD
- le texte avec markup conforme au DTD

Le DTD spécifie le markup d'un certain type de document. Ce type est déclaré par le mot clé `DOCTYPE`:

Exemple 25

```
<!DOCTYPE my.dtd [
  <!-- toutes les definition pour my.dtd -->
]>

<my.dtd>
  Ceci est un document du type my.dtd
</my.dtd>
```

(Tout ce qui est entouré par `<!--` et `-->` sont des commentaires qui sont ignorés par le parser.

En général on met le contenu du DTD dans un fichier à part. Le fichier contenant le texte commence par la déclaration de son type en donnant le nom du fichier ou les définitions du DTD se trouvent.

Exemple 26

```
<!DOCTYPE my.dtd SYSTEM "mydoc.dtd" [
]>

<my.dtd>
  ...
</my.dtd>
```

Même si on met les définitions dans un fichier à part, on peut toujours donner des définitions de plus entre les deux parenthèses [ ]. Dans ce cas le DTD utilisé pour ce document contient d'abord les définitions supplémentaires et ensuite les définitions du fichier à part. L'ordre est important parce que plusieures définitions d'une entité sont permises, mais seule la première définition d'une entité compte. C'est à dire on peut ici redéfinir une entité déjà présent dans le fichier à part. Plusieures définitions d'un élément par contre ne sont pas permises.

Pour ne pas mettre tout le texte annoté dans un même fichier, on peut aussi se servir des entités déclarées spécialement pour ce texte.

Exemple 27

```
<!DOCTYPE my.dtd SYSTEM "mydoc.dtd" [
    <!ENTITY part1 SYSTEM "part1.txt">
    <!ENTITY part2 SYSTEM "part2.txt">
]>

<my.dtd>
  <text>
    &part1;
    &part2;
  </text>
</my.dtd>
```