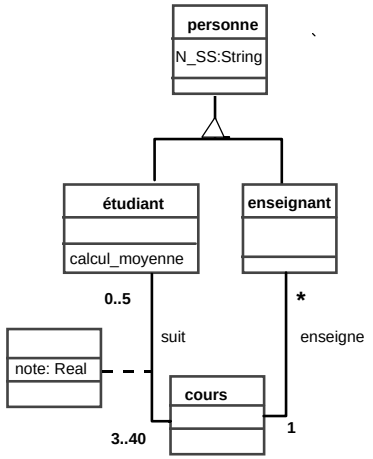
Laboratoire d'Informatique Scientifique et Industrielle
Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
Téléport 2 - BP 109
86960 FUTUROSCOPE Cedex

Spécification de données et présentation documentaire: L'exemple des catalogues electroniques

G. PIERRA
LISI / ENSMA

Introduction

Modélisation conceptuelle



Spécification de données

```
ENTITY personne
   num_ss: STRING(13) FIXED;
UNIQUE
   num_ss;
WHERE
   wr: num_ss[1] = '1'OR num_ss[1] = '2';
END_ENTITY;

ENTITY etudiant
SUBTYPE OF personne;
   notes: ARRAY [1:5] OF OPTIONAL REAL;
   suit: ARRAY [1:5] OF OPTIONAL cours;
DERIVE
   moyenne:REAL:= calcule_moyenne(SELF);
END_ENTITY;

ENTITY enseignant
SUBTYPE OF personne;
   enseigne: SET[1:?] OF cours;
END_ENTITY;

ENTITY cours ...  END_ENTITY;

FUNCTION calcule_moyenne(e: etudiant): REAL;
  LOCAL ...
```

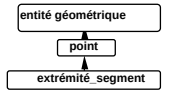

Plan

- **Spécification de données**
- **représentation de grammaires**
- **Présentation documentaire**

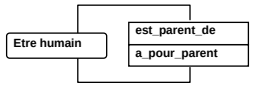
Que veut-on modeliser ? la connaissance

3 dimensions

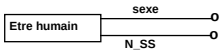
- structurelle:
  - catégories hiérarchisées



  - en relation



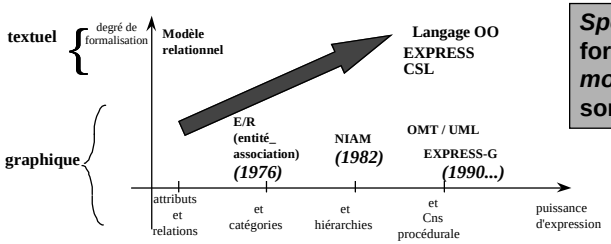
- descriptive



- procédurale:
  - logique (sexe = femme **ET** N_SS [1] = '2') **OU ...**
  - fonctionnel distance (Point_1, Point_2) = √**...**

«formalismes» de modélisation

- Longtemps uniquement graphiques
- Progres :
  - plus de puissance d'expression
  - plus de sémantique: invariants de données = sémantique ensembliste
- Recemment:
  - emergence du textuel
  - complétude des contraintes



Spécification de données*, formalismes où: *modèles* et *contraintes* sont *compilables

• **Question ouverte:**
quelle connaissance procédurale dans une base de données ?

Un formalisme de spécification: le langage EXPRESS

☛ **EXPRESS (ISO 10303-11:1994)** est un langage ensembliste et orienté objet de spécification de données

☛ **EXPRESS permet de décrire textuellement**:
- des types de données
- les contraintes appliquées aux instances de ces types (complétude calculatoire)
  - ☛ **invariant fonctionnels** : attributs dérivés
  - ☛ **invariant logiques :** règles locales et globales

☛ **Compilable, EXPRESS est associé à des «mappings»**:
- génération automatique des modèles d'échange, d'interfaces d'acces, de programme de lecture/écriture

Enfin, **EXPRESS-G** permet la représentation graphique de schémas EXPRESS

Un formalisme de spécification: le langage EXPRESS: les types

Simples | **Utilisateurs** | **Enuméré, Union** | **Collection**

```
TYPE t_num_ss = STRING(13) FIXED;
WHERE
wr: SELF[1] = '1'OR SELF[1] = '2';
END_TYPE;
```

```
TYPE cours = ENUMERATION OF(
        math, info, histoire, sport);
END_TYPE;
```

```
ENTITY personne
ABSTRACT SUPERTYPE OF (
ONEOF (etudiant, enseignant);
   nom: STRING;
   annee_naissance : integer
   num_ss: t_num_ss;
END_ENTITY;
```

```
ENTITY etudiant
SUBTYPE OF personne;
   notes: ARRAY [1:5] OF OPTIONAL REAL;
   suit: ARRAY [1:5] OF OPTIONAL cours;
END_ENTITY;
```

la valeur prédéfinie '?' est ajoutée aux valeurs de chaque type

EXPRESS: Les contraintes

☛ **EXPRESS supporte:**
- invariants **fonctionnels**
  propriétés calculables par le système = fonctions — DERIVE
- invariants **logiques**
  propriétés vérifiables par le système = prédicats — ...

☛ **Portée d'un invariant logique**:
- invariants **locaux**, propre à une entité et vérifiables séparément — WHERE
- invariants **globaux**, calculable sur l'ensemble de la population — RULE

☛ **Différents invariants logiques:**
- de type
- unicité (contrainte de clé, dépendance fonctionnelle) — UNIQUE
- totalité — OPTIONAL, [. , .], INVERSE
- cardinalité — - id -
- contraintes ensemblistes — $\{x \in E \mid p(x)\} \Leftrightarrow$ `QUERY (x <* E | p(x));`
- prédicats prédéfinis: — TYPEOF, EXISTS , USEDIN,...

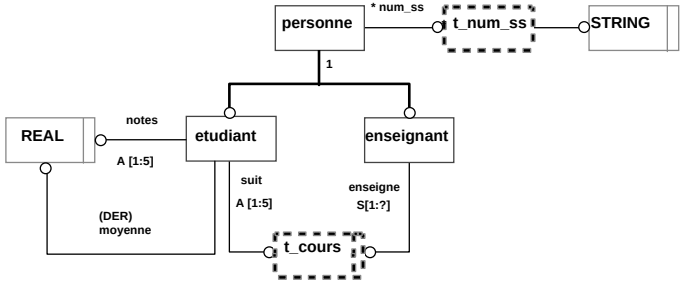
Contraintes = fiabilité

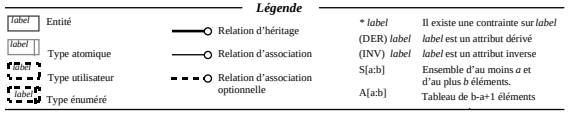
```
SCHEMA etablissement_schema;

TYPE cours = ENUMERATION OF(
        math, info, histoire, sport);
END_TYPE;

ENTITY personne
ABSTRACT SUPERTYPE OF
   ONEOF (etudiant, enseignant);
   num_ss: t_num_ss;
UNIQUE
   ur: num_ss;

END_ENTITY;

ENTITY etudiant
SUBTYPE OF personne;
   notes: ARRAY [1:5] OF OPTIONAL REAL;
   suit: ARRAY [1:5] OF OPTIONAL cours;
DERIVE
   moyenne:REAL:= calcule_moyenne(SELF);
END_ENTITY;
```

```
TYPE t_num_ss = STRING(13) FIXED;
WHERE
wr: SELF[1] = '1'OR SELF[1] = '2';
END_TYPE;

ENTITY enseignant
SUBTYPE OF personne;
   enseigne: SET[1:?] OF cours;
END_ENTITY;

FUNCTION calcule_moyenne(e: etudiant): REAL;
  LOCAL
     SOM: REAL; nombre: INTEGER;
  END_LOCAL;
  REPEAT I = 1 TO 5
     IF EXISTS (e.notes[i]) THEN
        som := som + e.notes[i];
        nombre := nombre + 1;
     END_IF;
  END_REPEAT;
RETURN(som/nombre);
END_FUNCTION;
```

```
RULE autant_etudiant_que_prof FOR (etudiant , enseignant);
WHERE         SIZEOF(etudiant) >= SIZEOF(enseignant);     END_RULE;
                                                          END_SCHEMA ;
```

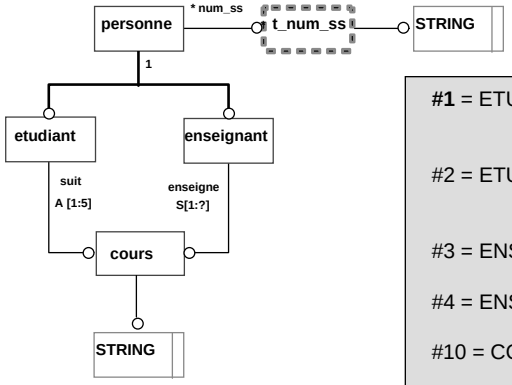

EXPRESS-G: une version graphique





Echanges de fichiers neutres - ISO 10303- 21

EXPRESS-G



Fichier physique

#**1** = ETUDIANT ('**1700975121457**',
(**#10**,#11, $, $, $));

#2 = ETUDIANT ('2700286054018',
(#11 ,$, **$**, $, $**)**);

#3 = ENSEIGNANT ('1541211100004', (#10));

#4 = ENSEIGNANT ('1600366015259', (#11)));

#10 = COURS ('math');

#11 = COURS ('info');

La "technologie" EXPRESS

☛ **EXPRESS** est un langage **formel** i. e., **traitable par machine**

☛ Le système de contraintes associe une **sémantique ensembliste** fine aux données

☛ A partir d'un modèle **EXPRESS** on peut générer **automatiquement**:

- Un interface d'acces à une base de données conforme au modèle («SDAI»)
- Un programme capable de lire/écrire un fichier d'échange et de représenter les données en Java, C++,...
- Un programme capable de lire un fichier d'échange et de charger une base de données;
- ...

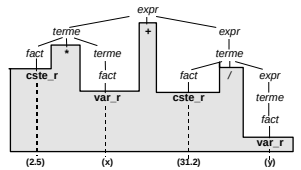
☛ **Un véritable technologie pour la modélisation et le traitement automatique des données fortement structurées**

Plan

- **Spécification de données**
- **représentation de grammaires**
- **Présentation documentaire**

Grammaire hors contexte

- Problème mal résolu dans les formalismes de modélisation:
  représentation de la connaissance procédurale
  - **METHODE USUELLE: GRAMMAIRES**
- Grammaire hors-contexte:
  - **T : symboles terminaux** **{ cste_r, cste_int, var_r, var_int, +, -, *', /, (,) }**
  - **N : symboles non terminaux** ***{ expr, terme, facteur }***
  - **R : règles de production** **R ⊂ N x (N ∪ T)***
    - *expr* → *terme* | *terme* **+** *expr* | *terme* **-** *expr*
    - *terme* → *fact* | *fact* ***** *terme* | *fact* **/** *terme*
    - *fact* → **cste_r | cste_int | var_r | var_int |** (*expr*)
  - **A : symbole initial ou axiome**
- Arbre syntaxique: 2.5 * x + 31.2 / y
- Syntaxe abstraite:
  seuls les opérateurs et opérandes sont significatifs



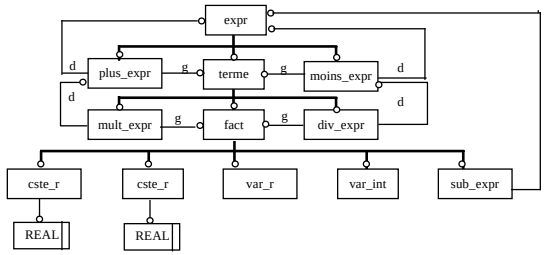
Représentation d'une grammaire hors contexte

- Principe de représentation:
  - **on remplace :** ***"→ : se ré-écrit "***
  - **par :** ***" se spécialise en "***
    - **1 terminal , 1 non-terminal ==> 1 entité**
    - **1 prodution ==> 1 entité sous-type de sa partie gauche**
  - ***Exemple:*** *expr* → *terme* | *terme* **+** *expr* | *terme* **-** *expr*
    `expr` `terme` `plus_expr` `moins_expr`

expr → *terme* | *terme* **+** *expr* | *terme* **-** *expr*

terme → *fact* | *fact* ***** *terme* | *fact* **/** *terme*

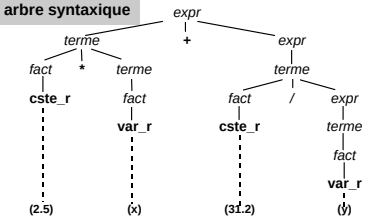
fact → **cste_r | cste_int | var_r | var_int |** (*expr*)



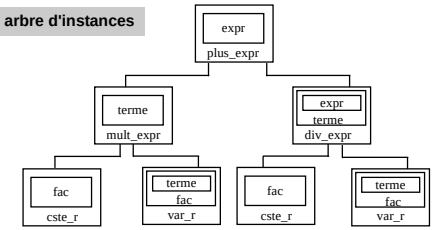
- ***inutile de représenter '+', '-',...***

Exemple: 2.5 * x + 31.2 / y

arbre syntaxique



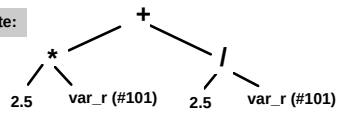
arbre d'instances



fichier physique

```
#1    =    plus_expr ( #10, #11);

#10   =    mult_expr ( #100, #101);
#11   =    div_expr ( #110, #111);

#100 =     cste_r ( 2.5 );
#101 =     var_r ();

#110 =     cste_r ( 31.2 );
#111 =     var_r ();
```

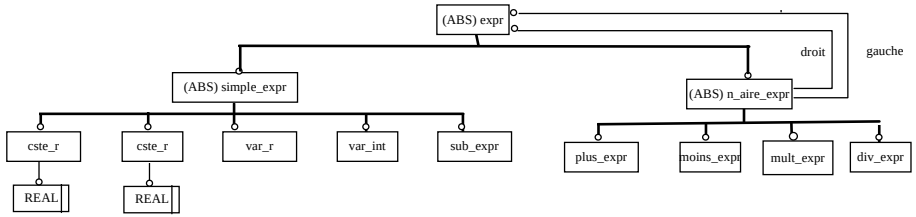
= syntaxe abstaite:



Tous langage décrit par une grammaire hors contexte peut se représenter / s'échanger sous forme de données

Simplification

- Inutile de représenter certains terminaux: représentés dans la classe
- inutile de représenter les priorités



Avantage

- règles de semantique statique ==> contraintes d'intégrité
- Exemple: règles de typage
  - récursif → fonctions locales: attributs dérivés

TYPE type_expr = ENUMERATION OF(int, r, erreur); END_TYPE;

```
ENTITY expr;
  son_type : type_expr
END_ENTITY;

ENTITY simple_expr     SUBTYPE OF (expr) ;
END_ENTITY;

ENTITY cste_r      SUBTYPE OF (simple_expr) ;
  valeur: REAL;
DERIVE
  SELF\expr. son_type : type_expr := r;
END_ENTITY;
```

```
ENTITY n_aire_expr SUBTYPE OF (expr);
   droite, gauche : expr;
DERIVE
  SELF\expr. son_type : type_expr
        := type_n_aire (droite, gauche);
END_ENTITY;

FUNCTION type_n_aire (d,g : expr ) : type_expr;
IF d.son_type = g.son_type
THEN   RETURN(d.son_type);
ELSE   RETURN(erreur);
END_FUNCTION;
```


programmation évenementielle

- Grammaires attribuées:
  - des ***attributs*** sont associés aux éléments (N ∪ T)
  - des ***routines sémantiques*** *sont* déclanchées sur chaque règle

- *Exemple:* Evaluation expression avec Lex & Yacc:
  - Chaque terminal a une valeur

  - ***expr*** → ***terme*** { $$ = $1} | ***terme + expr*** { $$ = $1 + $3} | ***terme - expr*** { $$ = $1 - $3}
  - ***terme*** → ***fact*** { $$ = $1} | ***fact * terme*** { $$ = $1 * $3 }

    | ***fact / terme*** { if $3 == 0.0 yyerror ("division par zero"); else $$ = $1 / $2}
  - ***fact*** → **cste_r** { $$ = $1} | **cste_int** { $$ = $1} | **var_r** { $$ = $1->valeur} | **...**

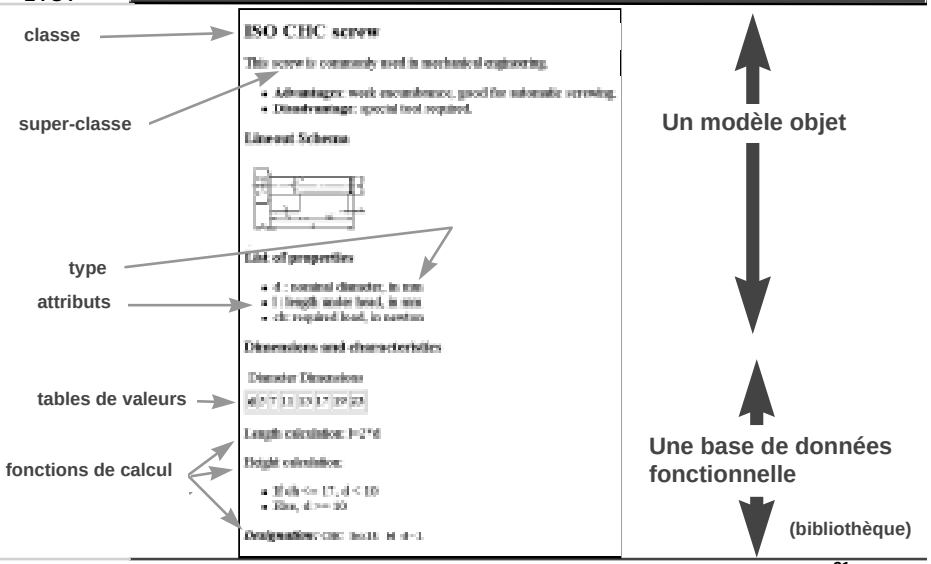
programmation évenementielle dans les spécification de données

TYPE t_valeur = SELECT (INTEGER, REAL) ; END_TYPE;

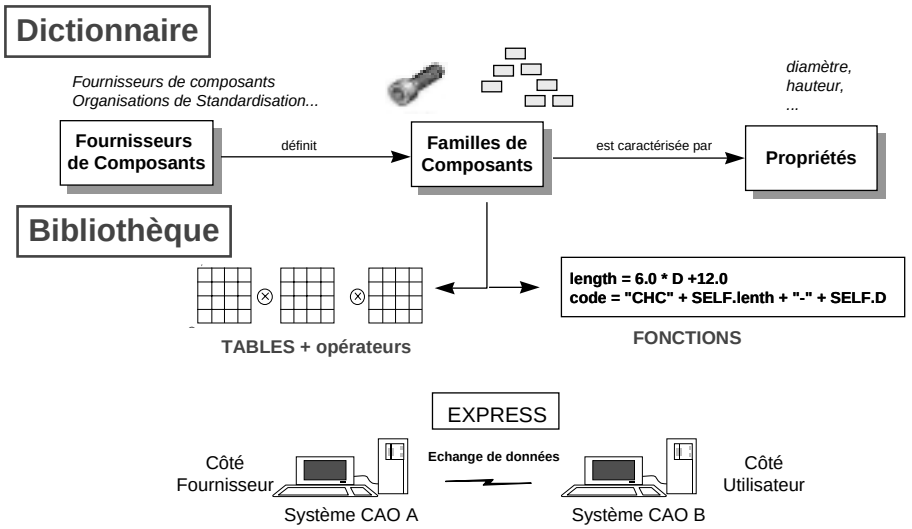
```
ENTITY expr ABSTRACT SUPERTYPE;
evalue : t_valeur;
END_ENTITY;

ENTITY simple_expr  ABSTRACT SUPERTYPE
      SUBTYPE OF (expr) ;
END_ENTITY;

ENTITY cste_r      SUBTYPE OF (simple_expr) ;
  valeur: REAL;
DERIVE
  SELF\expr. evalue : t_valeur := valeur ;
END_ENTITY;

ENTITY var_r    SUBTYPE OF (simple_expr) ;
DERIVE
  SELF\expr. evalue : t_valeur := interprete (SELF) ;
END_ENTITY;
```

```
ENTITY n_aire_expr   ABSTRACT SUPERTYPE
   SUBTYPE OF (expr);
   droite, gauche : expr;
END_ENTITY;

ENTITY plus_expr SUBTYPE OF (expr);
DERIVE
 SELF\expr. evalue : t_valeur
        := droite. evalue + gauche. evalue ;
END_ENTITY;

ENTITY div_expr    SUBTYPE OF (expr);
DERIVE
 SELF\expr. evalue : t_valeur
        := valeur_div (droite, gauche) ;
END_ENTITY;

FUNCTION valeur_div (d, g: expr) : type_expr;
IF d.evalue = 0 OR d.evalue = 0
THEN   RETURN(?);
ELSE   RETURN(droite. evalue / gauche. evalue );
END_FUNCTION;
```

application: les catalogues industriels



Le modèle P-LIB (ISO 13584)



Le modèle P-LIB ? (ISO 13584)

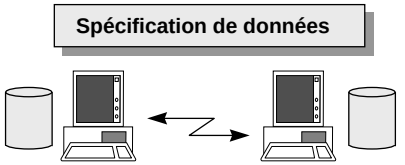
- Les différentes spécifications représentent de façon structurelle:
  - Le modèle objet ("dictionnaire PLIB")
  - les variables et fonctions
  - les types et domaines
  - les tables et l'algèbre relationnel
- La programmation evenementielle est utilisée pour
  - évaluer les valeurs implicites
  - générer du code SQL3
- Des vues en DHTLM (HTML + Javascript) sont générées automatiquement

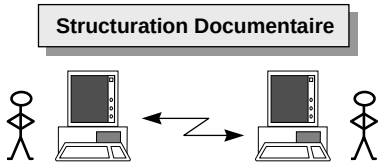
Exemples

Plan

- **Spécification de données**
- **représentation de grammaires**
- **Présentation documentaire**

Des données aux documents





*UML + OCL , **EXPRESS***

*SGML, **XML**, HTML*

Valeurs sûres (contraintes d'intégrité)
Représentation explicite

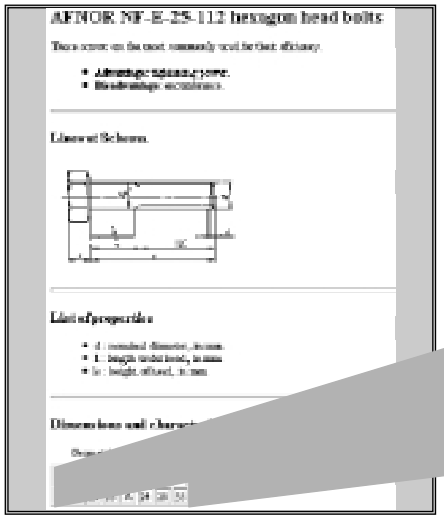
Pas de présentation des données

Valeurs peu sûres
Représentation implicite

Présentation des données

Complémentaire
=> intégration des deux: représentation XML de spécifications EXPRESS

Exemple


AFNOR NF-E-25-112 hexagon head bolts

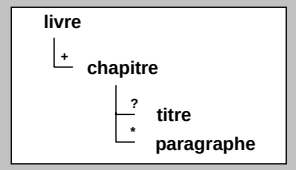
=> Pas de type de données, pas de contrainte
=
PAS DE SEMANTIQUE

Pas exploitable automatiquement

d	5	6	7	8	10	10	30
l	12	15	15	**A**	24	20	55

SGML/XML : Eléments et Attributs

Définition Type de Document (DTD)



<!**ELEMENT** livre **- -** (chapitre**+**) >
<!**ELEMENT** chapitre **- O** (titre**?**, paragraphe*****) >
<!**ELEMENT** titre - - **#PCDATA** >
<!**ELEMENT** paragraphe - - #PCDATA >

<!**ATTLIST** livre ident ID **#IMPLIED** >

Une instance de document

```
<livre ID = "1">
  <chapitre>
    <titre>Introduction à SGML/XML</titre>
  </chapitre>
  <chapitre>
    <p>... définition type de document ...</p>
    <p>Ainsi, les ...</p>
  <chapitre>
    <titre>Intégration des approches</titre>
    <p>... structuration documentaire...</p>
  </chapitre>
</livre>
```

SGML/XML: Entités

- Entité paramètre

Une DTD

```
<!ENTITY % type_de_vehicule
        (voiture | motocyclette | bicyclette)>
...
<!ELEMENT vehicule - -
        (#PCDATA  %type_de_vehicule;)>
```

Une instance de document

```
<vehicule>
        BMW R3
        <motocyclette/>
</vehicule>
```


DTD / BNF

- Déclaration d'un ELEMENT dans une DTD:
  <!ELEMENT livre - - (chapitre+)>
- Equivalent dans une grammaire de type BNF
  livre ::= '<livre>' {chapitre} '</livre>'.
- Déclaration d'une ENTITY dans une DTD:
  <!ENTITY % texte_stylé (normal | note | référence_biblio)>
  <!ELEMENT normal - - #PCDATA>
- Equivalent dans une grammaire de type BNF
  texte_stylé ::= normal | note | référence_biblio.
  normal ::= '<normal>' PCDATA '</normal>'.

Lisibilité (1)

Présenter simultanément les concepts fortement connectés (Cardinalité 1:1)

Présentation éclatée →

```
#1   = Personne(#21,#13,#42);
#5   = Personne(#16,#4,#8);
#9   = Personne(#15, #2, #12);
#15 = Nom('Durand');
#16 = Nom('Martin');
#21 = Nom('Dupont');
#2   = Prenom('Anatole');
#4   = Prenom('Michel');
#13 = Prenom('Jean');
#8   = Laboratoire('LEA');
#12 = Laboratoire('LISI');
#42 = Laboratoire('LISI');
```

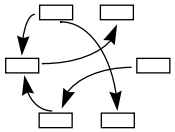
Présentation synthétique →

```
Personne (Nom('Dupont'), Prenom('Jean'), Laboratoire('LISI'))
Personne (Nom('Martin'), Prenom('Michel'), Laboratoire('LEA))
Personne (Nom('Durand'), Prenom('Anatole'), Laboratoire('LISI'))
```

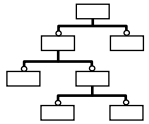

Lisibilité (2)

Définir un ordre logique de présentation des différents concepts

EXPRESS



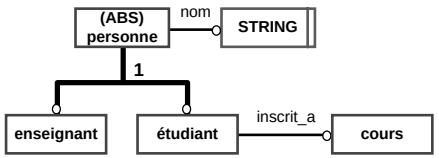
XML



- Identifier les catégories de concepts autonomes
- Choisir un ordre entre les concepts de même niveau (ELEMENT)
  = {section}
- Raffiner
  - si référence unique: *mode interne*
  - si références multiples dans même section: *mode externe* (réservoir dans section)
  - si références multiples dans plusieurs sections: *ré-itérer*

Classe, Attribut, Héritage

EXPRESS



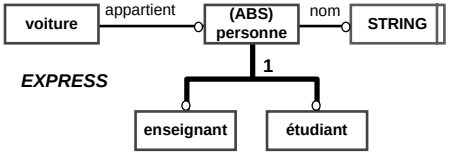
Une instance de document

```
<étudiant id = 2>
   <personne id = 1>
      <nom>Georges</nom>
   </personne>
   <inscrit_a>
      </cours id = 3>
   </inscrit_à>
</étudiant>
```

DTD

```
<!ELEMENT personne - - nom>
<!ATTLIST personne id ID #IMPLIED>
<!ELEMENT nom - - #PCDATA>

<!ELEMENT étudiant - -
                personne, inscrit_à>
<!ATTLIST étudiant id ID #IMPLIED>

<!ELEMENT inscrit_à - - cours>

<!ELEMENT cours - - EMPTY>
<!ATTLIST cours id ID #IMPLIED>
```


Polymorphisme et XML



EXPRESS

- instance(étudiant) = **étudiant**
- instance(enseignant) = **enseignant**

- instance(personne) = instance(étudiant) | instance(enseignant) **= étudiant | enseignant**

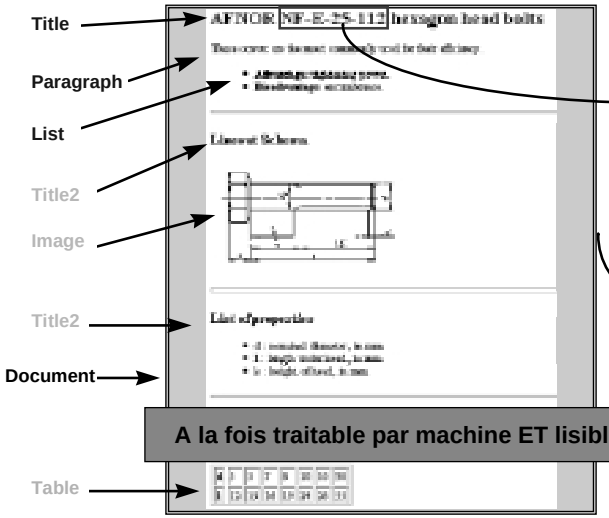
DTD

```
<!ENTITY % instances_of_personne (
      (%instances_of_étudiant;) | (%instances_of_enseignant;))>
<!ENTITY % instances_of_enseignant (enseignant)>
<!ENTITY % instances_of_étudiant (étudiant )>

<!ELEMENT voiture  - -  appartient>
<!ATTLIST voiture id ID #IMPLIED>

<!ELEMENT appartient - - (%instances_of_personne;)>
```

Une instance de document

```
<document>
   <voiture id =1>
      < appartient >
         <enseignant id = 2>
            <personne id = 3>
               <nom>toto</nom>
            </personne>
         </enseignant>
      </ appartient >
   </voiture>
</document>
```


Semantic document

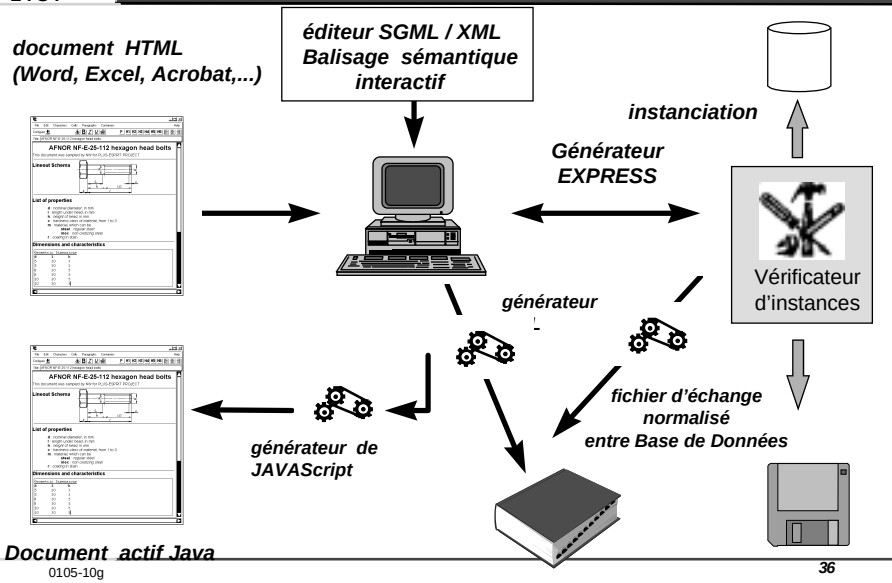



```
#10 = CLASS_BSU('
       NF-E-25-112,
       '001,
       #5');
```

```
#11 = COMPONENT_CLASS (
#10,
$, '001', #72,
TEXT('Class ...'),
$, $, $, $,
(#90, #100),
(), $, (),(), $);
```

A la fois traitable par machine ET lisible

Exemple d'intégration des outils



Conclusion

- But conférence: Discuter r**eprésentation** et **présentation** des données objets, en particulier de leur dimension dynamique
- Spécification de données: représentation où le modèle et les contraintes sont représentées formellement, i. e., "machinables"
  - **identifié deux types d'invariants:**
    - fonctionnels: un ***attribut*** résulte d'une ***fonction***
    - logiques: ***prédicat*** est vérifié
- Toute grammaire HC peut se représenter par une spécification de données
  - **même controle structurel**
  - **controle de la sémantique statique**
  - **si invariant fonctionnels ==> capacité d'interpréter la cns procédurale**

  ==> possibilité de représenter sous forme de données les comportements dynamiques, sans "effet de bord"

Conclusion

- *Base de données "fonctionnelle"*:
  - Données +
  - Attributs dérivés +
  - Invariants logiques

  capacité de représenter completement des ensembles d'objets du monde réel, y compris leur comportements dynamiques
- Insuffisance de XML pour échanger des données à traiter.
  Proposition:
  - XML pour la présentation
  - Formalisme de spécification pour la sémantique
- Mapping
  - semi-automatique au niveau modèle (spec → DTD)
  - automatique au niveau instance

  Résultat = document sémantique, lisible et valide.

0105-10g