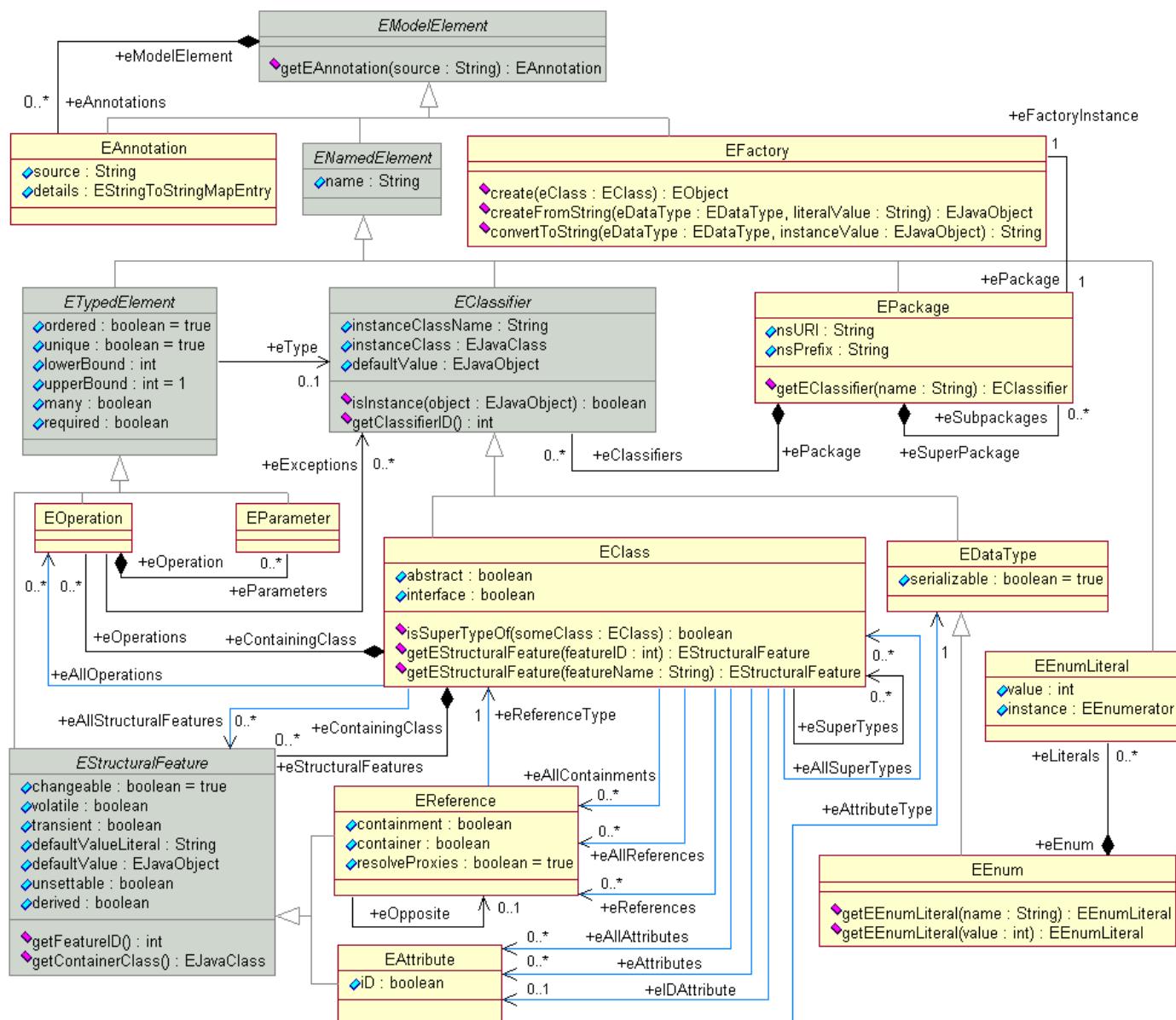


Ingénierie des Modèles

Méta-modélisation, DSL, M2M, M2T, T2M, MDA, XML
OCL, AQL, Acceleo, QVTo, ATL, XSLT, XPath



Emmanuel Grolleau
grolleau@ensma.fr

Bureau B414

Octobre 2022

Nombreux transparents repris et adaptés du
cours de Yassine Ouhammou

Ingénierie Dirigée par les Modèles

Emmanuel Grolleau

B414

Supports de cours en grande partie basés sur ceux de

Yassine Ouhammou

Ingénierie Dirigée par les Modèles

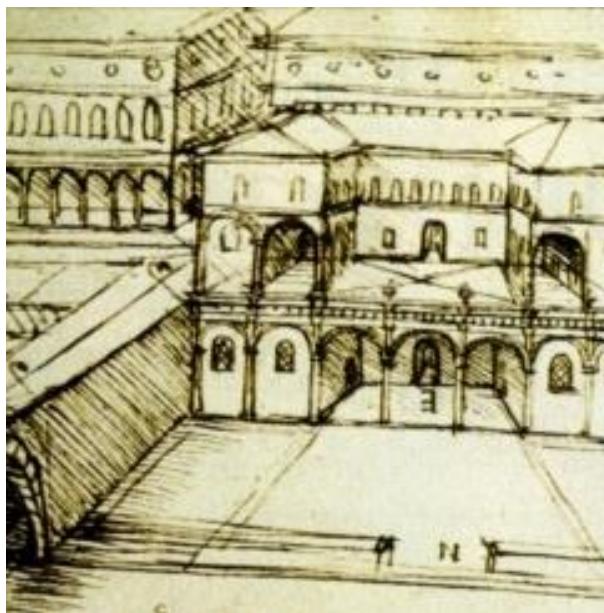
« De tout temps l'Homme »



- A cherché à concevoir des villes de façon holistique
 - Holistique: qui traite le tout plutôt qu'une partie
 - Entreposage du grain, et des réserves de nourriture, défenses de la ville, support pour le transport et le commerce, affichage de la puissance du pouvoir (palace), préparation des morts, lieux de culte,... et plus tard gestion des eaux, des déchets, etc.
- On peut même imaginer qu'il en était de même pour les habitats précédant la ville
- Révolution industrielle (fin 19^{ème})
 - Taylorisme, optimisation de la production
 - Prise en compte de différents aspects : gestion des coûts, de l'approvisionnement, etc.
- 2^{ème} guerre mondiale
 - Vision holistique centre de commande – terrain
 - Elargie à la gestion des ressources, de la fabrication, etc.
- A partir de 1960, de plus en plus d'informatique dans les systèmes
 - Facteur humain
 - Formalisation des notions d'exigences, de traçabilité, etc.
 - Emergence des systèmes critiques

Ingénierie Dirigée par les Modèles

Des « génies » du passé capables de conception holistique

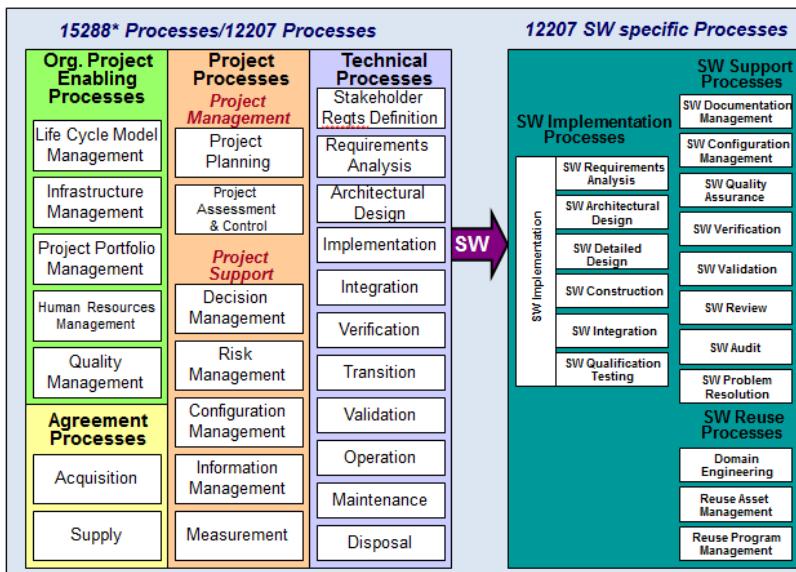


Ingénierie Dirigée par les Modèles

- Léonard de Vinci : artiste, ingénieur mécanique, concepteur d'engins de guerre, de systèmes de défense, urbaniste
 - Projet de capitale de la France à Romorantin
- Nombreux scientifiques de l'antiquité à la renaissance
 - Eratosthène : astronome, géographe, philosophe, mathématicien
 - Archimède : mathématicien, physicien, ingénieur, défendra Syracuse attaquée par Rome durant 3 ans
- De nos jours, se détourner 10 ans d'un domaine rend obsolète les connaissances avancées
- Il n'y a plus de génie transdisciplinaire capable d'aborder un problème de façon holistique

3

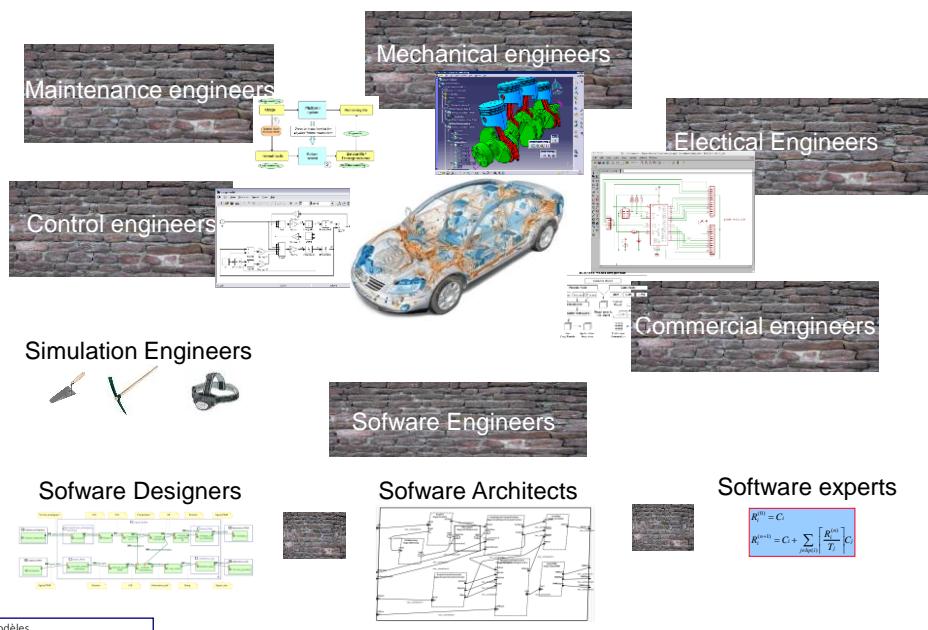
Ingénierie système et ingénierie logicielle



*Based on 15288:2008. See 15288:2015 for the most recent SE Life Cycle Processes

Source sebokwiki.org

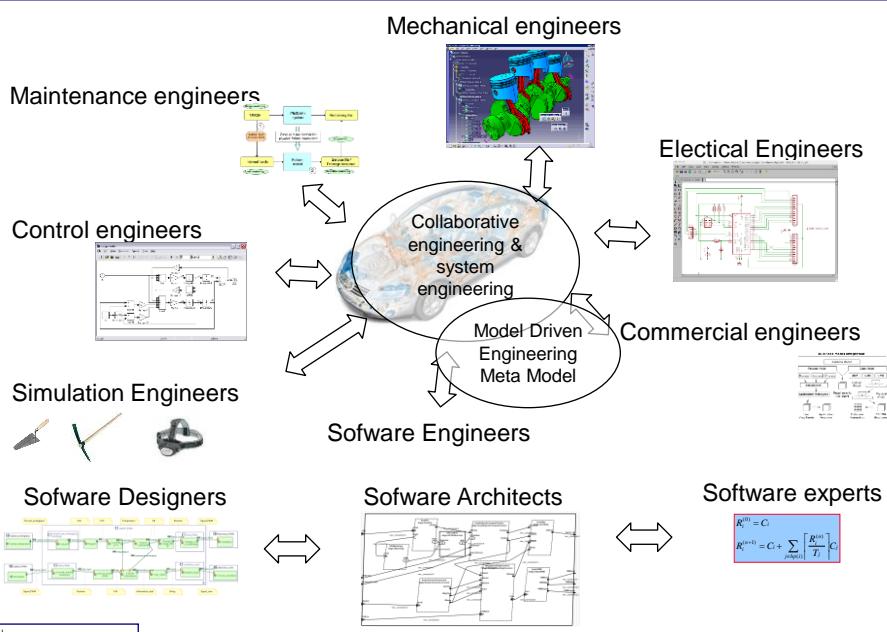
Ingénierie logicielle pour les systèmes cyber physiques jusqu'à aujourd'hui



Ingénierie Dirigée par les Modèles

5

Ingénierie logicielle pour les systèmes cyber physiques demain

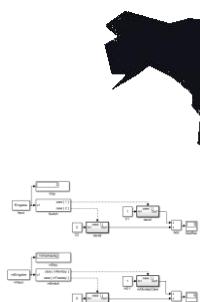


Ingénierie Dirigée par les Modèles

6

Ingénierie Dirigée par les Modèles (Model Driven Engineering)

- Sens différents suivant le côté de l'Atlantique



Approche basée métamodélisation
Inspirée/dérivée de Model Driven Architecture (MDA)

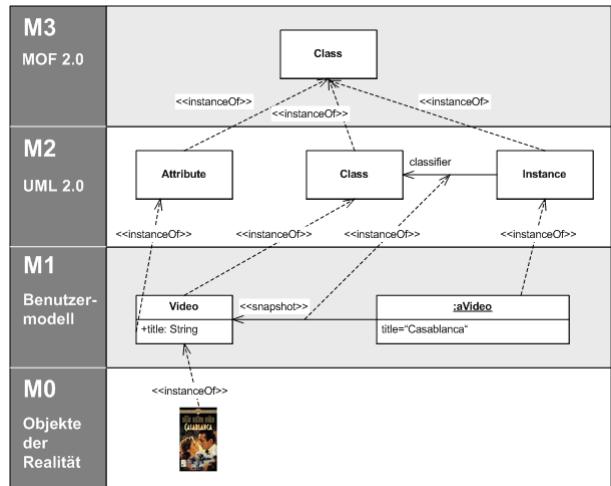
Matlab Simulink + génération de code

Model Driven Architecture (MDA)

- Norme OMG (Object Management Group) lancée en 2001
 - Computation-Independent Model (CIM)
 - Platform-Independent model (PIM)
 - Platform-Specific model (PSM)
- Cela doit vous rappeler des choses concernant Arcadia/Capella où on peut retrouver des couches similaires dans les couches
 - Opérationnel et Système <-> CIM
 - Système et Logique <-> PIM
 - Physique <-> PSM
- La MDA s'appuie grandement sur UML (Unified Modeling Language), dont les différents diagrammes, métamodèles permettant d'exprimer des points de vue modèles du système, sont mis en relation via un métaméta-modèle commun appelé MOF (Meta-Object Facility). Le système produit est une instance des modèles du système.
- La sérialisation se base sur XML Metadata Interchange (XMI)

Niveaux de modélisation avec UML

- M0 : Objet issu de la conception
- M1 : modèle de l'objet, exprimé conformément à une syntaxe et sémantique définies dans un modèle de modèle : un métamodèle
- M2 : ce métamodèle exprimant les constructions premières et leur sémantique
- M3 : modèle de métamodèle M2, autrement dit métaméta-modèle décrivant les éléments de construction possible utilisables au niveau M2
- Pourquoi le niveau M3 ?
 - Permet de faciliter les transformations entre différents métamodèles (on peut parler de langages métier)
- On dit qu'un modèle se conforme à un métamodèle
- A l'instar d'un code source qui se conforme à la grammaire d'un langage de programmation



Grammaire et langage

```
Axiome: s
s -> decl*
decl -> varDecl | funDecl
varDecl -> typeSpec varDeclList ;
funDecl -> typeSpec? ID ( params )
statement
typeSpec -> int | bool | char
Etc.
```

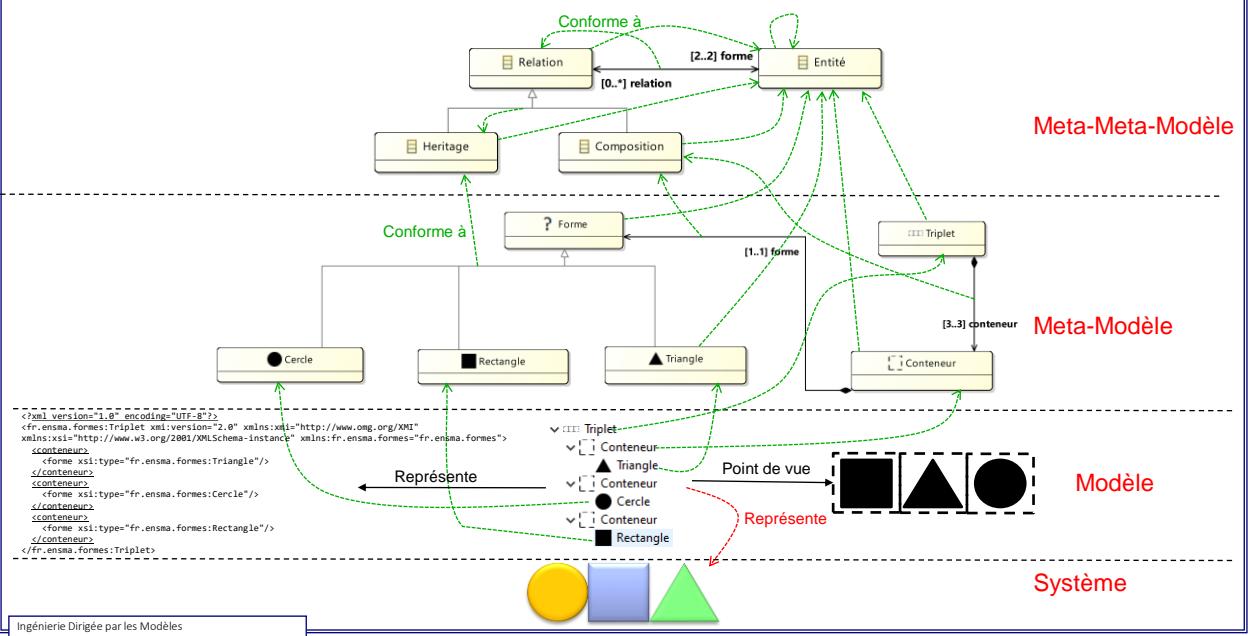
Langage (grammaire hors contexte)

```
int f1(int b, int c) {
    int i;
    return 3+i*4;
}
```

Code source se conformant au langage C

- Les langages de programmation « classiques », sauf scories historiques (comme C et C++) ne sont pas exprimés dans un métalangage commun
 - En fait, si, un peu, ce qui permet aux compilateurs écrits par exemple en C de compiler d'autres langages
 - Cependant aucun lien sémantique n'existe entre le int C et le integer Ada
- C'est dommage, imaginez comment on pourrait simplement transformer des codes source d'un langage à l'autre
- Imaginez que cela soit le cas pour un métalangage commun à toutes les langues
 - La langue commune de la Terre du Milieu imaginée par JRR Tolkien
- Même si c'était le cas, les constructions ne pourraient toutes se traduire
 - Langage compilé vs interprété, impératif vs fonctionnel, avec ou sans ramasse-miettes, etc.

Modèle, meta-modèle, meta-meta-modèle : quesako ?



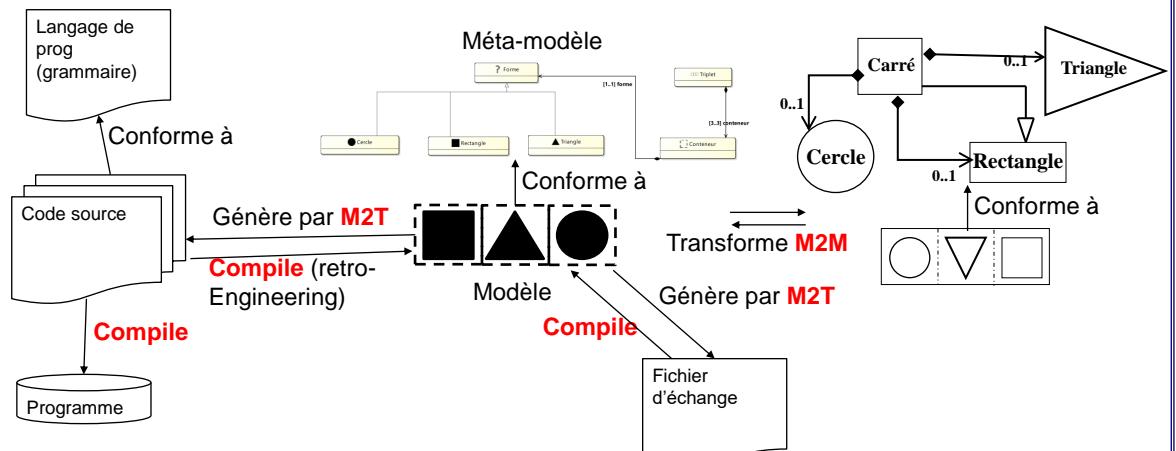
11

De la bonne compréhension du schéma précédent

- Qu'a-t-on choisi d'ignorer dans la modélisation des systèmes que notre modèle peut représenter ?
- Sachant qu'on pourrait construire un modèle de conteneur contenant une entité Forme, que manque-t-il dans notre meta-meta-modèle qui permettrait de ne pas autoriser cette modélisation ?
- La composition et la relation, utilisées dans le meta-modèle et le meta-meta-modèle sont-elles vraiment conformes à la relation définie dans le meta-meta-modèle ? Si non, pourquoi ?
- Sachant que les diagrammes du meta-modèle et du modèle ont été modélisés sous Eclipse, pourquoi le code XMI du meta-modèle n'est pas montrable sur ce slide (indépendamment de l'espace requis) ?
- Qu'en déduire concernant le lien entre outillage et meta-meta-modèle ?

Du texte et des modèles

- Dans le monde informatique, hors bases de données, on trouve principalement deux catégories de textes
 - Les données d'échange
 - Les codes source de programmes
- L'IDM offre de nombreux outils permettant des manipulations et représentations graphiques de ces textes



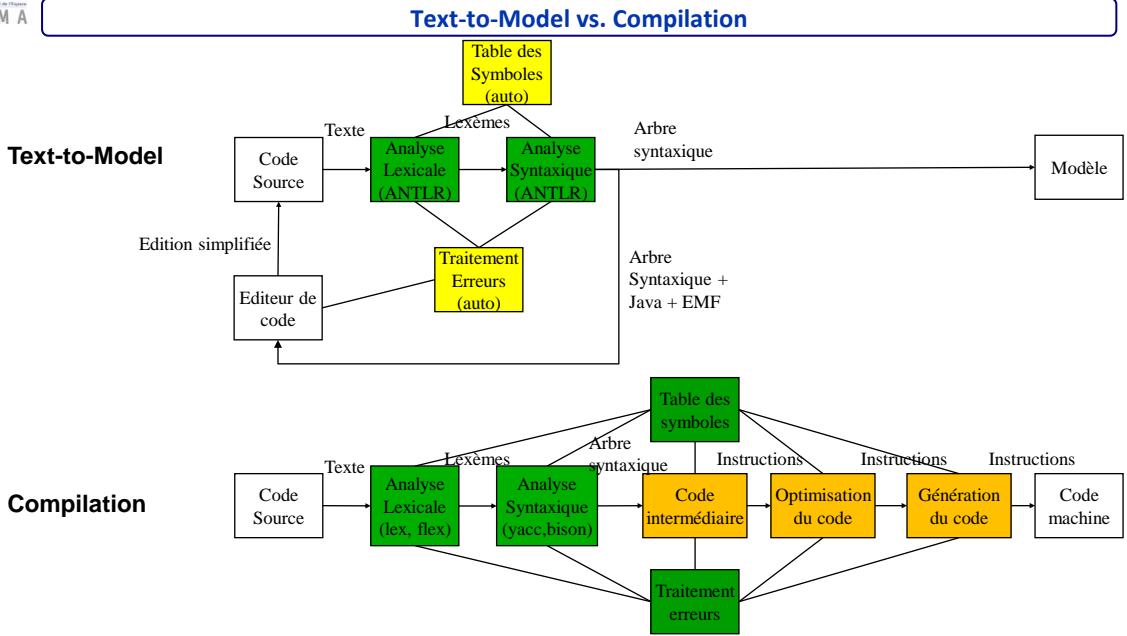
13

Ce que l'IDM et EMF offrent

- Un framework principalement basé Java
- De nombreuses constructions existantes, code helpers, etc. facilitant le développement d'environnement de son propre DSL, ou de suivi de ses propres processus
- Excellente solution pour proposer un client adapté aux données et processus de son service/entreprise
- Les outils sont aujourd'hui principalement faits pour développer des **clients lourds**
 - Ressemble à Eclipse, Yakindu, Capella, etc.
- Des solutions émergent pour des **clients légers** sur navigateur
 - Nécessite généralement soit un abonnement sur un serveur cloud fournissant le service
 - ❖ Par exemple solutions Obeo designer sur Sirius
 - Soit de déployer son propre serveur – mises à jour, maintenance, choix d'accès local à l'entreprise ou extérieur avec ce que cela implique en termes de sécurité
- Pensez à réfléchir et si possible discuter de vos choix technologiques en PFE
- Faites de la veille technologique autant que possible avant de choisir
- « il y avait un client python – mysql avant » n'est pas une raison de partir sur la même technologie

De l'interprétation de la grammaire

Text to Model ou Text to Program – les premiers pas menant à la compilation



Principes de l'analyse lexicale

- Principe: lecture du code source caractère par caractère afin de reconnaître des lexèmes (anglais: *tokens*) le code ainsi décomposé sera plus facile à traiter par l'analyseur syntaxique
- Reconnaissance de lexèmes
 - mots clés du langage *begin, if, true, ...*
 - opérateurs *+, -, ...*
 - symboles *>, >=, .., ; ...*
 - Littéraux *54, -3.2E-5, ...*
 - identificateurs
 - ❖ Noms de variables, fonctions, ...
- Les lexèmes constants sont simples à reconnaître
- Mais comment reconnaît on les identificateurs et les littéraux ?
- On doit savoir les décrire : **expressions régulières**, et tester qu'un mot respecte une expression régulière : **automates finis**

Les langages reconnaissables par expression régulière (ou equiv. Automates finis) sont classifiés par Chomsky sous le nom **Langages de type 3**. On les appelle **langages réguliers**,

Les langages réguliers dans la théorie des langages

- Expression régulière=outil de description lexicale
- **Alphabet** : Σ formé de symboles (exemple: $\Sigma=\{a,b,c\}$)
- **Mot** : suite de symboles de Σ (exemple: abcba), le mot vide se note ϵ
- Notions de **sous-mot**, **préfixe**, **suffixe** sur les mots (bcb sous-mot de abcba, ab préfixe de abcba, ba suffixe de abcba)
- **Langage L** ensemble de mots
- Opérations sur les langages:
 - **Concaténation** (dit aussi produit) $L_1L_2=\{xy \mid x \in L_1, y \in L_2\}$, $L\{\epsilon\}=\{\epsilon\}L=L$, $LL\dots L=L^i$
 - **Union** $L_1 \cup L_2 = \{x \mid x \in L_1 \text{ ou } x \in L_2\}$, $\emptyset \cup L = L \cup \emptyset = L$, souvent noté $L_1 | L_2$
 - **Fermeture (étoile)** $L^* = \bigcup_{i=0}^{\infty} L^i$
 - **Complément**
 - **Intersection**
- Les expressions régulières expriment les **langages réguliers**
- Les langages réguliers sont **fermés** par les opérations concaténation, union et étoile
 - Soient L_1 et L_2 langages réguliers, les langages L_1L_2 , $L_1|L_2$, L_1^* sont réguliers

Notations des opérations usuelles

- ϵ est le mot vide, mais n'est pas utilisé dans la représentation des E.R.
- Chaque lettre de l'alphabet est une E.R.
- Dans tout outil traitant des E.R., les symboles de l'alphabet sont vus comme ordonnés (ex. ASCII), et chaque symbole a un prédecesseur et un successeur, ce qu'on utiliser pour faciliter l'expression d'unions de symboles
 - On peut définir des intervalles de caractères: $[a-z]$ représente $a|b|c|...|z$ et $[a-z0-9]$ représente $a|...|z|0|...|9$
 - Si les 128 premiers caractères ASCII sont universels, on a souvent de mauvaises surprises avec les caractères régionaux (é, è, ä, ê, ç, etc.) pas toujours pris en charge par les outils
- Soient L une E.R.
 - L^+ représente LL^* (au moins un)
 - $L^{[i,j]}$ représente $L^i|L^{i+1}|...|L^j$ (entre i et j répétitions)
 - ❖ L^i exactement i fois
 - ❖ L^i au moins i fois
 - $L^?$ représente $\epsilon|L$ (optionnel)

Notations des opérations usuelles/suite

- Le . représente tout symbole de l'alphabet, ^ le début du mot en entrée, \$ sa fin
 - Le caractère . doit être échappé \.
 - Le caractère d'échappement est \, et \ suivis d'un symbole prend une sémantique particulière
 - ❖ Si le caractère qui suit a habituellement une sémantique, l'\ la retire
 - ✓ \\ représente le symbole \, idem pour \(), \[, \], \], \+, *, \", \', \\$, \^
 - ❖ Certains caractères non affichables sont représentés par une combinaison avec \
 - ✓ \n retour chariot, \r retour à la ligne, \t tabulation, \b blanc, et parfois macros comme \w word
- Soient a et z deux symboles de l'alphabet, ordonnés tels qu'ils pourraient l'être dans l'ASCII
 - $[a-z]$ représente l'union de chaque singleton de symbole entre a et z
 - $[\wedge a-z]$ représente tout singleton de symbole sauf l'intervalle [a-z]
 - ❖ $[\wedge @]+$ toute chaîne de caractères de taille au moins 1 ne contenant pas @
 - ❖ Le caractère . perd sa sémantique entre crochets
 - $[az]$ est équivalent à $a|z$
 - $[a-zA-Z]$ représente donc $[a-z] | [A-Z]$
 - ❖ $[_a-zA-Z][_a-zA-Z0-9]^*$ identifiant dans de nombreux langages de programmation
 - Entre crochets, la plupart des caractères spéciaux perdent leur sémantique spéciale

Exemples d'expressions régulières

- Reconnaître un verbe du premier groupe à l'infinitif

- Reconnaître un verbe du 1er ou 2^e groupe à l'infinitif

- Reconnaître une phrase du type sujet verbe complément terminée par un point

- Identifier les composants d'un URI (*Uniform Resource Identifier*)

- Adresse email simple

Outils de reconnaissance d'expressions régulières

- Bibliothèques regexp dans les langages de programmation

- re en python
- java.util.regex.Matcher et java.util.regex.Pattern en java
- Gnat.regexp en Ada
- <regex> en C++
- Etc.

- Expressions régulières pour création de lexèmes en vue d'analyse syntaxique

- Outils historiques de compilation lex et flex
- Partie tokens d'ANTLR

Exercice

- En python, reconnaître une chaîne donnée en entrée sous la forme « prénom nom note » utilisant un espace comme séparateur, créer un dictionnaire python correspondant au schéma JSON donné, et l'afficher

– Etape 1 : sur <https://regex101.com/> préparer l'expression régulière, puis générer le code python

❖ Remarquer comme, sans passer par le \w en unicode, les accents sont complexes à gérer

– Aller sur <https://www.online-python.com/> pour adapter le programme de façon à générer le JSON

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "array",
    "items": [
        {
            "type": "object",
            "properties": {
                "nom": {
                    "type": "string"
                },
                "prénom": {
                    "type": "string"
                },
                "note": {
                    "type": "number"
                }
            },
            "required": [
                "nom",
                "prénom",
                "note"
            ]
        }
    ]
}
```

Emmanuel Grolleau 8.5
Emmanuel-Miçkaël Grolleau--Richard 18 → {
 "nom": "Grolleau",
 "prénom": "Emmanuel",
 "note": 8.5
 }
 "nom": "Grolleau--Richard",
 "prénom": "Emmanuel-Miçkaël",
 "note": 18
}]

Pour les plus rapides, changer afin que la , soit aussi acceptée comme séparateur décimal, et cette fois ;\t ou , (csv) sont les séparateurs à la place de l'espace. Travailler sur des fichiers au lieu de chaînes test et affichage.

Limitations vs. langages de type 3

- En police 4 à gauche, l'expression régulière implémentant la RFC 822 (source Paul Warren) en Perl
- En fait, une adresse email est bien plus complexe qu'il n'y paraît au premier abord
- On a le droit à de nombreux caractères, avec des limites pour certains (par exemple, le « . » est autorisé mais pas deux fois à la suite), usage d'accents, etc.
- Mais le pire... c'est que cette expression n'implémente pas 100% de la RFC 822
- On peut en effet utiliser des commentaires imbriqués...
- Et ça... ce n'est pas de type 3, mais requière du type 2

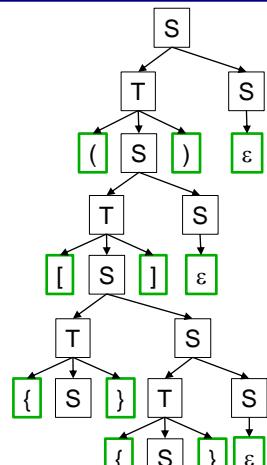
Limitations

- Les expressions régulières ne permettent pas d'exprimer des langages classiques en informatique, le langage de Dyck
 - Mots parenthésés, commentaires imbriqués, et imbrication de structures de contrôle
 - Exemple $\Sigma=\{(),[]\}$ on veut pouvoir écrire tout mot licite tel que le nombre de parenthèses ouvrantes est toujours \geq au nombre de parenthèses fermantes, pour arriver à l'égalité à la fin du mot
 - ❖ $()$, $(())$, $(((())))$ sont des mots de ce langage
 - ❖ $()$, $((())()$ ne sont pas des mots de ce langage
 - Exemple $\Sigma=\{([{}],{})\}$
 - ❖ $[{{}}]$ est un mot de ce langage
 - ❖ $[{{}}]$ même si on respecte la règle énoncée du nombre d'ouvrants \geq nombre de fermants du même type, ce mot n'est pas un mot du langage de Dyck. Cette règle doit s'appliquer à chaque niveau hiérarchique.
- Problème: les langages réguliers ne savent pas compter
 - Le langage $a^n b^n$ pour n quelconque n'est pas reconnaissable par E.R. car pas non plus reconnaissable par automate fini
- Les langages de programmation intègrent le langage de Dyck
 - Blocs imbriqués, if then else imbriqués, boucles imbriquées, etc.

Ingénierie Dirigée par les Modèles

Grammaire des langages de Dyck

- Axiome $S: TS|\epsilon$
- $T: (S)|[S]|S$
- Mot $([{}])$ reconnu par la grammaire, car on peut construire un arbre à partir de l'axiome dont les feuilles sont les terminaux (ici les lettres) du mot
- Comment construire cet arbre à partir de la grammaire ?
- Deux familles de méthodes
 - Analyse descendante, plus aisée à comprendre, puisqu'on construit l'arbre au fur et à mesure qu'on lit le mot du langage en entrée de façon à produire le prochain terminal dans la feuille la plus à gauche. On parle d'analyse LL.
 - Analyse ascendante plus complexe, où on construit l'arbre à partir du bas, en réduisant les feuilles à leur nœud parent. Strictement plus puissante que l'analyse descendante. On parle d'analyse LR.
- Les grammaires analysables en LL(1) sont strictement contenues dans les grammaires analysables par LR(1) – (1) signifie on ne regarde qu'un caractère



Ingénierie Dirigée par les Modèles

De l'analyse syntaxique des langages de type 2

Analyse syntaxique

Ingénierie Dirigée par les Modèles

Classification des langages par Chomsky

Type 0
Langages/Grammaires récursivement énumérables
Tout est autorisé à gauche et à droite de la règle de grammaire

Type 1
Langages/Grammaires contextuels
 $BXC : BEAdC$ (X donne Eaq si entre B et C)
Peut tenter de décrire le langage naturel

Type 2
Langages/Grammaires hors contexte
 $A : BaC | B$

Type 3
Langages/grammaires réguliers ou rationnels
 $A : aB | a$
équivalent expressions régulières

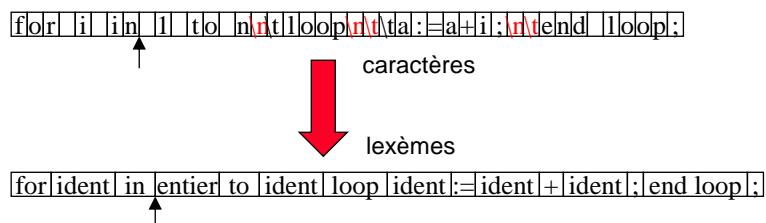
Ingénierie Dirigée par les Modèles

Classification des méthodes d'analyse syntaxique (parsers) par Donald Knuth

- LL(k) : scan from the left, using left productions
- LR(k) : scan from the left, using right reductions
- Beaucoup moins utilisés car partir de la fin, ce serait étrange
 - RL(k) : scan from the right, using left reductions
 - RR (k) : scan from the right, using right productions
- (k) signifie “looking k characters ahead of the terminal symbols”
- LL(1) par exemple ne peut pas regarder au delà du lexème courant pour choisir quelle règle dériver pour générer ce lexème à gauche de l'arbre de dérivation.
- Si la grammaire est recursive à gauche, aucun moyen d'analyser
 - expr: expr + expr
 - La récursion peut être indirecte
 - ❖ expr: produit + somme | somme
 - ❖ produit: expr | NUM * NUM
- Il existe (de nombreux) langages exprimables par grammaire qui ne sont pas LL(1)
- Nous utiliserons ANTLR qui est LL(*) : au 21^{ème} siècle, on se permet du backtracking semble-t-il

Ingénierie Dirigée par les Modèles

De l'analyse lexicale à l'analyse syntaxique



Ingénierie Dirigée par les Modèles

Grammaires hors-contexte (gcf)

- Définition: grammaire formelle $G=(V,\Sigma,\rightarrow,S)$
 - V ensemble fini de symboles **non terminaux**, exemple $\{A,B,C,\dots,S,\dots\}$
 - Σ l'alphabet des symboles **terminaux**, exemple $\{a,b,c,\dots\}$
 - $\rightarrow: (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ ensemble de **règles de production**, exemple $AabC \rightarrow DeF$
 - $S \in V$ **axiome ou symbole initial**
- Grammaire **hors-contexte ou context-free (gcf)**
 - grammaire formelle $G=(V,\Sigma,\rightarrow,S)$ où
 - $\rightarrow: Vx(V \cup \Sigma)^*$, exemple $A \rightarrow bBc$
- gcf **régulière (ou linéaire) droite**
 - $\rightarrow: Vx\Sigma^*\{\varepsilon \cup V\}$, exemple $A \rightarrow b, A \rightarrow abcC$
- gcf **régulière (ou linéaire) gauche**
 - $\rightarrow: Vx\{\varepsilon \cup V\}\Sigma^*$, exemple $A \rightarrow b, A \rightarrow CabC$
- L'opérateur $|$: $A \rightarrow a|b$ signifie $A \rightarrow a, A \rightarrow b$
- Définition
 - Le langage $L(G)$ d'une grammaire est l'ensemble des mots que l'on peut produire en dérivant les règles de production

Ingénierie Dirigée par les Modèles

Exemple de gcf

- Soit une grammaire $G=(V,\Sigma,\rightarrow,S)$
 - $V=\{S,A\}$
 - $\Sigma=\{0,1,\dots,9\}$
 - \rightarrow défini par
 - ❖ $S \rightarrow S+A \mid A$
 - ❖ $A \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid \dots \mid 9$
- $L(G)$ est constitué de toute somme de chiffres
 - $0,1,\dots,9, 0+0, 0+1,\dots,9+9, 0+0+0, 0+0+1, \dots$ sont des mots de $L(G)$

Ingénierie Dirigée par les Modèles

Cas particulier des gcf régulières

➤ Les langages réguliers à droite sont définis par une grammaire régulière

➤ Chomsky les a définis par le terme « **langage de type 3** »

➤ Exercice: Créer l'AF correspondant à

- $S \rightarrow aB$
- $B \rightarrow cS \mid bC$
- $C \rightarrow a$

➤ Théorème de Kleene

- La classe des langages de type 3 est la plus petite famille des langages formels qui contient les langages finis et qui est fermée par union, concaténation, étoile

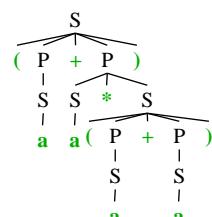
➤ Ces grammaires, nous les avons déjà vues, elles sont représentables par des expressions régulières

Reconnaissance d'un mot par une gcf

➤ A toute dérivation on associe un arbre (c.f. cours de 1ère année)

➤ Exemple:

- (1) $S \rightarrow (P+P)$
- (2) $S \rightarrow a$
- (3) $P \rightarrow S^*S$
- (4) $P \rightarrow S$



➤ $(a+a^*(a+a))$ est-il reconnu par la gcf ?

➤ Dérivation gauche

- $S \Rightarrow (P+P) \Rightarrow (S+P) \Rightarrow (a+P) \Rightarrow (a+S^*S) \Rightarrow (a+a^*S) \Rightarrow (a+a^*(P+P)) \Rightarrow (a+a^*(S+P)) \Rightarrow (a+a^*(a+P)) \Rightarrow (a+a^*(a+S)) \Rightarrow (a+a^*(a+a))$

➤ Dérivation droite

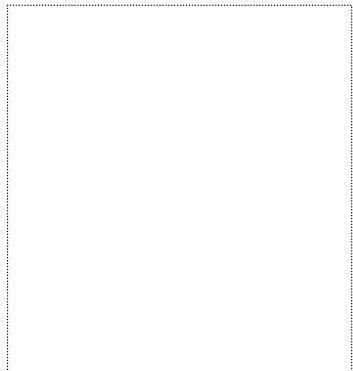
- $S \Rightarrow (P+P) \Rightarrow (P+S^*S) \Rightarrow (P+S^*(P+P)) \Rightarrow (P+S^*(P+S)) \Rightarrow (P+S^*(P+a)) \Rightarrow (P+S^*(S+a)) \Rightarrow (P+S^*(a+a)) \Rightarrow (P+a^*(a+a)) \Rightarrow (S+a^*(a+a)) \Rightarrow (a+a^*(a+a))$

➤ Pour cette grammaire, dans les deux cas on obtient le même arbre

Ambiguïté

- Soit G une gcf, s'il existe au moins un mot de $L(G)$ qui possède au moins deux arbres de dérivation différents, alors G est dite **ambiguë**.
- Si toutes les grammaires engendrant un langage L sont ambiguës, alors L est ambigu.
- Par construction, les langages informatiques ne sont pas ambigu.
- Exemple:
 - $S \rightarrow_S S$ alors S sinon S
 - $S \rightarrow_S S$ alors S
 - $S \rightarrow_{ident}$
- Exercice : construire deux arbres de dérivations différents pour :

si ident alors si ident alors ident sinon ident
- On comprend mieux la différence de grammaire entre C, et Ada



Suppression de l'ambiguïté

- Soit la grammaire
- $B \rightarrow B \text{ et } B \mid B \text{ ou } B \mid \text{ vrai } \mid \text{ faux } \mid \text{ non } B \mid B=B \mid (B)$
- Donner trois arbres de dérivation différents pour
 - non vrai et faux ou vrai et vrai
- Transformer la grammaire afin que les opérateurs suivent les priorités suivantes: non > et > ou > = et que ces opérateurs soient associatifs à gauche puis donner l'arbre de dérivation de l'expression ci-dessus



Technique usuelle de suppression de l'ambiguïté : priorité et associativité

Ingénierie Dirigée par les Modèles

Propriétés des grammaires

- Notation: \Rightarrow est la fermeture transitive de \rightarrow
- Réversibilité: $A \rightarrow x, B \rightarrow x \Rightarrow A=B$
- Sans cycle: Il n'existe pas $A / A \stackrel{\pm}{\rightarrow} A$
- Sans chaîne (ou sans règle superflue): Il n'existe pas $A, B / A \stackrel{\pm}{\rightarrow} B$
- Forme normale de Chomsky: $A \rightarrow a, A \rightarrow BC, A \rightarrow \epsilon$
- Forme normale de Greibach: $A \rightarrow au$ avec $a \in \Sigma, u \in V^*$
- Récursivité à gauche $A \stackrel{\pm}{\rightarrow} Au$ avec $u \in V^*$
- Sans symbole inaccessible: $\forall A \in V, \exists x, y \in \{\epsilon \cup V \cup \Sigma\}^*/S \stackrel{*}{\rightarrow} xAy$

Ingénierie Dirigée par les Modèles

Suppression de la récursivité à gauche

- Gênant pour la plupart des algorithmes de compilation
- A tout langage $L(G)$ défini par une gcf récursive à gauche, on peut faire correspondre une grammaire G' non récursive à gauche telle que $L(G)=L(G')$
- Principe: construction d'une grammaire équivalente telle que toute règle de la forme $A_i \rightarrow A_j a_k$ soit telle que $j \geq i$
- Algorithme de suppression de la récursivité à gauche

Numéroter les symboles de V , $V=\{A_1, \dots, A_n\}$

Pour i de 1 à n faire

(invariant de boucle: aucune règle de A_p , $p < i$, ne peut commencer par une variable A_k avec $k \leq p$)

Pour j de 1 à $i-1$ faire

Soit $A_i \rightarrow A_j a_k$ avec $A_j \rightarrow a_1 | \dots | a_m$

Remplacer la règle par

$A_i \rightarrow a_1 a_k | \dots | a_m a_k$

Soit $A_i \rightarrow B_1 | B_2 | \dots | B_m | A_j a_1 | \dots | A_j a_k$

Remplacer la règle par

$A_i \rightarrow B_1 | B_2 | \dots | B_m | B_i A'_i | \dots | B_m A'_i$

$A'_i \rightarrow a_1 | \dots | a_k | a_1 A'_j | \dots | a_k A'_j$

Ingénierie Dirigée par les Modèles

Exercice: suppression de la récursivité à gauche

- Eliminer la récursivité à gauche de la grammaire ci-dessous
- $A \rightarrow BC | a ; B \rightarrow CA | Ab ; C \rightarrow AB | CC | a$

Ingénierie Dirigée par les Modèles

Analyse syntaxique descendante LL(1)

- Principe: construire un arbre à partir de l'axiome.
- Il est difficile de choisir une règle, donc il existe une technique permettant de guider ce choix
- Soit une gcf $G=(V,\Sigma,\rightarrow,S)$
- $\text{First}_k(\alpha)=\{a \in \Sigma^* / \alpha \rightarrow a\beta \text{ avec } |a|=k, \beta \in V^* \text{ ou } \alpha \rightarrow a \text{ avec } |a|<k\}$
 - Ensemble des symboles terminaux de longueur $\leq k$ préfixes de ce qu' α peut générer
- $\text{Follow}_k(\beta)=\{a \in \Sigma^* / A \rightarrow \alpha\beta\gamma \text{ avec } a \in \text{First}_k(\gamma)\}$
 - Ensemble des terminaux de longueur $\leq k$ pouvant suivre β
- Exemple
 - $E \rightarrow TM ; T \rightarrow FN ; M \rightarrow \epsilon | +TM ; N \rightarrow \epsilon | *FN ; F \rightarrow (E) | a$

	First_1	Follow_1
E	a, ($\epsilon,)$	
T	a, ($\epsilon, +,)$	
M	$\epsilon, +$ $\epsilon,)$	
N	$\epsilon, *$ $\epsilon, +,)$	
F	a, ($\epsilon, +, *,)$	

Calcul de First_1

- $\text{First}_1(A) = \emptyset, \text{First}_1(a)=\{a\}, \text{First}_1(\epsilon)=\{\epsilon\}$
- Pour toute règle $A \rightarrow B_1B_2\dots B_k$
 - $i=0;$
 - Répéter
 - ❖ $i=i+1$
 - ❖ $\text{First}_1(A)=\text{First}_1(A) \cup (\text{First}_1(B_i) \setminus \{\epsilon\})$
 - Jusqu'à ce que $\epsilon \notin \text{First}_1(B_i)$
 - Si $i=k$ et $\epsilon \in \text{First}_1(B_k)$ alors $\text{First}_1(A)=\text{First}_1(A) \cup \{\epsilon\}$
- Extension: $\text{First}_1(B_1B_2\dots B_k)$ est calculé de la même façon
- Exercice: calculer les ensembles First_1 pour la grammaire suivante
- $S \rightarrow BS | \epsilon$
- $B \rightarrow id := id | \text{if id then BE}$
- $E \rightarrow I | \text{else BI}$
- $I \rightarrow \text{endif}$



Calcul de Follow₁

- $\text{Follow}_1(A) = \emptyset, \text{Follow}_1(S) = \{\epsilon\}$
- Pour toute règle $B \rightarrow \alpha A \gamma$
 - $\text{First}_1(\gamma) \setminus \{\epsilon\} \subseteq \text{Follow}_1(A)$
 - Si $\epsilon \in \text{First}_1(\gamma)$ alors $\text{Follow}_1(B) \subseteq \text{Follow}_1(A)$

➤ Exercice: calculer les ensembles Follow_1 pour la grammaire

- $S \rightarrow BS | \epsilon$
- $B \rightarrow \text{id} := \text{id} | \text{if id then BE}$
- $E \rightarrow I | \text{else BI}$
- $I \rightarrow \text{endif}$

Analyse descendante

- Une grammaire est dite LL(k) si, lors d'une analyse descendante, le choix de la règle à appliquer peut se faire en connaissance des k prochains terminaux.
- En particulier, une grammaire est LL(1) si
 - Pour toute règle $A \rightarrow \alpha_1 | \alpha_2$, $\text{First}_1(\alpha_1 \cdot \text{Follow}_1(A)) \cap \text{First}_1(\alpha_2 \cdot \text{Follow}_1(A)) = \emptyset$
- Les grammaires suivantes sont-elles LL(1) ?
 - $E \rightarrow TM ; T \rightarrow FN ; M \rightarrow \epsilon | +TM ; N \rightarrow \epsilon | *FN ; F \rightarrow (E) | a$
 - $S \rightarrow BS | \epsilon ; B \rightarrow \text{id} := \text{id} | \text{if id then BE} ; E \rightarrow I | \text{else BI} ; I \rightarrow \text{endif}$
- Table d'Analyse Prédictive (TAP)
- Pour toute production $A \rightarrow \alpha$, ajouter cette règle dans les cases $\text{TAP}(A, \text{First}_1(\alpha))$, et si $\epsilon \in \text{First}_1(\alpha)$, l'ajouter aussi dans les cases $\text{TAP}(A, \text{Follow}_1(A))$, dans ce cas faire correspondre $\$$ à ϵ ($\$$ signifie fin de l'entrée)

	+	*	()	a	\$
E	X	X	$E \rightarrow TM$	X	$E \rightarrow TM$	X
T	X	X	$T \rightarrow FN$	X	$T \rightarrow FN$	X
M	$M \rightarrow +TM$	X	X	$M \rightarrow \epsilon$	X	$M \rightarrow \epsilon$
N	$N \rightarrow \epsilon$	$N \rightarrow *FN$	X	$N \rightarrow \epsilon$	X	$N \rightarrow \epsilon$
F	X	X	$F \rightarrow (E)$	X	$F \rightarrow a$	X

	First ₁	Follow ₁
E	a, ($\epsilon,)$
T	a, ($\epsilon, +,)$
M	$\epsilon, +$	$\epsilon,)$
N	$\epsilon, *$	$\epsilon, +,)$
F	a, ($\epsilon, +, *,)$

- S'en servir pour analyser les chaînes $a^*a (a+a)^*a (a^*a) + a$

Gestion des erreurs

➤ Façons de gérer les erreurs

- S'arrêter à la 1ère erreur (problème: n erreurs \Rightarrow n+1 compilations)
- Signaler l'erreur, opérer une correction syntaxique mais arrêter la génération de code
- Signaler l'erreur, opérer une correction syntaxique et continuer à générer du code

➤ Exemple de gestion d'erreur en s'arrêtant sur la 1ère erreur

	+	*	()	a	\$
E	J'attends (ou a	J'attends (ou a	E \rightarrow TM	J'attends (ou a	E \rightarrow TM	Fin de fichier inattendu
T	J'attends (ou a	J'attends (ou a	T \rightarrow FN	J'attends (ou a	T \rightarrow FN	Fin de fichier inattendu
M	M \rightarrow +TM	J'attends + ou) ou fin	J'attends + ou) ou fin	M \rightarrow ϵ	J'attends + ou) ou fin	M \rightarrow ϵ
N	N \rightarrow ϵ	N \rightarrow *FN	J'attends +, *,) ou fin	N \rightarrow ϵ	J'attends +, *,) ou fin	N \rightarrow ϵ
F	J'attends (ou a	J'attends (ou a	F \rightarrow (E)	J'attends (ou a	F \rightarrow a	Fin de fichier inattendu

Ingénierie Dirigée par les Modèles

Gestion des erreurs/_{suite}

➤ Types d'erreurs

- Erreurs lexicales
 - ❖ Caractère illégal, overflow, nombre incorrect, indicateur trop long
- Erreurs syntaxiques
 - ❖ Parenthèse non fermée, erreur de structure de bloc, instructions composées mal construites, absence de délimiteur ou séparateur
- Erreurs sémantiques
 - ❖ Identificateur non déclaré, mauvaise orthographe, déclarations multiples, incompatibilité de types, inconsistance entre paramètres réels et formels
- Erreurs à l'exécution
 - ❖ Division par zéro, racine carrée d'un négatif, débordement de tableau, débordement de mot mémoire
- Erreurs dues aux limitations du compilateur
 - ❖ Limite de dimension de la taille des symboles, limite de capacité de la pile, limite du nombre de blocs, limite du niveau d'emboîtement

➤ Diagnostic d'erreur

- Messages expressifs en terme d'utilisateur et non de machine, spécifiques, localisateurs, complets, lisibles, polis

Ingénierie Dirigée par les Modèles

Limites de l'analyse descendante

- Les grammaires LL(1) sont toutes LR(1), l'inverse est faux
- Ainsi la plupart des langages de programmation ont des grammaires LR(k)
 - L'outil Yacc (Yet Another Compiler Compiler, ou bison) utilisé pour créer les compilateurs de langages de programmation est LALR (Look-Ahead LR), donc LR(k)
- L'outil utilisé avec les technologies de l'IDM, ANTLR (ANother Tool for Language Recognition) est LL(*)
 - Il peut **en théorie** reconnaître tout langage LR(k) au prix de recherche du bon arbre syntaxique avec backtracking
 - En pratique, on utilise PEG (Parsing Expression Grammars) – Bryan Ford 2004 qui, au prix de plus de mémoire, est capable de faire l'analyse syntaxique en temps linéaire et pas exponential pour du LL(*), tout en **supportant la récursivité à gauche**
 - Cependant, les erreurs de compilation sont différentes de l'habitude avec des compilateurs LR(k)
 - ❖ Il faut regarder et corriger la dernière erreur d'abord

PEG, magie du 21^{ème} siècle ?

- Plutôt que générer des tables, PEG crée des fonctions pour chaque règle de grammaire
- Il existe une unique fonction expect: in out position x terminal -> boolean qui, s'il trouve le terminal à la position, renvoie vrai et avance la position après le terminal, sinon renvoie faux sans toucher à la position
- Pour toute règle, on va créer une fonction qui fonctionne sur le même principe

➤ expr: term '+' term | term

boolean term(in out pos) {

➤ term: ID | INT

return expect(pos,ID) || expect(pos,INT);

boolean expr(in out pos) {

}

 savepos=pos;

Si entrée « toto », la fonction term(0) est appelée deux fois, car la première condition échoue.

 if term(pos) && expect(pos,'+') && term(pos) return true;

On voit qu'on pourrait factoriser term(pos) en mémorisant son résultat dans une sorte de cache, Ainsi term(0) est calculé et mémorisé la première fois dans une table. La seconde fois on lit la table.

 pos=savepos;

Cela s'appelle **packrat parsing** et permet une analyse en temps linéaire.

 if term(pos) return true;

 pos=savepos;

 return false;

}

Ajout d'actions et de noms

➤ expr: x=term '+' y=term {self=x+y;} | term

– Règle par défaut, comme dans la seconde alternative, est self=valeur de la seconde alternative

➤ term: ID | INT

– L'idée ici est de typer les terminaux tels que ID et INT

boolean expr(in out pos) {

 savepos=pos;

 if (int x=term(pos)) && expect(pos,'+') && (int y=term(pos)) return x+y;

 pos=savepos;

 if (int term=term(pos)) return term;

 pos=savepos;

 return null;

}

Cela ressemble de moins en moins à du C,
mais cela donne l'idée générale

Règles récursives à gauche

➤ PEG supporte la récursivité à gauche

➤ L'idée est de choisir au début un borne supérieure du nombre de récursions

➤ Tant qu'on n'arrive pas à trouver mais que en augmentant le nombre de récursions on arrive plus loin, on continue.

➤ Si on n'arrive pas plus loin alors on arrête la récursion.

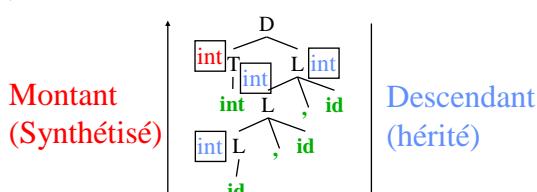
ANTLR

➤ ANTLR utilise PEG et possède donc la puissance LL(*), en autorisant la récursivité à gauche, et la décoration de l'arbre par attributs (associer une valeur à chaque nœud de l'arbre syntaxique) synthétisés, puisque c'est au retour des fonctions appelées pour le sous arbre que l'on calcule les attributs d'un nœud, mais que les attributs précédents ont été calculés. Il supporte l'Extended BNF (EBNF), c-à-d X+, X?, X*,(XY)

➤ Exemple: grammaire de déclaration de types (int a,b,c)

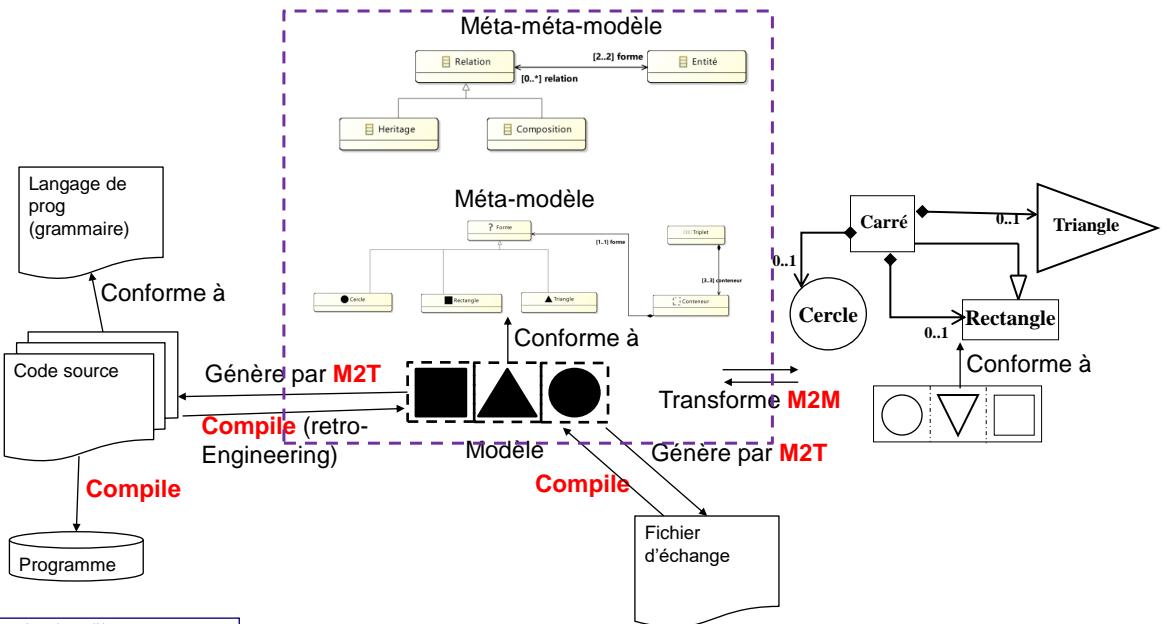
- $D \rightarrow t = T \quad v = L \quad \{v=t\}$
- $T \rightarrow \text{INT} \quad \{t=\text{int}\}$
- $T \rightarrow \text{FLOAT} \quad \{t=\text{float}\}$
- $I1 = L \rightarrow I2 = L \quad , \quad i = \text{ID} \quad \{I2 = I1; i = I1\}$
- $I1 = L \rightarrow i = \text{id} \quad \{i = I1\}$

➤ Exemple: int a,b,c



MOF standard OMG ou pragmatique EMF ?
Modèles, Méta-modèles, Méta-méta-modèles

Monde de l'IDM



53

Avantages apportés par l'IDM

- Génération de code
- API réflexive
 - On peut accéder et parcourir les informations des classes, comme les attributs, et les méthodes
 - Permet sérialisation automatique
 - Persistance des entités
 - Migration d'entités d'une version à une autre
- Généralement interfaces de manipulation fournies
 - Arbres d'entités, formulaires pour édition d'entité, tables d'entités

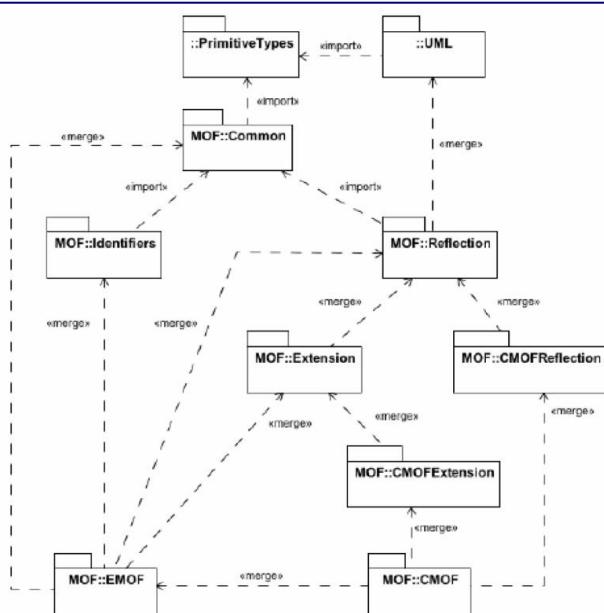
Réflexion en informatique

- Capacité d'un programme, durant son exécution, d'examiner (introspection) et éventuellement modifier (intercession) sa structure interne
 - En programmation objet, typiquement pouvoir parcourir non seulement les instances, mais aussi les classes, leurs méthodes, et propriétés
 - Imaginer comment un formulaire peut être généré automatiquement et dynamiquement
 - Cela explique pourquoi le *design pattern* (voir cours M. Richard) Factory est souvent requis
 - ❖ Création de tout objet via une factory qui va retenir une référence à tout objet créé
 - ❖ La factory permet donc de parcourir toute entité créée
 - ❖ La présenter simplement sous forme d'arbre
 - ❖ Elle peut donc automatiquement gérer la sérialisation
 - ❖ Elle peut aussi automatiquement fournir undo/redo
 - ❖ Elle peut être utilisée pour parcourir tout le modèle en vue de le représenter graphiquement aussi

Ingénierie Dirigée par les Modèles

55

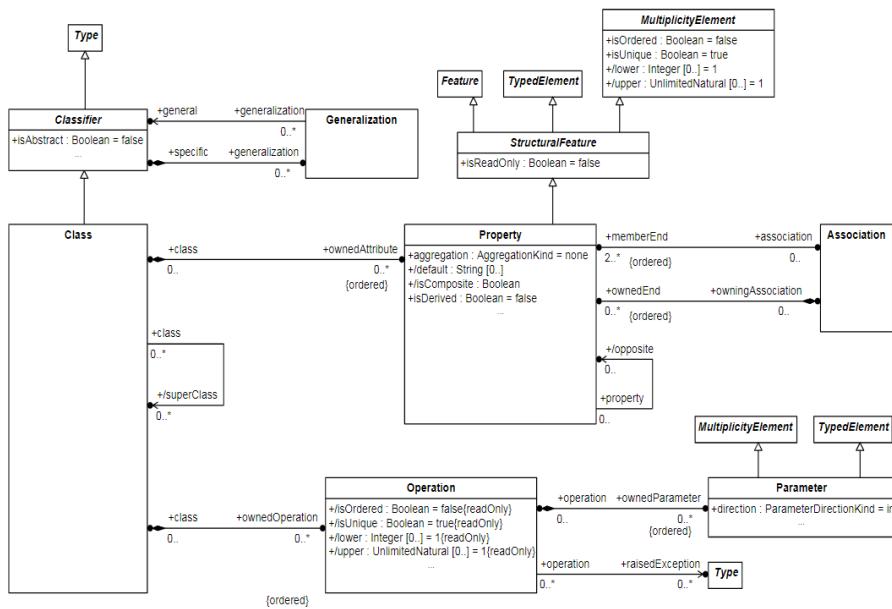
MOF : organisation en paquetages



Ingénierie Dirigée par les Modèles

56

Essential MOF : EMOF - classes

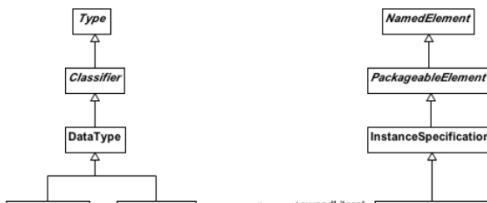


Ingénierie Dirigée par les Modèles

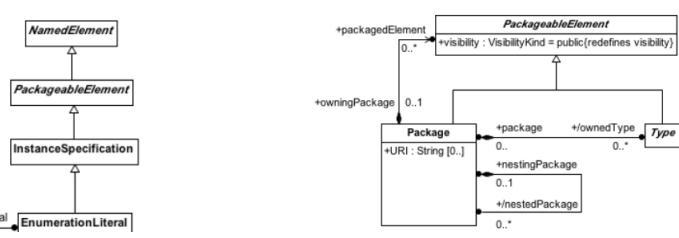
57

EMOF -

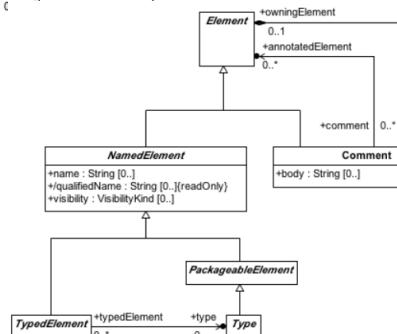
➤ Data Types



➤ Packages



➤ Types



Ingénierie Dirigée par les Modèles

58

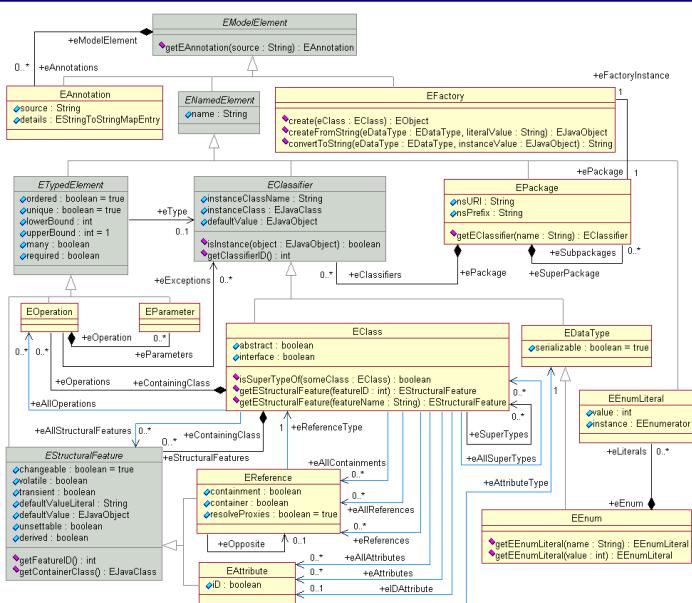
MOF vs. EMF

- Eclipse Modeling Framework est utilisé dans de nombreux logiciels tels que, pour ne citer que ceux que j'ai utilisés
 - Eclipse, YAKINDU, ReqIF Studio, Acceleo, Capella, CodeWarrior, Obeo Designer, OMNeT++, PyDev, Sirius
- En fait, il en existe des centaines
- Outils basés sur la norme Meta Object Facility
 - ?

Ingénierie Dirigée par les Modèles

59

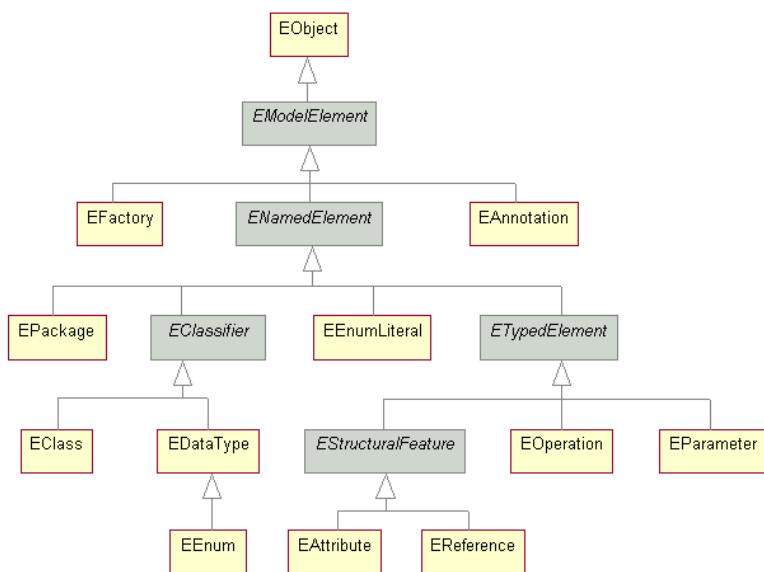
Ecore meta-meta-modèle



Ingénierie Dirigée par les Modèles

60

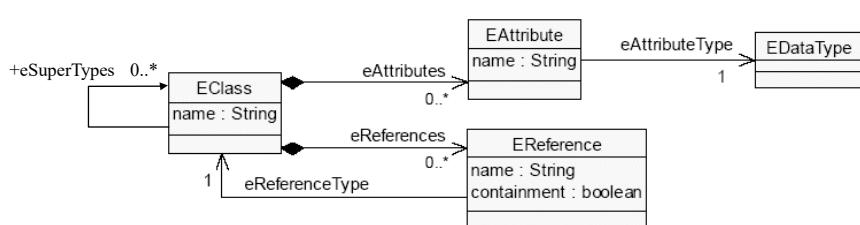
Relations d'héritage des entités ECore



Ingénierie Dirigée par les Modèles

61

Eléments fondamentaux d'ECore



➤ Eléments classiques de construction

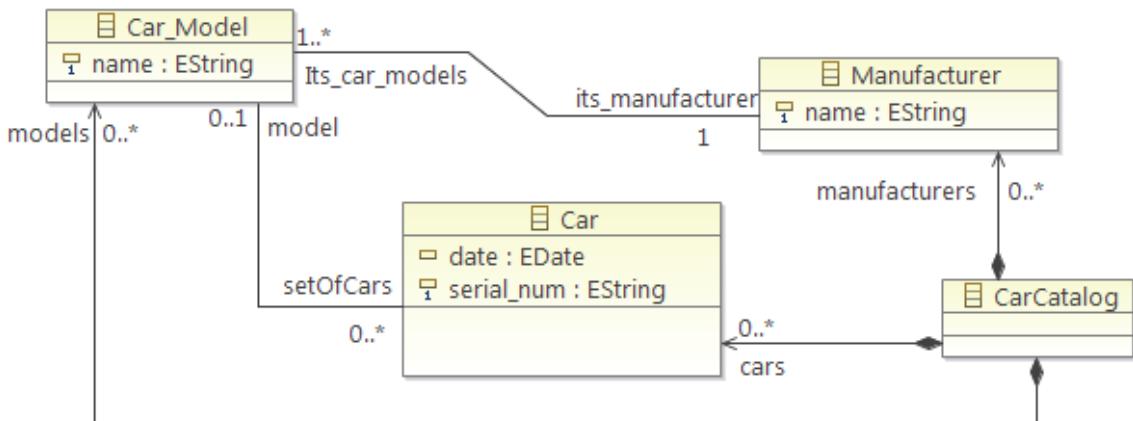
- Classes
- Attributs typés
- Associations (propriétés)
- Généralisation

Ingénierie Dirigée par les Modèles

62

Fil conducteur : parc de voiture

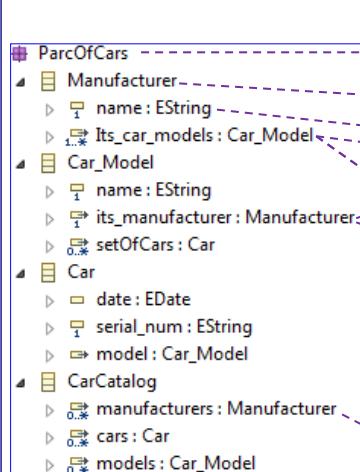
➤ Diagramme de classes



Ingénierie Dirigée par les Modèles

63

Modèle de base et version textuelle



```

import ecore : 'http://www.eclipse.org/emf/2002/Ecore#';

package ParcOfCars : parc = 'www.ensma.fr/parc'
{
    class Manufacturer
    {
        attribute name : String { id };
        property Its_car_models#its_manufacturer : Car_Model[+];
    }

    class Car_Model
    {
        attribute name : String { id };
        property its_manufacturer#Its_car_models : Manufacturer;
        property setOfCars#model : Car[*];
    }

    class Car
    {
        attribute date : ecore::EDate[?];
        attribute serial_num : String;
        property model#setOfCars : Car_Model[?];
    }

    class CarCatalog
    {
        property manufacturers : Manufacturer[*] { composes };
        property cars : Car[*] { composes };
        property models : Car_Model[*] { composes };
    }
}
    
```

Ingénierie Dirigée par les Modèles

64

Au format d'échange XMI (XML Metadata Interchange)

```
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:.ecore="http://www.eclipse.org/emf/2002/Ecore" name="ParcOfCars" nsURI="http://www.ensma.fr/parc" nsPrefix="parc">
  <eClassifiers xsi:type="ecore:EClass" name="Manufacturer">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" ordered="false" lowerBound="1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString" id="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="Its_car_models" ordered="false"
      lowerBound="1" upperBound="-1" eType="#//Car_Model" eOpposite="#//Car_Model/its_manufacturer"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Car_Model">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" ordered="false" lowerBound="1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString" id="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="its_manufacturer" ordered="false"
      lowerBound="1" eType="#//Manufacturer" eOpposite="#//Manufacturer/Its_car_models"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="setOfCars" ordered="false"
      upperBound="-1" eType="#//Car" eOpposite="#//Car/model"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Car">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="date" ordered="false" eType="ecore:EDataType http://www.eclipse
      lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="model" ordered="false"
      eType="#//Car_Model" eOpposite="#//Car_Model/setOfCars"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="CarCatalog">
    <eStructuralFeatures xsi:type="ecore:EReference" name="manufacturers" ordered="false"
      upperBound="-1" eType="#//Manufacturer" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="cars" ordered="false" upperBound="-1"
      eType="#//Car" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="models" ordered="false"
      upperBound="-1" eType="#//Car_Model" containment="true"/>
  </eClassifiers>
</ecore:EPackage>
```

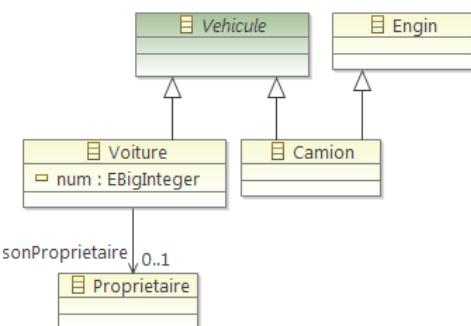
Ingénierie Dirigée par les Modèles

65

EClass : les classes

➤ Les instances de EClass

- désignent les classes des modèles dont la classe racine ;
- identifiées par un nom ;
- peuvent contenir des attributs et/ou des références ;
- supportent l'héritage multiple ;
- peuvent être abstraites (pas d'instanciation possible)



```

abstract class Vehicule;
class Voiture extends Vehicule
{
    property sonProprietaire : Proprietaire[?] ;
    attribute num : Integer[?];
}
class Camion extends Vehicule, Engin;
class Engin;
class Proprietaire;
```

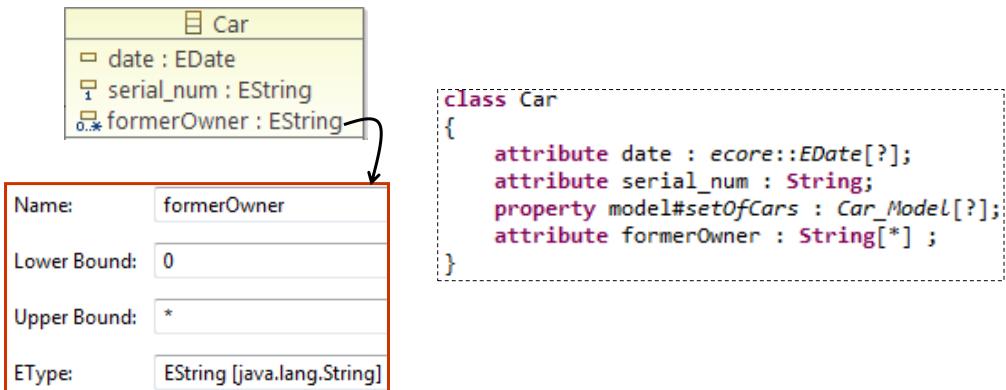
Ingénierie Dirigée par les Modèles

66

EAttribute : les attributs de classe

➤ Les instances de EAttribute :

- identifiées par un nom et un type;
- possèdent des bornes mini et maxi pour la cardinalité
- Le type est en général un type de base, peut aussi être une classe Java



Ingénierie Dirigée par les Modèles

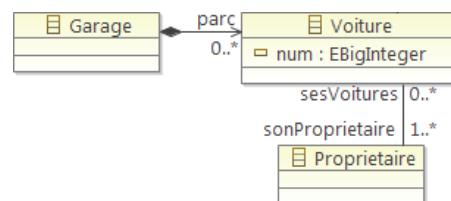
67

EReference : les relations entre classes

➤ Les instances de EReference :

- sont des associations entre deux classes ;
- identifiées par un **nom** (**leur rôle**) et la classe de destination ;
- relations **inverses** possibles (opposite / #) ;
- possèdent des bornes mini et maxi pour la **cardinalité** ;
- peuvent être de type **composition** (containment / composes).
- représentent le sens de la **navigabilité**

```
class Voiture extends Vehicule
{
    property sonProprietaire#sesVoitures : Proprietaire[+];
    attribute num : Integer[?];
}
class Proprietaire
{
    property sesVoitures#sonProprietaire : Voiture[*];
}
class Garage
{
    property parc : Voiture[*] { composes };
}
```



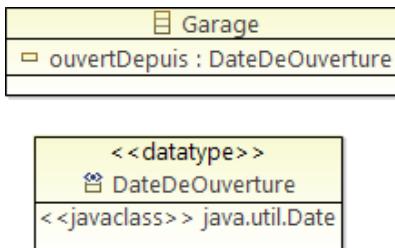
Ingénierie Dirigée par les Modèles

68

EDataType les types de base ou non modélisés (classes Java externes)

➤ Les instances de EDataType :

- modélisent des types simples dont la structure n'est pas modélisée;
- sont de type primitif ou type objet défini par une classe Java;
- encapsulation des types simples;
- utilisées pour typer les attributs (EAttribute)



```

class Garage
{
    property parc : Voiture[*] { composes };
    attribute ouvertDepuis : DateDeOuverture[?];
}
datatype DateDeOuverture : 'java.util.Date';
  
```

Ingénierie Dirigée par les Modèles

EPackage l'unité organisationnelle

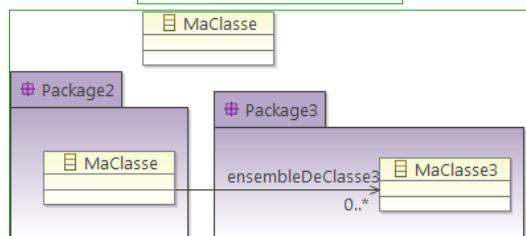
➤ Les instances de EPackage :

- désignent les *packages* des modèles;
- sont des conteneurs de *classifiers* et de sous-packages;
- sont définies par un nom de package (unique) et une URI (Uniform Resource Identifier) pour l'identification lors de la sérialisation.

```

package Package1 : p1 = 'www.ensma.fr/package1'
{
    package Package2 : p2 = 'www.ensma.fr/package1/package2'
    {
        class MaClasse
        {
            property ensembleDeClasse3 : Package1::Package3::MaClasse3[*];
        }
    }
    package Package3 : p3 = 'www.ensma.fr/package1/package3'
    {
        class MaClasse3;
    }
    class MaClasse;
}
  
```

Name:	Package1
Ns Prefix:	p1
Ns URI:	www.ensma.fr/package1

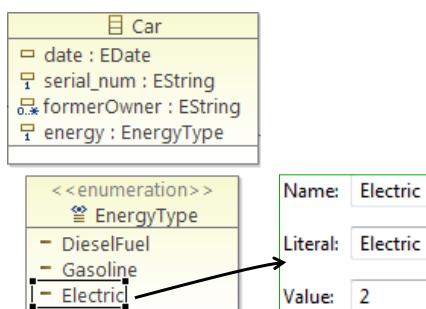


Ingénierie Dirigée par les Modèles

EEnum et EEnumLiteral les énumérations

➤ Les instances de EEnum :

- désignent les types énumérés ;
- Se composent d'un ensemble d'instances de EEnumLiteral ;
- Chaque instance de type EEnumLiteral :
 - est identifiée par un nom ;
 - est associée à un entier et une valeur (label)



```

class Car
{
    attribute date : ecore::EDate[?];
    attribute serial_num : String;
    property model#setOfCars : Car_Model[?];
    attribute formerOwner : String[*];
    attribute energy : EnergyType ;
}

enum EnergyType
{
    literal DieselFuel;
    literal Gasoline = 1;
    literal Electric = 2;
}
    
```

71

EOperation les méthodes

➤ Les instances de EOperation :

- désignent les opérations d'une classe pouvant être invoquées ;
- sont identifiées par un nom et des paramètres représentant la signature ;
- sont caractérisées par un type de retour (qui peut être null)



```

class Voiture extends Vehicule
{
    operation VerifierVidange(kilometrage : Integer, ancienKilometrage : Integer ) : Boolean;

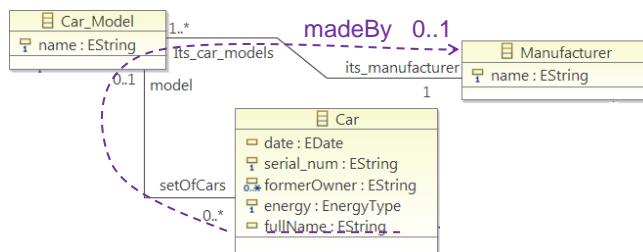
    property sonProprietaire#sesVoitures : Proprietaire[+];
    attribute num : Integer;
}
    
```

Ingénierie Dirigée par les Modèles

72

Attributs et références dérivés

- Les instances de EStructuralFeature (EAttribute et EReference) sont caractérisées par un ensemble de propriétés, et notamment « derived » :
 - pour un attribut ou une référence calculé à partir d'autres attributs ;
 - Par défaut les EStructuralFeature sont non dérivés !derived



```

class Car
{
    attribute date : ecore::EDate[?];
    attribute serial_num : String;
    property model#setOfCars : Car_Model[?];
    attribute formerOwner : String[*];
    attribute energy : EnergyType;
    property madeBy : Manufacturer[?] { derived }
    {
        derivation: self.model.its_manufacturer;
    }
    attribute fullName : String[?] { derived }
    {
        derivation : model.name.concat(serial_num);
    }
}
    
```

Ingénierie Dirigée par les Modèles

73

Attributs et références changeables ou non

- Les instances de EStructuralFeature (EAttribute et EReference) sont caractérisées par un ensemble de propriétés, et notamment « changeable »
 - si l'instance est « non changeable », sa valeur ne peut changer : cela peut être pratique pour les relations inverses ;
 - Par défaut les features sont changeable ;
 - mot clé : **readonly** ;

```

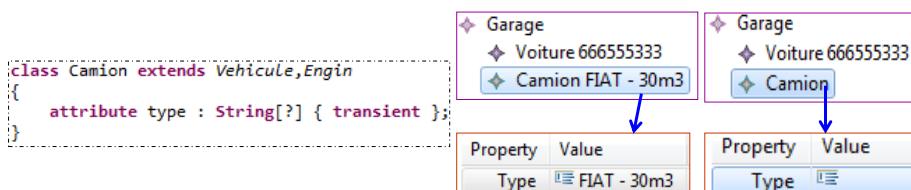
class Manufacturer
{
    attribute name : String { id };
    property Its_car_models#its_manufacturer : Car_Model[+] { readonly };
}
class Car_Model
{
    attribute name : String { id };
    property its_manufacturer#Its_car_models : Manufacturer;
    property setOfCars#model : Car[*];
}
    
```

Ingénierie Dirigée par les Modèles

74

Persistante des attributs et références

- Les instances de EStructuralFeature (EAttribute et EReference) sont caractérisées par un ensemble de propriétés, et notamment « transient »
 - une *feature* caractérisée par « transient » ne sera pas persistante ;
 - pour déclarer des données qui ne seront pas invoquées ;
 - par défaut les *features* ne sont pas « transient »
 - Dans la pratique, les *features* de type « derived » sont « transient »

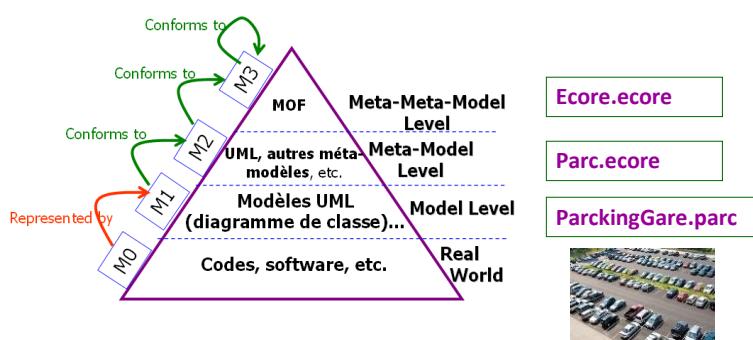


Ingénierie Dirigée par les Modèles

75

Architecture d'un modèle ECore

- Un modèle Ecore
 - Est un arbre hiérarchique contenant une **classe racine**
 - ❖ classe racine = système à modéliser
 - Il se compose d'un ensemble de :
 - ❖ classes, attributs, relations, opérations
 - Un modèle conforme directement à Ecore est appelé un métamodèle
 - ❖ selon la pyramide de l'OMG



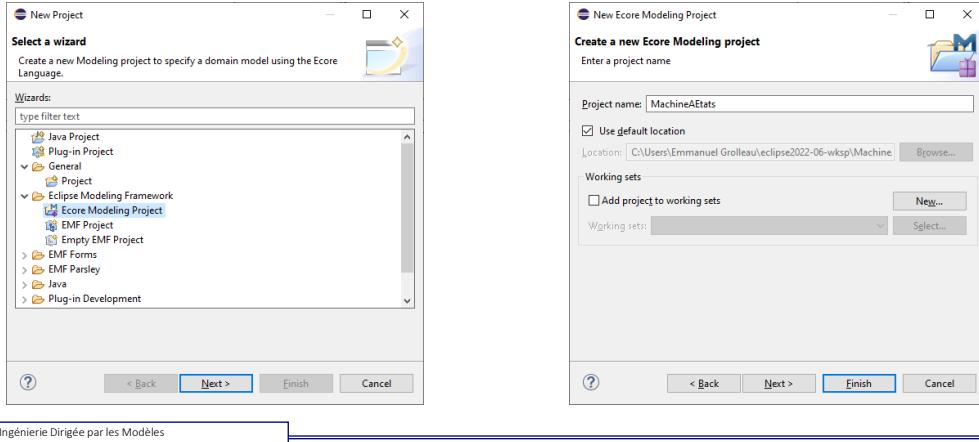
Ingénierie Dirigée par les Modèles

76

Exercice : méta-modèle et modèle de machine à états

- Une machine à états possède un nom, elle se compose d'un ensemble d'états et de transitions entre états.
- Un état est initial ou non, possède un nom, et se caractérise par des transitions sortantes.
- Une transition est conditionnée par un événement, et possède un état source et un état destination
- Un événement est une chaîne de caractères

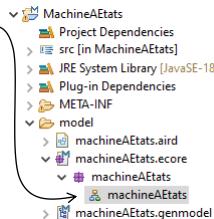
➤ Dans Eclipse, créer un projet de type Eclipse Modeling Framework -> Ecore Modeling Project



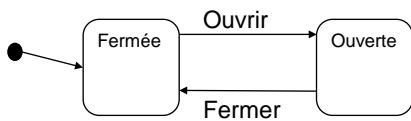
77

Exercice /suite

- Ouvrir la vue diagramme de classe



- Proposer un méta-modèle de machine à états
- Afin de le tester sur une instance, clic droit sur la classe racine, et « Create Dynamic Instance »
- Ouvrir cette instance dynamique, et créer une instance où on modélise une porte, dont la machine à état est donnée ci-dessous
 - Pour le moment, nous devons nous restreindre à utiliser la représentation arborescente du modèle



78

OCL Object Constraint Language

Sémantique statique des modèles

Ingénierie Dirigée par les Modèles

Contraintes sur les modèles

- Graphiquement on ne définit qu'un aspect partiel du système
- Exemple
 - « Il y a un et un seul état initial dans une machine à états »
 - « Un état doit posséder au moins une transition entrante »
- On peut l'exprimer sous forme de commentaires, mais dans ce cas pas de vérification automatique possible
- Nécessité d'un langage formel (i.e., avec une sémantique bien définie) permettant d'exprimer les contraintes que nous n'avons pas pu exprimer dans le modèle
 - Par exemple, « dans une machine à états, une transition possède exactement un état source et un état destination » s'exprime grâce à la cardinalité

Ingénierie Dirigée par les Modèles

Le langage OCL (Object Constraint Language)

➤ Langage de requête :

- qui permet de calculer une expression sur un modèle en s'appuyant sur son méta-modèle.
- expression exprimée une fois au niveau modèle peut être évaluée sur toutes les instances de ce modèle
- Exemple d'expression : le nombre d'états initiaux est-il exactement 1 ?
- Une expression OCL calcule une valeur mais laisse le modèle inchangé !

➤ Langage formel :

- langage de spécification (pas de programmation)
- proposé par l'OMG : <http://www.omg.org/spec/OCL>
- supporté par des outils;
- s'applique entre autres sur les méta-modèles basés MOF ou Ecore

Le langage OCL (Object Constraint Language) /suite

➤ Principe :

- Définir formellement des expressions à évaluer sur une classe et ses opérations
- Utilisation des invariants, des pré-conditions et des post-conditions

- Invariant : expression définie sur une classe qui doit toujours être vraie
- Pré-condition : expression qui doit être vérifiée pour que l'exécution d'une opération soit possible
- Post-condition : propriété caractérisant le résultat obtenu après exécution d'une opération

➤ Utilisation étendue d'OCL

- Calcul d'attributs dérivés
- Règles de transformation de modèles
- Règles de génération de code
- Règles de construction de DSL
- ...

Laboratoire d'Informatique et d'Automatique pour les Systèmes - École Nationale Supérieure de Mécanique et d'Aérotechnique

I- 83

Installation dans Eclipse

➤ Help -> Install new software...

Available Software

Check the items that you wish to install.

Work with: 2022-09 - <https://download.eclipse.org/releases/2022-09>

Selected Software

Name	Version
General Purpose Tools	
OCL Classic SDK: Ecore/UML Parsers,Evaluator,Edit,Developer Resou	5.17.1.v20220309-0840
Modeling	
OCL Build Support	6.17.1.v20220309-0840
OCL Build Support Developer Resources	6.17.1.v20220309-0840
OCL Classic SDK: Ecore/UML Parsers,Evaluator,Edit,Developer Resou	5.17.1.v20220309-0840
OCL Examples and Editors SDK	6.17.1.v20220309-0840
OCL Examples and Editors SDK Developer Resources	6.17.1.v20220309-0840

2 items selected

Details

Show only the latest versions of available software

Group items by category

Show only software applicable to target environment

Contact all update sites during install to find required software

Hide items that are already installed

What is [already installed?](#)

Do you trust unsigned content of unknown origin?

Type	Id/Fingerprint	Name	Validity Dates
Unsigned	n/a	Unknown	n/a

Always trust all content

Select All Deselect All

Classifier Id Version

Classifier	Id	Version
osgi.bundle	ipg.runtime.java.source	2.0.17.v201004271640

Trust Selected Cancel

Ingénierie Dirigée par les Modèles

83

Laboratoire d'Informatique et d'Automatique pour les Systèmes - École Nationale Supérieure de Mécanique et d'Aérotechnique

I- 84

Édition de contraintes OCL sur un métamodèle

SM

Project Dependencies

JRE System Library [jdk-19]

Plug-in Dependencies

META-INF

MANIFEST.MF

model

MachineAEtats.xmi

sMard

sM.ecore

sM.genr

New

Show In Alt+Shift+W F3

Open Open With

- Ecore Editor
- Generic Text Editor
- Sample Ecore Model Editor
- Sample Reflective Ecore Model Editor
- Text Editor
- Trace file viewer
- System Editor
- In-Place Editor
- Default Editor
- Other...

General Information

This section describes general information a

ID: SM

Version: 0.1.0.qualifier

Name: SM

Vendor:

Platform filter:

Activator:

Activate this plug-in when one of its clas

This plug-in is a singleton

```

1 package sM : sM = 'http://www.example.org/sM'
2 {
3 class MachineAEtats
4 {
5   attribute Nom : String[?];
6   property etat : Etat[*|1] { ordered composes };
7   property transition : Transition[*|1] { ordered composes };
8   invariant UniqueInitial : etat.Initial->count(true)=1;
9 }
10 class Etat
11 {
12   attribute Nom : String[?];
13   attribute Initial : Boolean[1];
14   property outTransition : Transition[*|1] { ordered };
15 }
16 class Transition
17 {
18   property source : Etat[1];
19   property dest : Etat[1];
20   attribute label : String[?];
21 }
22 }
23 
```

Problems encountered during validation

Reason: Diagnosis of Machine AEtats

The 'UniqueInitial' constraint is violated on 'Machine AEtats'

OK << Details

Ingénierie Dirigée par les Modèles

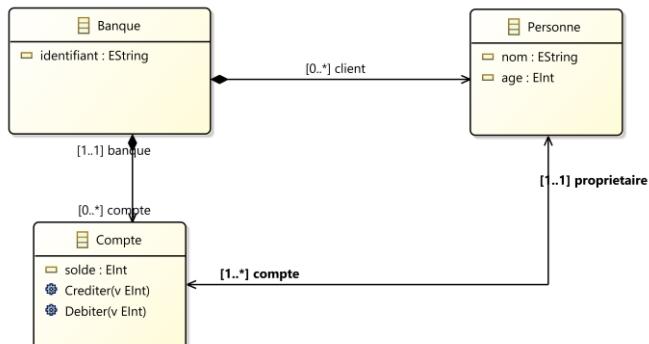
84

Préparation de contraintes OCL

- Afin de préparer au mieux les contraintes OCL, il convient d'utiliser la console OCL sur une instance du métamodèle
- Créer d'abord une instance dynamique (create dynamic instance)
 - Click droit sur la classe racine -> Show OCL console
- La sélection d'une entité fait que le contexte de règle OCL tapée dans la console est considéré sur la classe sélectionnée
- Lorsqu'on est satisfait de la contrainte, on va la copier/coller dans le fichier textuel du métamodèle Ecore
- Dans la classe contexte
- Sous forme « Invariant nom_invariant : » suivi de la contrainte OCL testée dans la console, qui doit renvoyer un booléen, terminée par « ; »

OCL et contexte

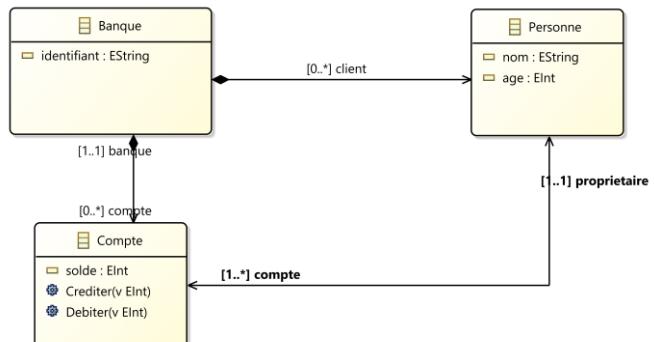
- Une expression OCL est toujours définie dans un contexte qui identifie :
 - La cible : élément du modèle
 - Le rôle : l'expression OCL s'applique à l'élément du modèle définie par le contexte
 - Mot-clé : context
 - Exemple : context Compte



Invariants OCL

➤ Chaque invariant :

- exprime une contrainte, sur une instance ou un groupe d'instances;
- doit être toujours vrai
- Mot-clé : **inv**
- Exemple : pour toutes les instances de la classe Compte, l'attribut solde doit toujours être positif



invariant solde_positif : solde >= 0;

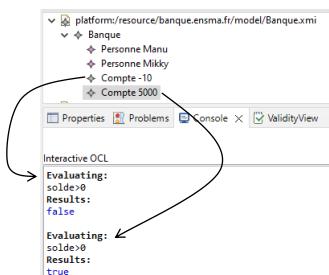
87

Avec la console OCL ou OCLinEcore

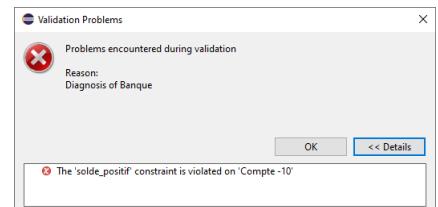
```

import ecore : 'http://www.eclipse.org/emf/2002/Ecore';
package banque : banque = 'http://www.ensma.fr/idm/banque'
{
    class Banque
    {
        attribute identifiant : String[?];
        property client : Personne[*|1] { ordered composes };
        property compte#banque : Compte[*|1] { ordered composes };
    }
    class Personne
    {
        attribute nom : String[?];
        attribute age : ecore::EInt[1];
        property compte#proprietaire : Compte[+|1] { ordered };
    }
    class Compte
    {
        operation Crediter(v : ecore::EInt[1]);
        operation Debiter(v : ecore::EInt[1]);
        property banque#compte : Banque[1];
        property proprietaire#compte : Personne[1];
        attribute solde : ecore::EInt[1];
        invariant solde_positif: solde > 0;
    }
}
  
```

OCLinEcore est un DSL textuel



Lorsqu'on cherche à valider le modèle



88

Types en OCL

- Les types sont soit primitifs:
 - Integer, Real, Boolean, Enumeration, String, UnlimitedNatural, **OclAny**, **OclInvalid**, **OclVoid**, **null**
 - 3, -5, 1.2, 1.5E-3, true, false, 'manu'
 - Munis des opérations usuelles proches d'Ada : +, -, *, /, >, <=, =, <>, and, or, xor, implies, not...
- Ou des tuples
 - Tuple<year:Integer,month:String,day:Integer>
- Soit ce sont des collections, nom générique pour des regroupements
 - Bag où des éléments peuvent être répétés par exemple Bag {1, Bag{2,3}, 2, 2} et où on peut avoir plusieurs niveaux de collections
 - Sequence où l'ordre d'insertion est conservé mais où un élément peut apparaître plusieurs fois
 - Set où un même élément ne peut apparaître qu'une fois Set {2,1}
 - OrderedSet où l'ordre d'insertion est conservé OrderedSet {1,2} si 1 a été inséré avant 2
- Souvent OCL est utilisé soit pour vérifier un invariant, expression OCL renvoyant donc un booléen, soit pour renvoyer un objet du meta-meta-modèle. Cet objet peut donc être n'importe quel élément d'un meta-modèle ou d'un modèle: une classe, un objet, un attribut, une opération, une propriété, etc., ou une collection de ceux-ci

Ingénierie Dirigée par les Modèles

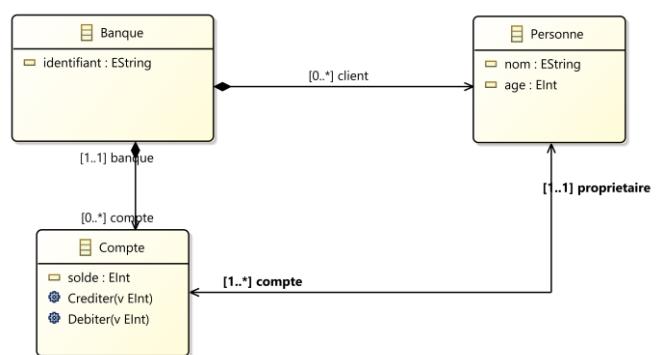
Naviguer dans un modèle à partir d'OCL

- La raison pour laquelle OCL est très utilisé (encore comme base par exemple d'Acceleo Query Language qui le supplante) est qu'il permet de naviguer dans un modèle à partir d'une instance dite « contexte ». L'idée est que l'expression sera évaluée sur une instance du modèle ou MM.
- Ici, étant donnée une instance, extrayons la collection des clients mineurs de la banque.
- On doit commencer par instancier le modèle, sur cette instance nous pourrons tester des contraintes OCL
 - Création d'une instance dynamique de banque

platform/resource/fr.ensma.idm.banque/model/Banque.xmi

- ❖ Banque Crédit Yodais
 - ❖ Personne Yoda
 - ❖ Personne Grogu
 - ❖ Personne Mélody
 - ❖ Compte 200
 - ❖ Compte 20000
 - ❖ Compte 5
 - ❖ Compte 30

	Propriétaire	Age	Banque
❖ 200	Yoda	899	Crédit Yodais
❖ 20000	Yoda	899	Crédit Yodais
❖ 5	Grogu	50	Crédit Yodais
❖ 30	Mélody	4	Crédit Yodais

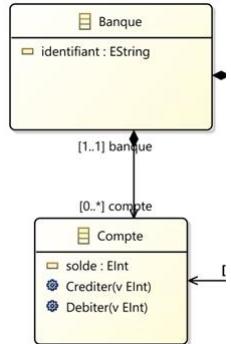


Ingénierie Dirigée par les Modèles

Deux opérateurs de navigation, le « . » et la « -> »

➤ Deux séparateurs possibles dans une expression OCL

- Le « . » permet d'accéder à un attribut, ou changer d'objet(s) via une relation, ou d'appeler une fonction sur chaque élément d'une collection.
- Ex: contexte Banque Crédit Yodais, self.identifiant est un Estring
 - ❖ self.compte correspond à un OrderedSet des 4 comptes existants (car la relation compte est ordered)
 - ❖ self.compte.banque est un OrderedSet de 4 fois le crédit Yodais
 - ❖ self.compte.banque.compte est un Bag contenant en tout 16 comptes (4x4)
- La « -> » permet d'appeler une fonction OCL sur la collection.
 - ❖ self.compte.banque.compte->size() renvoie 16
 - ❖ self.compte.banque.compte->asOrderedSet() renvoie les 4 comptes
 - ❖ self.compte.banque.compte->asOrderedSet()->size() renvoie 4
 - ❖ self.compte.proprietaire.nom->size() renvoie 4, mais .size() renvoie la Séquence{4,4,5,6}
- a.b
 - ❖ Si a est un type de base
 - ✓ si b une relation d'arité 1 renvoie un résultat du type d'objet pointé par b.
 - ✓ Si arité multiple, renvoie un OrderedSet
 - ❖ Si a est une collection, renvoie une séquence si arité 1, un bag si arité multiple
 - ✓ Reste Séquence si en fait l'interpréteur est AQL



Ingénierie Dirigée par les Modèles

Navigation OCL sur une instance dynamique

- On peut simplement tester des expressions OCL
- Par exemple en ouvrant l'interpréteur Sirius ou Acceleo (point de vue Acceleo)
 - En sélectionnant un élément de modèle ou d'instance qui sera utilisé comme contexte
- En sélectionnant sur l'instance « Personne Yoda », et en tapant ocl:self, j'obtiens naturellement Personne Yoda
- En sélectionnant sur le MM Personne, j'obtiens Personne
- Qu'obtiens-je si je tape avec Personne Yoda comme contexte
 - ocl:self.compte
 - ocl:self.banque
 - ❖ Nous n'avons pas de liaison retour Personne -> Banque
 - ocl:self.compte.banque.compte
 - ocl:self.compte.banque.compte->asOrderedSet()
 - ocl:self.compte.banque.compte->asOrderedSet()->select(c|c.proprietaire=self)
 - ocl:self.compte.banque.compte->select(c|c.proprietaire=self).solde->sum()
 - ❖ Attention cette fois nous retrouvons une sequence et non un OrderedSet)
 - Comment vérifier que tous les soldes d'une banque (à partir du contexte banque) sont positifs ?

Ingénierie Dirigée par les Modèles

Quelques opérations OCL

- Sur les chaînes de caractères, s est une chaîne, i un index
 - size(), concat(s), substring, toUpper, toLower, toInteger, toReal
- Sur les collections, o est un objet, c une collection, e une expression qui peut être une lambda expression
 - Une lambda expression est de la forme $c|c.val > 3$
 - size(), includes(o), excludes(o), count(e), includesAll(c) (inclue la collection donnée?), excludesAll(c), including(o), excluding(o), isEmpty(), notEmpty(), isUnique(e), forAll(e), sortedBy(e), iterate(e), sum(), union(c), intersection(c), symmetricDifference(c) (union-intersection), reject(e), collect(e), collectNested(e), flatten(), asSequence(), asBag(), asSet(), any(e), one(e)
 - Si la collection est ordonnée (OrderedSet ou Sequence)
 - ❖ prepend(o), append(o), insertAt(i,o), subSequence(i1,i2), at(i), first(), last(), indexOf(o), iterate(e), sortedBy(e)
- Fonctions spéciales
 - oclIsTypeOf(t), oclIsKindOf(t), oclAsType(t)
 - oclIsUndefined(), oclIsDefined() utile lorsque la cardinalité est 0..x
 - Classe->allInstances()

Bloc conditionnels

- En AQL on peut utiliser des conditionnelles de type impératif
 - [if expr]
 - Code
 - [else]
 - Code
 - [/if]
- En OCL, le if then else endif ne peut s'utiliser que dans une expression
 - If expr1 then expr2 else expr3 endif
 - ❖ Vrai si et seulement si expr1 et expr2 ou non expr1 et expr3
 - ❖ Le else est obligatoire, vous pouvez le voir comme la structure conditionnelle en LabVIEW qui doit toujours renvoyer une valeur, quelle que soit la valeur de la condition
 - ❖ Si vous ne voulez pas de else, utiliser le implies
 - ✓ expr1 implies expr2

Définition de variable ou opération locale à une expression, ou globale à l'interpréteur

- let nomvar=expr in expr
 - nomvar est défini pour expression
- context MaClasse def: nomvar :type = expr
 - nomvar est défini comme un attribut de MaClasse dans l'interpréteur OCL
- La construction def permet aussi de définir des fonctions
- context MaClasse def: f(param:type): typeretour = expr
 - On pourra alors l'utiliser comme une opération de MaClasse
- context MaClasse inv: self.f(x)->size()>3

Divers

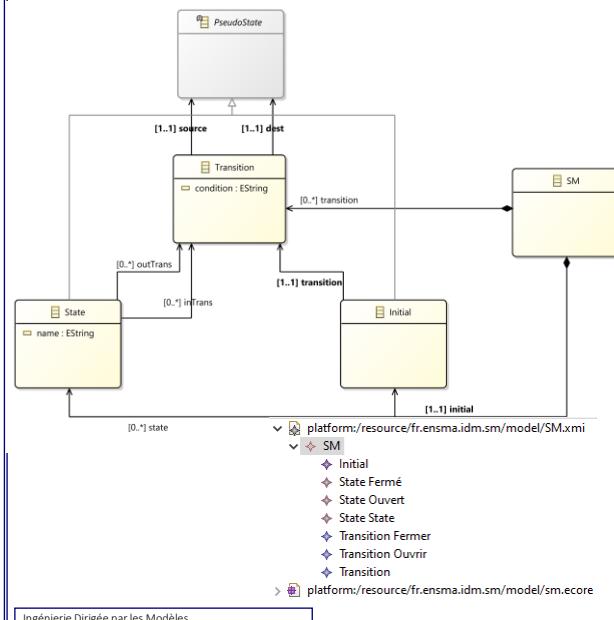
- Si un diagramme d'état est défini pour une classe, on peut utiliser l'état courant avec oclInState(s)
- Les littéraux d'énumérations sont dénotés NomEnum::valeur
- La fonction oclIsNEw() est utilisable dans un post-condition
- Un mémo OCL créé par un collègue, Charles André, à garder sous la main :
 - <http://www-sop.inria.fr/members/Charles.Andre/CAdoc/ESINSA/UMLOCL-memo.pdf>

AQL (Acceleo Query Language)

- AQL est beaucoup plus rapide qu'OCL
- Il fait partie d'Acceleo que nous verrons lors de la génération Motel-to-Text
- La syntaxe peut être à la OCL, en préfixant une expression par aql: au lieu de ocl:
- La syntaxe native Acceleo [expr_aql/] peut aussi être utilisée
 - Elle est mieux supportée par la console de l'interpréteur et le ctrl-espace
- AQL est recommandé à la place d'OCL à partir de Sirius 3.0
- Principales différences AQL vs. OCL
 - L'ordre de toute collection est préservé et toute collection est aplatie
 - ❖ Il n'y a que des Sequence et des (Ordered) Set
 - En OCL, le self peut être implicite, pas en AQL
- Là encore nous avons le choix entre mieux et normalisé
 - OCL est plus restreint, moins rapide mais normalisé par l'OMG
 - AQL est open source mais pas normalisé

Domain Specific Language Création de DSL

Nous avons créé un MM StateMachine, mais comment le rendre utilisable ?



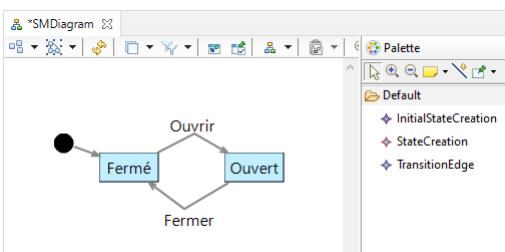
```
<?xml version="1.0" encoding="UTF-8"?>
<sm:SM xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sm="http://www.ensma.fr/idm/sm"
  xsi:schemaLocation="http://www.ensma.fr/idm/sm_sm.ecore">
<initial transition="//@transition.2"/>
<state name="Fermé"
  outTrans="//@transition.1"
  inTrans="//@transition.0 //@transition.2"/>
<state name="Ouvert"
  outTrans="//@transition.0"
  inTrans="//@transition.1"/>
<transition
  condition="Fermer"
  source="//@state.1"
  dest="//@state.0"/>
<transition
  condition="Ouvrir"
  source="//@state.0"
  dest="//@state.1"/>
<transition
  source="//@initial"
  dest="//@state.0"/>
</sm:SM>
```

Pour moi, « professionnel » des StateMachines,
ça ne ressemble en rien à une machine à états

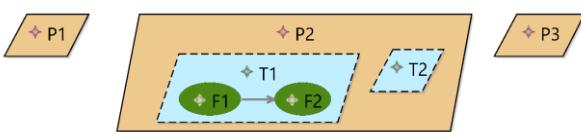
99

Le DSL permet de représenter un modèle en proposant un point de vue métier

➤ C'est mieux non ?

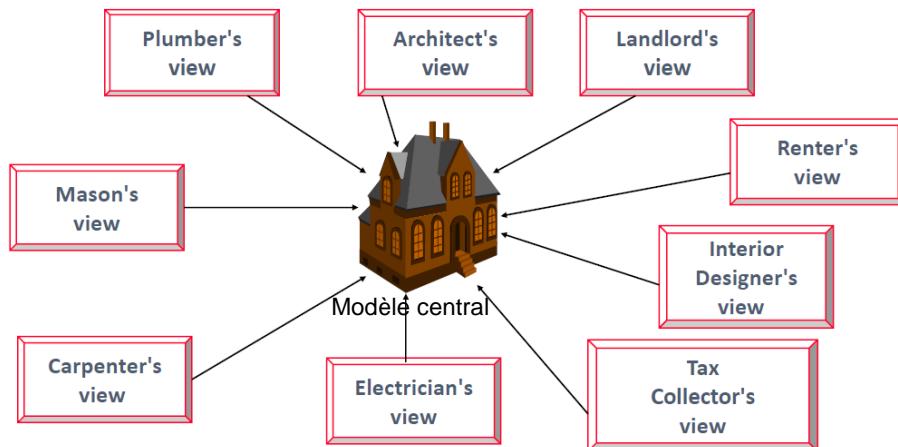


- Et encore la palette pourrait utiliser des labels et icônes plus liés à leur fonction
 - C'est possible à faire, mais en temps limité nous nous contenterons de cela
- En TD/TP nous attaquerons des DSL « à la AADL »



DSL

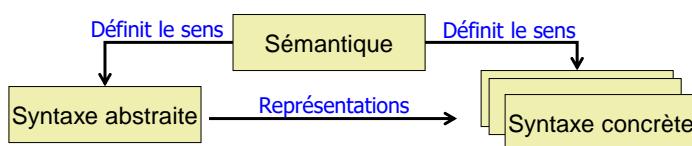
- L'IDM favorise la définition de langages de modélisation dédiés à un domaine particulier



- Chaque vue peut être exprimée en un langage dédié : un DSL (Domain Specific Modeling Language)

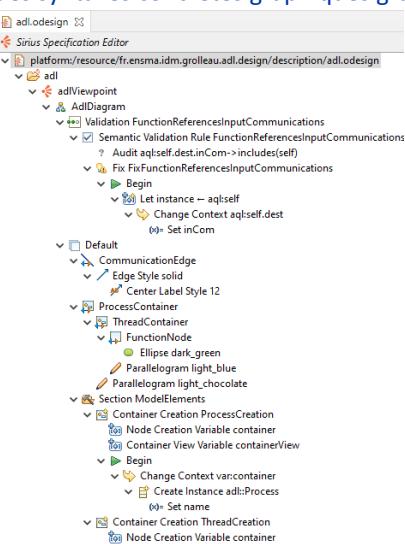
Syntaxe abstraite, concrète et sémantique

- Un DSL est défini par trois concepts
 - Une syntaxe abstraite : la structure du langage
 - ❖ Les éléments et leurs relations
 - ❖ Correspond à ce qui est défini au niveau du métamodèle
 - Une ou plusieurs syntaxes concrètes : les représentations spécifiques du langage de modélisation
 - ❖ Syntaxe graphique et/ou syntaxe textuelle
 - ❖ Plusieurs syntaxes concrètes possibles pour une même syntaxe abstraite
 - Une sémantique : le sens des éléments de la syntaxe abstraite et le sens de leurs représentations (syntaxe concrète)
 - Document de spécification



Outillage

- L'outil xtext permet de définir des syntaxes concrètes textuelles se basant sur ANTLR
- L'outil Obeo Designer permet de définir des syntaxes concrètes graphiques grâce aux Viewpoint specification models
- Nous verrons cela en TD et TP



Ingénierie Dirigée par les Modèles

103

ATL (Atlas Transformation Language) et QVT (Query/View/Transformation) Transformation de modèles Le cas Model-to-Model

Ingénierie Dirigée par les Modèles

104

Intérêts de la transformation de modèles

- La transformation des modèles dans l'IDM :
 - Le passage d'un modèle A à un modèle B
 - ❖ Transformation d'un formalisme à un autre
 - Raffinement d'un modèle
 - Extraction d'une vue d'un modèle
- Il existe deux types
 - Endogène : dans le même métamodèle
 - Exogène : dans un MM différent

■ Définition :

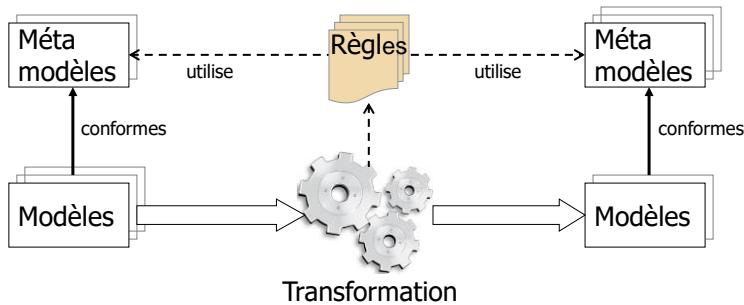
Une transformation de modèle est une fonction $T : \text{inMM} \rightarrow \text{outMM}$
T prend en entrée un ensemble de modèles sources **inM** (conformes aux métamodèles **inMM**) et produit un ensemble de modèles **outM** (conformes aux métamodèles **outMM**) en sortie.

Exemples

- La transformation des modèles sert à :
 - Intégrer des données issues de différents modèles
 - ❖ Création d'un entrepôt de données (data warehouse)
 - Raffiner la modélisation d'un système
 - ❖ Les machines abstraites du langage B
 - ❖ La méthode ARCADIA (Capella) pour les systèmes embarqués
 - Fusionner des modèles différents
 - ❖ Mapping d'une architecture logicielle à une architecture matérielle
 - Gérer l'évolution d'un langage de modélisation (versioning)
 - ❖ Passage de UML 1.4 à UML 2.0
 - Générer le logiciel (ou une partie) à partir de la modélisation
 - ❖ Génération de modèles AADL à partir de modèles LAB de Capella (déploiement)
- etc...

Principes de la transformation M2M

- Un modèle d'entrée est transformé en un modèle de sortie en suivant des règles de transformation spécifiées par un autre modèle. Les transformations sont décrites au niveau métamodèle.
- Les règles sont exécutées sur les modèles sources afin de générer les modèles cibles

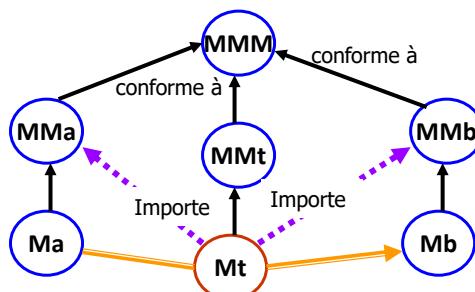


Ingénierie Dirigée par les Modèles

107

Concepts de la transformation M2M

- Fonction de transformation : $T(MMa^*, MMb^*, Mt, Ma^*) \rightarrow Mb^*$
- * : Il pourrait y avoir plusieurs Ma et Mb
- Mt : modèle qui se compose des règles de transformation



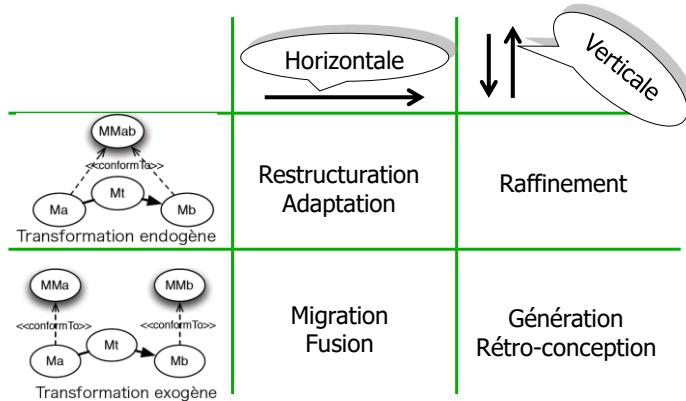
Ingénierie Dirigée par les Modèles

108

Formes de transformation M2M

➤ Une transformation M2M peut être :

- **endogène** ($MM_a = MM_b$) ou **exogène** ($MM_a \neq MM_b$)
- **horizontale** (MM_a et MM_b sont dans le même niveau d'abstraction) ou **verticale** (MM_a et MM_b ne sont pas dans le même niveau d'abstraction)



Règles de transformation

➤ Les règles des transformations M2M :

- définissent la manière dont un ensemble d'éléments du méta-modèle source est transformé en un ensemble d'éléments du méta-modèle cible
- peuvent être exprimées suivant deux paradigmes :
 - ❖ Impérative : des étapes claires qui décrivent comment la règle est exécutée;
 - ❖ Déclarative : informations qui décrivent ce qui doit être créé par la règle.

➤ L'agencement et la structuration des règles exécutées dans une transformation permet de reconstruire l'ensemble du modèle cible.

QVT : la norme OMG

➤ Le standard QVT (Query / Views / Transformations) (Juin/2006):

- proposé par l'OMG afin de normaliser un moyen d'exprimer des correspondances (transformations) entre les langages définis avec MOF.

➤ L'aspect Query :

- Exprime des requêtes pour filtrer et sélectionner des éléments d'un modèle (y compris sélectionner les éléments sources d'une transformation.)

- Basé OCL

➤ L'aspect Views :

- Propose un mécanisme pour créer des vues (Views).
- Vue = modèle déduit d'un autre pour en révéler des aspects spécifiques.

➤ L'aspect Transformations :

- Formalise une manière de décrire des transformations (Transformations).

➤ Peut proposer des règles déclaratives ou impératives

➤ Problème comme souvent avec les standards OMG très ambitieux : outillage manquant

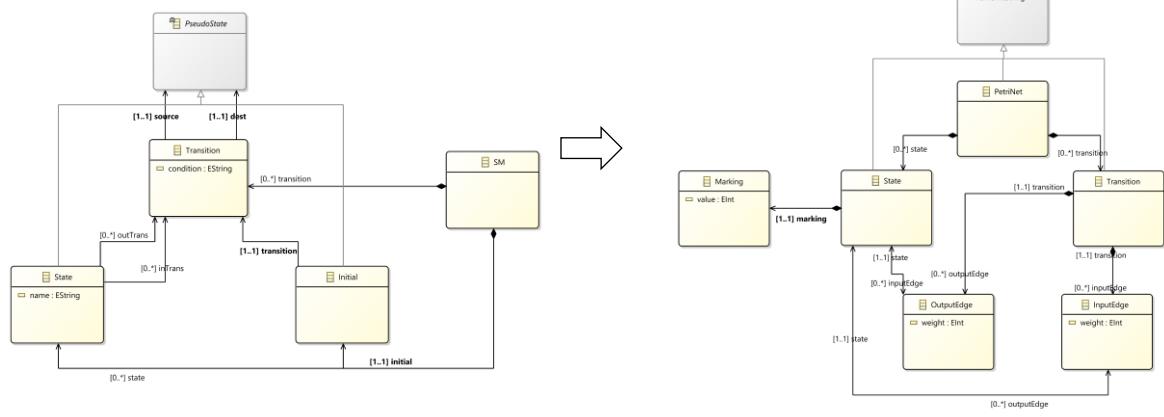
- Il existe une implémentation avec règles impératives : QVTo (pour operational)

Eléments d'une transformation QVTo

```
modeltype.ecore "strict" uses.ecore('http://www.eclipse.org/emf/2002/Ecore');
modeltype.sm "strict" uses.sm('http://www.ensma.fr/idm/sm');
modeltype.pn "strict" uses.pn('http://www.ensma.fr/idm/pn');
```

➤ Nous prenons l'exemple d'une transformation SM to PetriNet basés Ecore

- Le RdP obtenu est nécessairement sauf (1 seul jeton)



QVTo est impératif

➤ Ce n'est pas un problème pour un MM pour DSL car on a un élément racine contenant de façon directe ou indirecte tout élément du MM, il suffit de suivre les relations de *containment* (compositions)

– Nous partons donc de la classe racine d'une SM, qui s'appelle SM

```
transformation SM2PN(in inSM:sm, out outPN:pn);
```

```
main() {
```

```
    inSM.rootObjects()[SM]->map SM2PetriNet();
```

```
}
```

```
mapping SM :: SM2PetriNet() : PetriNet {
```

```
    name := 'From SM';
```

```
    state += self.state->map State2State();
```

```
    transition += self.transition->map Trans2Trans();
```

```
}
```

self fait référence à l'objet source

Le fait qu'on parle d'un PetriNet créé par la règle est implicite (result)

Relations d'arité multiple

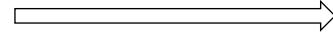
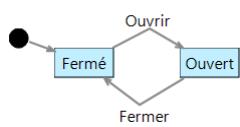
Impératif : l'ordre d'appel des transfos est explicité

Sections init et end, conditionnelle

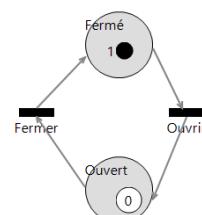
```
mapping sm::State :: State2State() : pn::State {
init {log("Creation etat "+self.name);}
name := self.name;
if self.inTrans.source->selectByType(Initial)->size()>0 then {
marking := self.map CreateMarking(1) ;
} else {
marking := self.map CreateMarking(0) ;
} endif;
end {log("Fin creation etat "+self.name);}
}
```

OCL

```
mapping sm::State :: CreateMarking(in i:Integer) : pn::Marking {
init {log("Creation marking");}
value := i;
end {log("Fin creation marking");}
}
```



Creation etat Fermé
Creation marking
Fin creation marking
Fin creation etat Fermé
Creation etat Ouvert
Creation marking
Fin creation marking
Fin creation etat Ouvert

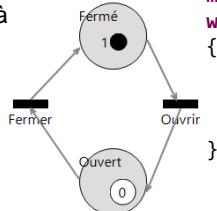


Mapping conditionnel et section population

Garde: ici on ne veut appliquer la règle que sur les transitions autres que la transition initiale

Section que l'on peut nommer population.
Attention, dans ce cas il faut passer par le nom complet, result, pour accéder aux propriétés et attributs de l'objet créé

Accède à la trace de ce qui a été créé avant, ici on veut récupérer le nom de la pn::Transition pour l'objet self, ainsi que Le pn::State correspondant à self.dest



```

mapping sm::Transition::Trans2Trans() : pn::Transition
when {self.source.ocLIstTypeOf(State);} {
population {
    result.name:=self.condition;
    result.inputEdge += self.map DestTrans2InputEdge();
    result.outputEdge += self.map SourceTrans2OutputEdge();
}
}

mapping sm::Transition::DestTrans2InputEdge() : InputEdge
when {self.source.ocLIstTypeOf(State);}
{
    weight:=1;
    transition:=self.resolveone(pn::Transition);
    state:=self.dest.resolveone(pn::State);
}

mapping sm::Transition::SourceTrans2OutputEdge() : OutputEdge
when {self.source.ocLIstTypeOf(State);}
{
    weight:=1;
    transition:=self.resolveone(pn::Transition);
    state:=self.source.resolveone(pn::State);
}

```

Autres éléments de transformation QVTo

- On peut hériter d'une règle afin de la réutiliser par exemple dans des sous-classes
 - inherit
- On peut aussi utiliser les clauses when de plusieurs règles, qui normalement s'excluent mutuellement, pour les fusionner en une seul
 - merge
- On peut utiliser la disjonction typiquement lorsque le polymorphisme induit par l'architecture de modèle permet de choisir une règle ou une autre en fonction de la sous-classe
 - disjuncts
- Modularité
 - On peut utiliser des transfos existant dans un autre module
 - ❖ Soit par access, soit par extends
 - ✓ Extends permet d'utiliser l'héritage
- Classes et propriétés intermédiaires
 - On peut créer des classes et des propriétés intermédiaires, qui ne seront pas générées
 - ❖ intermediate class ou property
- Lorsqu'on génère plusieurs modèles d'entrée ou de sortie, en transfo endogène, on voudra utiliser les model extent permettant de discriminer pour une classe donnée, de quel MM on parle.

Le langage ATL

- ATL (ATLAS Transformation Language) :
 - développé dans le cadre du projet ATLAS au LINA à Nantes.
 - Page principale : <http://www.eclipse.org/atl/>
 - Manuel utilisateur et autres documentations accessibles sur : www.eclipse.org/atl/documentation/

- ATL se compose :
 - d'un langage de transformation (s'appuyant sur OCL)
 - d'un compilateur et d'une machine virtuelle
 - d'un IDE s'appuyant sur Eclipse

Ingénierie Dirigée par les Modèles

Structure générale d'ATL

- Le langage ATL permet de faire trois types d'utilisation :

- **Modules :**
 - Le module permet de spécifier les transformations
- **Requêtes :**
 - La requête permet de faire des calculs qui retournent un type primitif (chaîne de caractères, entier, etc.)
- **Librairies :**
 - Une librairie permet de factoriser et structurer le code. Elle peut être importée par des modules ou d'autres librairies.

- Chaque type est défini dans un fichier distinct portant l'extension «.atl »

Ingénierie Dirigée par les Modèles

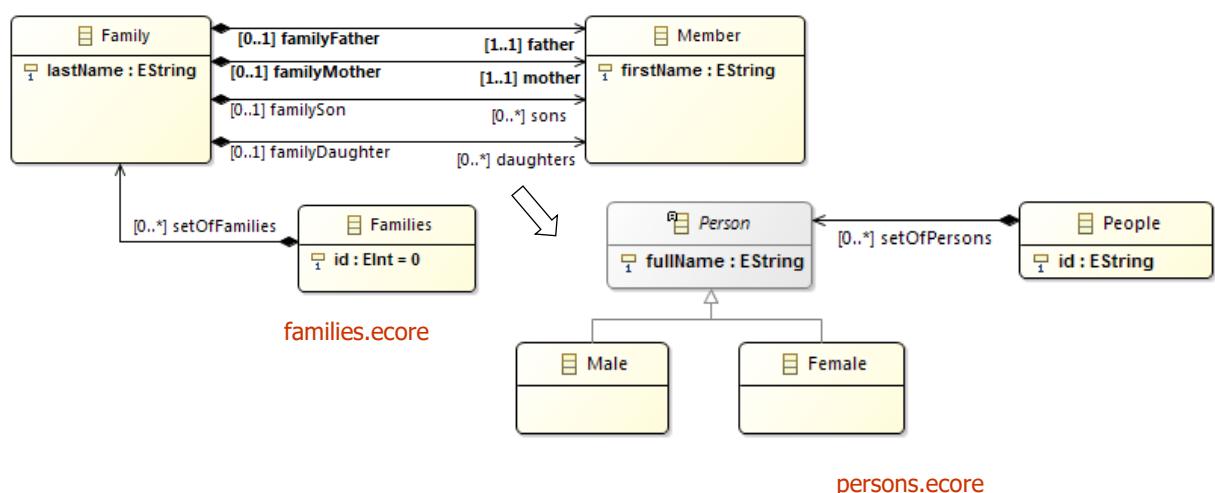
Modules ATL

➤ Un module ATL se compose :

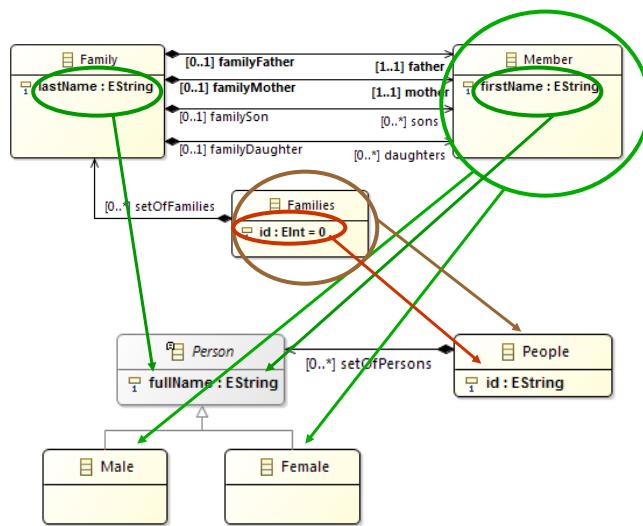
- Entête (Header section) : permet de définir les attributs relatifs à la transformation;
- Import de bibliothèques (Import section) : section optionnelle pour importer des librairies ATL;
- Opérations (Helpers) : ce sont des méthodes (comme les méthodes Java)
- Règles de transformation (Rules) : permettent de décrire la façon dont les éléments cibles sont générés à partir des éléments sources.

Exemple de MM source et destination

➤ Issu de la documentation Eclipse



Transformation visée



Ingénierie Dirigée par les Modèles

121

Section header ATL

➤ Déclaration du module :

– Syntaxe :

```
module module_name;
create output_models from input_models;
```

– Exemples :

```
-- @path Families=/Families2Persons/families.ecore
-- @path Persons=/Families2Persons/persons.ecore
```

```
module Families2Persons;
create OUT : Persons from IN : Families;
```

Ingénierie Dirigée par les Modèles

122

Section import d'ATL

➤ Import de librairies (optionnel):

- Syntaxe :

```
uses extensionless_library_file_name;
```

- Exemple :

❖ Pour importer la librairie strings implémentée dans « string.atl »

```
uses strings;
```

```
uses GeometryLib;
```

```
library GeometryLib;

helper def: PIdiv180 : Real = 180.t
-- and some further geometric global constants

-- adds two vectors
helper def : forward( a : TupleType,
                      b : TupleType,
                      TupleType)
{
    Tuple {
        x = a.x + b.x,
        y = a.y + b.y,
        z = a.z + b.z
    };
}

-- subtracts the second from the first
helper def : backward(a : TupleType,
                      b : TupleType,
                      TupleType)
```

Helpers ATL

➤ Opérations du module :

- Chaque helper est une méthode définie par un contexte (e.g. une classe)
- Les helpers qui portent le même nom ne doivent pas avoir la même signature

- Syntaxe :

```
helper [ context contextName ] def : helperName(parameters) : returnType = atl_expression;
```

- Exemples :

```
helper context Integer def : carre() : Integer = self * self ;
```

Helpers ATL/suite

➤ Opérations du module :



```

helper context Families!Member def: isFemale() : Boolean =
    if not self.familyMother.oclIsUndefined() then
        true
    else
        if not self.familyDaughter.oclIsUndefined() then
            true
        else
            false
        endif
    endif;
    
```

Règles ATL

➤ Règles de génération du module :

- Matched rules : règle déclenchée sur un élément du modèle
 - ❖ Génération d'un élément cible à partir d'un élément source
 - ❖ La règle s'exécute une seule fois
- Lazy rules : règle déclenchée par une autre « matched rule »
 - ❖ Même structure que matched rule + mot clé lazy
 - ❖ La règle peut être exécutée plusieurs fois
 - ❖ Peuvent être uniques
- Called rules : règle déclenchée par une autre « matched rule »
 - ❖ Génération d'un élément cible sans se référer à un élément source
 - ❖ La règle peut être exécutée plusieurs fois
 - ❖ Les called rules ressemblent aux helpers

Syntaxe des règles ATL

➤ Structure des matched rules :

```
rule rule_name { --nom de la règle
  From -- spécification de l'élément source
  inVar: inMetaModel!ClassNameIn [( condition )] -- condition pour le filtrage
  [Using { variables locales --déclaration [et initialisation] des variables }]
  To --l'élément cible outVar : outMetaModel!ClassNameOut ( initialisation)
  [do {
    Statements --instructions à exécuter après la création des éléments cibles.
  }]
}
```

Exemples de règles

Exemple d'une *lazy rule* :

```
lazy rule Member2Female {
  from
    s : INFamilies!Member (s.isFemale())
  to
    t : OUTPersons!Female (
      fullName <- s.firstName + ' ' + s.familyName
    )
}
```

Exemple d'une simple *matched rule* :

```
rule Member2Male {
  from
    s : INFamilies!Member (not s.isFemale())
  to
    t : OUTPersons!Male (
      fullName <- s.firstName + ' ' + s.familyName
    )
}
```

Exemple d'une matched rule complexe:

```

rule Families2People {
from s : INFamilies!Families
using {
    allFamilies : Sequence(INFamilies!Family)=s.setOfFamilies;
    allFemales : Sequence(INFamilies!Member)=INFamilies!Member
    >allInstances()->select(elt | elt.isFemale()==true);
    idVar : String='';
}
to t : OUTPersons!People (
    id <- 'coming_from_element_'+s.id.toString(),
    setOfPersons <- OUTPersons!Male->allInstances()->
    union(allFemales->collect(elt|thisModule.Member2Female(elt)))
)
do {
    idVar <- '_and_'+allFamilies.size().toString()+'_families_are_found';
    t.id<-t.id+idVar;
}
}

```

Autres langages M2M

- Langages opérationnels/Impératifs
 - ATL (*ATLAS Transformation Language*)
 - YAMTL (*Yet Another Model Transformation Language*) Langage textuel proche du Java intégré à Xtend
 - QVTo
 - Kermeta
 - SiTra
- Langages déclaratifs/relationnels basés
 - ATL
 - YAMTL
 - QVT Relational
 - Tefkat
 - Langages basés transformation de graphe, graphiques
 - ❖ VIATRA2, AGG, ATOM3, Fujaba, MOFLON, MOLA, GReAT

Acceleo

Transformation de modèles Le cas Model-to-Text

Ingénierie Dirigée par les Modèles

131

Transformation M2T

- Objectif de la transformation M2T :
 - Engendrer un texte à partir d'un modèle
- Intérêt de la transformation M2T :
 - Génération de code (exemple : code Java à partir de UML)
 - Edition de la documentation
 - Génération d'un texte d'entrée d'un outil (pour l'analyse et la simulation)
 - Création d'une syntaxe concrète textuelle (pour un DSL)
- Comment élaborer une transformation M2T :
 - Utilisation des parseurs :
 - ❖ Exemple : XML/XSLT
 - Utilisation des langages de programmation :
 - ❖ Exemple : API Java d'EMF (= API du Modèle Ecore)
 - Utilisation des templates de transformations :
 - ❖ Exemple : JET/Acceleo

Ingénierie Dirigée par les Modèles

132

Approche M2T à base de templates

➤ Définition d'un template :

- Une structure qui se compose du texte à générer en se basant sur les métadonnées.

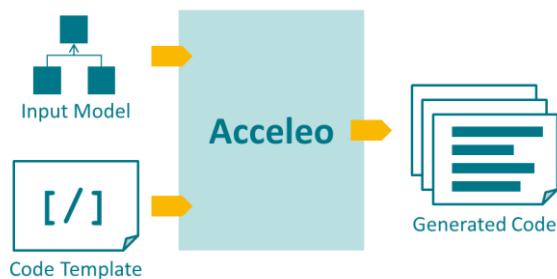
➤ Définition d'un template par l'OMG

- Le standard MTL : Model to Text Language
- Spécification d'un modèle de texte avec des espaces réservés pour les données à extraire à partir de modèles traités.
- Les espaces réservés sont des expressions spécifiées sur les éléments du méta-modèle.
- Les expressions représentent des requêtes de sélection et d'extraction des valeurs à partir de modèles.
- Les valeurs sont converties en fragments de texte en utilisant un langage d'expression avec une bibliothèque de manipulation de chaînes.

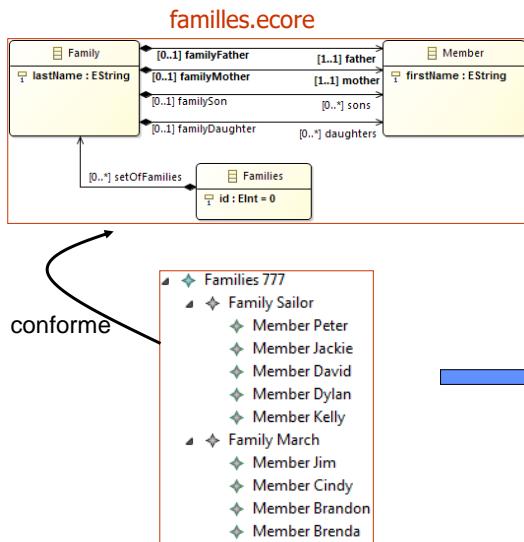
Acceleo, langage de création de templates M2T

➤ Acceleo :

- Editeur de générateurs (templates) de code à partir de modèles MOF ou Ecore.
- Le langage est basé sur AQL et permet d'importer des librairies Java.
- Site officiel : <http://www.acceleo.org>
- Acceleo dispose de générateurs prêts à l'emploi



Exemple : du MM familles à une page HTML



→ Page HTML

[La page 777]

La famille Sailor

Le père est : Peter

La mère est : jackie

Les enfants sont : David, Dylan et Kelly

La famille March

Le père est : Jim

La mère est : Cindy

Les enfants sont : Brandon et Brenda

Partie déclarative en Acceleo

- Déclaration :
- Référence au méta-modèle
 - Définition du fichier à générer
 - Définition de l'élément racine à partir duquel la génération commence

- Exemple :

```

[module generatehtml('http://www.ensma.fr/lesfamilles')]
[template public maingeneratortemplate(aFamilies : Families)]
[comment @main/]
[file('resultat_'.concat(aFamilies.id.toString()).concat('.html'),false)]

```

Corps du template

```

[/file]
[/template]

```

Titre de la page HTML

```
<head>
<title>
La page [aFamilies.id.toString() /]
</title>
</head>
```

Le corps du template est du texte, dans lequel on glisse des requêtes AQL

La page 777

La famille Sailor

Le père est : Peter

La mère est : jackie

Les enfants sont : David, Dylan et Kelly

La famille March

Le père est : Jim

La mère est : Cindy

Les enfants sont : Brandon et Brenda

Structure répétitive Acceleo

```
[for (f:Family|aFamilies.setOfFamilies)]
<h1>
    La famille [f.lastName /]
</h1>

[for (p: Member | f.father)]
<h3>
    Le père est : [p.firstName /]
</h3>
[/for]

[/for]
```

La page 777

La famille Sailor

Le père est : Peter

La mère est : jackie

Les enfants sont : David, Dylan et Kelly

Structure conditionnelle Acceleo

```
[if ((f.daughters->union(f.sons))->size()>0 )]
Les enfants sont :
[for (e : Member | f.daughters->union(f.sons))
[if ((f.daughters->union(f.sons))->asSequence()->indexOf(e)=1)
[comment *premier element*]
[e.firstName/]
[else]
[if ((f.daughters->union(f.sons))->asSequence()->last()=e)]
[comment *dernier element*]
et [e.firstName/]
[else]
, [e.firstName/]
[/if]
[/if]
[/for][else] [comment *si pas d'enfant dans la famille*]
Cette famille n'a pas d'enfant
[/if]
```

La page 777

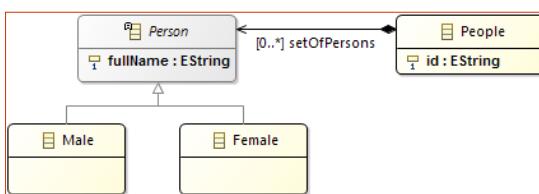
La famille Sailor

Le père est : Peter

La mère est : jackie

Les enfants sont : **David, Dylan et Kelly**

Exercice



Il existe 9 personnes(s)

Les femmes sont :

Jackie Sailor

Cindy March

Kelly Sailor

Brenda March

Les hommes sont :

Brandon March

David Sailor

Dylan Sailor

Jim March

Peter Sailor

L'approche normalisée par l'OMG Model Driven Architecture (MDA)

Ingénierie Dirigée par les Modèles

141

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

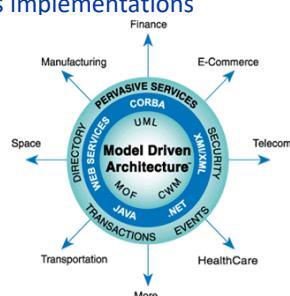
98

99

100

Vision globale MDA

- MDA est une variante particulière de l>IDM
- L'objectif de l'OMG est de réaliser des logiciels standards certifiés MDA
 - Des spécifications plutôt que des implémentations
- MDA est une marque déposée



- La démarche générale :
 - Classification des modèles en quatre types : CIM, PIM, PDM, PSM
 - Processus en Y

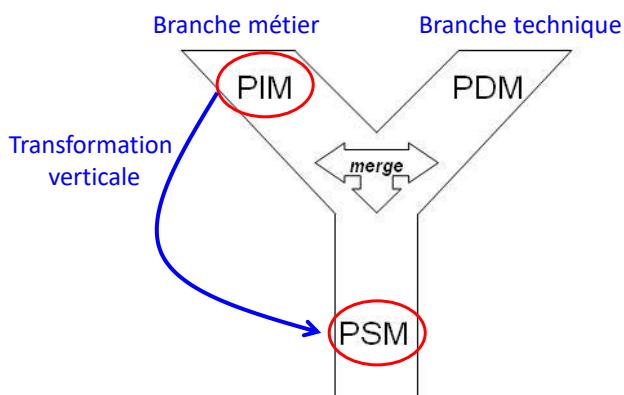
Ingénierie Dirigée par les Modèles

142

Niveaux de modèles OMG

- CIM : Computation Independent Model
 - Modèle métier indépendant de l'informatisation
- PIM : Platform Independent Model
 - Modèle métier abstrait d'un système (pas de déploiement)
 - Exemple : modèle UML d'un système GPS
- PDM : Platform Dependent Model
 - Modèle technique décrivant la plateforme
 - Le terme plateforme n'est pas clair dans le guide MDA
 - Exemple : modèle pour Ada, modèle Androïde, modèle processeur ARM
- PSM : Platform Specific Model
 - Modèle opérationnel contenant les aspects spécifiques de la plateforme
 - Exemple : modèle UML d'un système GPS implémenté en Ada.

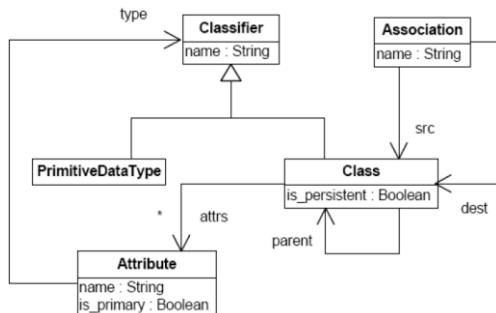
Processus en Y



UML au cœur de la MDA

- UML est un GPML (General Purpose Modeling Language) :

- Les notations UML sont utilisables dans tous les secteurs mais ne sont pas précises → Extension et adaptation : recours aux profils UML



- Profil UML :

- Mécanisme d'extension d'UML pour l'adapter à un contexte métier ou technique particulier
- Spécialisation du méta-modèle UML en ajoutant des nouveaux types d'éléments : **Stéréotypes**, **Tagged Values** (valeurs marquées), **Contraintes** (OCL)

Stéréotypes et tagged values UML

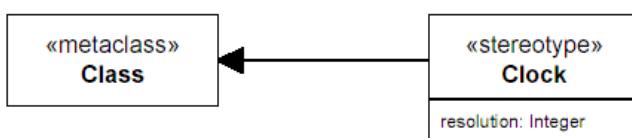
- Stéréotype :

- Spécialisation d'un élément du méta-modèle UML : classe, association, attribut, package ...
- Le nom d'un stéréotype est indiqué entre <<name_of_Stereotype>>
- S'appuie sur une sémantique donnée dans une documentation (purement textuelle)
- Un stéréotype peut avoir des propriétés (tagged values)

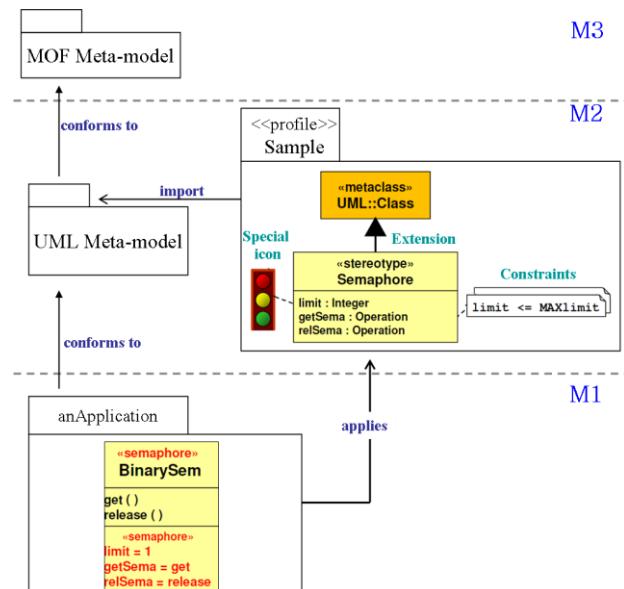
- Tagged Value :

- Attributs spécifiques pour les stéréotypes

- Exemple :



Exemple de profil UML

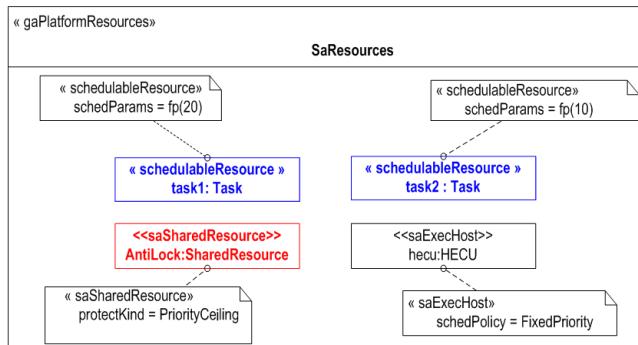


Ingénierie Dirigée par les Modèles

147

Quelques profils UML normalisés

- SysML : System Modelling Language
- MARTE : Modeling and Analysis of Real Time and Embedded systems
- UML profile for Corba
- UML profile for EJB (Enterprise Java Beans)
- CWM : Common Warehouse Metamodel (CWM)



Ingénierie Dirigée par les Modèles

148

Conclusion

- MDE – MDA – MBE – MBSE
- L'utilisation systématique des modèles comme concept clé tout au long du processus d'ingénierie
- Gains : Model Once, Generate Anywhere
 - Maîtrise de la complexité
 - Automatisation de la vérification et de la validation
 - Automatisation du passage entre modèles et/ou espaces technologiques
 - Favorisation de l'interopérabilité entre outils
- État actuel de l'IDM :
 - IDM utilisée de plus en plus dans l'industrie : CEA, Airbus, Thales, etc.
 - DSL et profils : SysML, MARTE, AADL, Capella
- Contraintes techniques :
 - Manque de maturité de certains outils
 - Comment tester la correction et complétude d'une transformation ?

Références

- Diverses documentations Eclipse : <https://wiki.eclipse.org>
 - OCL, AQL, Acceleo, ATL, modélisation, DSL, etc.
- Les cours de :
 - [Yassine Ouhammou](#)
 - [Jean-Philippe Babau](#)
 - [Yamine Ait-Ameur](#)
 - [Alfonso Pierantonio](#)
 - [Eric Cariou](#)
 - [Jean-Marc Jézéquel](#)
 - [Jean Bézivin](#)
- Ouvrages
 - Model-Driven Software Engineering in Practice
 - Ingénierie dirigée par les modèles : des concepts à la pratique
- Tutoriaux
 - <https://www.urbanisation-si.com>

XML

eXtensible Markup Language

Emmanuel Grolleau

B414

Supports de cours en grande partie basés sur ceux de

Yassine Ouhammou

XML – eXtensible Markup Language

Pourquoi XML ?

- Définit comme une alternative
 - Aux fichiers binaires, dépendant de l'architecture sous-jacente
 - Aux multiples formats de fichier nécessitant chacun un interpréteur différent
- De façon à stocker et échanger des données
- « Lisible » par un humain doté d'un peu de patience
- Surtout lisible par une machine
- XML est partout
 - IDM, où le schéma XMI basé XML est très répandu
 - Bases de données
 - Technos internet
 - ❖ Même si HTML, autre langage balisé, n'est pas XML, le DOM (Document Object Model) et les outils sont similaires, voire partagés avec le monde XML
 - Vos documents office sont en fait des fichiers zip contenant du XML

XML – eXtensible Markup Language

Exemple de code XML : le slide 1 du cours d'IDM

- Faites l'expérience, ouvrez un fichier .docx, .pptx, etc. avec un outil de gestion d'archives (type 7zip ou autre)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<p:sld
    xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main"
    xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
    xmlns:p="http://schemas.openxmlformats.org/presentationml/2006/main">
    <p:cSld>
        <p:spTree>
            <p:nvGrpSpPr>
                <p:cNvPr id="1" name="" />
                <p:cNvGrpSpPr />
                <p:nvPr />
            </p:nvGrpSpPr>
        ETC...
        <p:txBody>
            <a:bodyPr />
            <a:lstStyle />
            <a:p>
                <a:r>
                    <a:rPr lang="fr-FR" dirty="0"/>
                    <a:t>Ingénierie Dirigée par les Modèles</a:t>
                </a:r>
                <a:endParaRPr lang="en-US" dirty="0"/>
            </a:p>
        </p:txBody>
    ETC...
</p:sld>
```

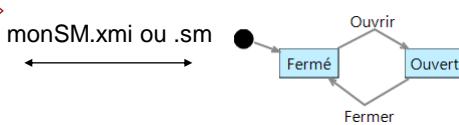
XML – eXtensible Markup Language

3

XML dans l'IDM

- Lorsque nous éditons un modèle .ecore ou une instance .xmi, nous avons en fait de l'XML

```
<?xml version="1.0" encoding="UTF-8"?>
<sm:SM xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:sm="http://www.ensma.fr/idm/sm"
    xsi:schemaLocation="http://www.ensma.fr/idm/sm sm.ecore">
    <initial transition="//@transition.2"/>
    <state name="Fermé"
        outTrans="//@transition.1"
        inTrans="//@transition.0 //@transition.2"/>
    <state name="Ouvert"
        outTrans="//@transition.0"
        inTrans="//@transition.1"/>
    <transition
        condition="Fermer"
        source="//@state.1"
        dest="//@state.0"/>
    <transition
        condition="Ouvrir"
        source="//@state.0"
        dest="//@state.1"/>
    <transition
        source="//@initial"
        dest="//@state.0"/>
</sm:SM>
```



XML – eXtensible Markup Language

4

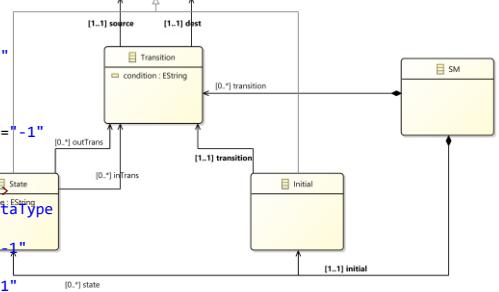
Fichier .ecore du modèle StateMachine

➤ Référencé dans les instances `xmlns:sm=http://www.ensma.fr/idm/sm`

➤ Définit de nouvelles balises et d'attributs

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="sm" nsURI="http://www.ensma.fr/idm/sm" nsPrefix="sm">
    <eClassifiers xsi:type="ecore:EClass" name="Initial" eSuperTypes="#/PseudoState">
        <eStructuralFeatures xsi:type="ecore:EReference" name="transition" lowerBound="1"
            eType="#//Transition"/>
    </eClassifiers>
    <eClassifiers xsi:type="ecore:EClass" name="SM">
        <eStructuralFeatures xsi:type="ecore:EReference" name="initial" lowerBound="1"
            eType="#//Initial" containment="true"/>
        <eStructuralFeatures xsi:type="ecore:EReference" name="state" upperBound="-1"
            eType="#//State" containment="true"/>
        <eStructuralFeatures xsi:type="ecore:EReference" name="transition" upperBound="-1"
            eType="#//Transition" containment="true"/>
    </eClassifiers>
    <eClassifiers xsi:type="ecore:EClass" name="State" eSuperTypes="#/PseudoState">
        <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType"
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
        <eStructuralFeatures xsi:type="ecore:EReference" name="outTrans" upperBound="-1"
            eType="#//Transition"/>
        <eStructuralFeatures xsi:type="ecore:EReference" name="inTrans" upperBound="-1"
            eType="#//Transition"/>
    </eClassifiers>
    ETC...
    <eClassifiers xsi:type="ecore:EClass" name="PseudoState" abstract="true"/>
</ecore:EPackage>
```

XML – eXtensible Markup Language



5

Formats d'échange : XML vs. JSON

- Dans ce cours nous parlerons beaucoup d'XML (eXtensible Markup Language) et XMI (XML Metadata Interchange) qui dispose de nombreux outils permettant de le lire ou de le générer aisément, surtout à la base des normes OMG pour l'échange
 - Inconvénient: si initialement pensé pour être lu par la machine ou l'humain (format textuel), il est en réalité peu lisible
- Ce n'est pas, loin s'en faut, le seul format d'échange utilisé aujourd'hui. Leur but principal est d'échanger des objets
 - listes de « property – value »,
 - « value » peut être un objet => hiérarchie,
 - relations éventuelles entre objets,
 - éventuellement être structuré, s'adosser à une sémantique
- Exemple de normes d'échange de données
 - JSON (JavaScript Object Notation) 2003
 - ❖ Basé sur un JSON schéma, écrit en JSON, donc **réflexif**
 - ❖ Il existe des outils inférant un schéma à partir d'un fichier JSON donné
 - ❖ Très léger, lisible, très bien outillé
 - ❖ Directement utilisable comme objet javascript
 - ❖ Un typage relativement faible cependant, et une difficulté à combiner plusieurs schémas

XML – eXtensible Markup Language

6

Formats d'échange : JSON

- Extrait du schéma JSON utilisé pour les extensions Google Chrome – et d'un manifest.json

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "additionalProperties": true,
    "definitions": {
        "action_v2": {
            "type": "object",
            "properties": {
                "default_title": {
                    "type": "string",
                    "description": "Tooltip for the main toolbar icon."
                },
                "default_popup": {
                    "$ref": "#/definitions/uri",
                    "description": "The popup appears when the user clicks the icon."
                },
                "default_icon": {
                    "anyOf": [
                        {
                            "type": "string",
                            "description": "FIXME: String form is deprecated."
                        },
                        {
                            "type": "object",
                            "description": "Icon for the main toolbar.",
                            "properties": {
                                "19": {
                                    "$ref": "#/definitions/icon"
                                }
                            }
                        }
                    ]
                }
            }
        }
    }
}
```

XML – eXtensible Markup Language

Et au-delà d'XML et JSON ?

➤ YAML (Yet Another Markup Language) 2001

- Peut se baser sur un... JSON schéma
- Utilise l'indentation pour éviter accolades et crochets dans les définitions de listes d'objets hiérarchiques
- En contrepartie nombreuses constructions à connaître pour l'écrire

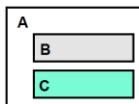
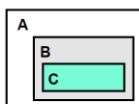
```
--- !<tag:clarkevans.com,2002:invoice>
invoice: 34843
date   : 2001-01-23
bill-to: &id001
    given : Chris
    family : Dumars
    address:
        lines: |
            458 Walkman Dr.
            Suite #292
            city   : Royal Oak
            state  : MI
            postal : 48046
ship-to: *id001
```

➤ Mais aussi : Google Protocol buffers, ConfigObj (fichiers ini à l'ancienne), et des dizaines d'autres formats

XML – eXtensible Markup Language

Revenons à XML : Structuration d'un fichier

- Un fichier XML contient un prologue suivi du corps qui est un arbre, démarrant par une **racine**, et contenant des blocs imbriqués repérés par des balises

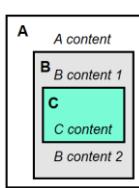

BIEN


<A>

 <C/>

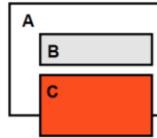


<A>
 B content
 <C/>



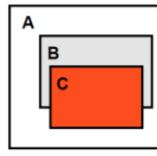
<A> A content

 B content 1
 <C> C content </C>
 B content 2

PAS BIEN


<A>

 <C/>



<A>

 <C>

 </C>



XML – eXtensible Markup Language

Balises et attributs

- Un attribut contient une valeur, un élément, ajouté aux balises

Elément racine

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<lesgens>
    <un_gen>
        <prenom>Manu</prenom>
        <nom>G.</nom>
        <naissance>Millénaire dernier</naissance>
    </un_gen>
    <un_gen>
        <prenom>Mikky</prenom>
        <nom>R.</nom>
        <naissance>Pas de la dernière pluie</naissance>
    </un_gen>
</lesgens>
```

Prologue
Elément vide, la balise est auto fermante

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<lesgens>
    <un_gen prenom="Manu" nom="G." naissance="Millénaire dernier" />
    <un_gen prenom="Mikky" nom="R." naissance="Pas de la dernière pluie" />
</lesgens>
```

Eléments enfant de la racine
Attribut prenom de l'élément un_gen, doit être entre cotes ou guillemets
Contenu de l'élément prenom

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<lesgens>
    <un_gen prenom="Manu" nom="G." naissance="Millénaire dernier" />
    <un_gen prenom="Mikky" nom="R." naissance="Pas de la dernière pluie" />
</lesgens>
```

- Attribut ou élément ?

- Pas de règle
- Mais élément plus évolutif

❖ Si naissance se décompose plus tard en année, mois, jour ?

DOM (Document Object Model) navigateur internet

XML – eXtensible Markup Language

Quelques règles

➤ Toute balise ouverte est fermée (auto-fermée par `>/` pour les éléments vides)

➤ Un seul élément racine, les éléments sont correctement imbriqués

➤ Les noms de balises ou d'attributs

- N'ont pas d'espace

➤ Cinq caractères sont interdits en dehors de cotes ou guillemets

Entité	Valeur	Exemple	Résultat
<code>&lt;</code>	less than (<)	<code>10 &lt; 100</code>	<code>10<100</code>
<code>&gt;</code>	greater than (>)	<code>x &gt; 119</code>	<code>x>119</code>
<code>&amp;</code>	ampersand (&)	<code>AT&amp;T</code>	<code>AT&T</code>
<code>&apos;</code>	apostrophe (')	<code>It&apos;s XML</code>	<code>It's XML</code>
<code>&quot;</code>	quote(" ")	<code>&quot;Wow!&quot;</code>	<code>" Wow!"</code>

➤ Les noms de balises et attributs sont sensibles à la casse

- Ne peut commencer par un nombre ou caractère spécial

- Eviter les caractères . (utilisés pour les attributs d'objets), : (utilisés pour les namespaces), - (peut se confondre avec l'opérateur)

- Ne peut commencer par xml ou XML ou Xml, etc.

➤ Respecter si possible une convention de nommage

- Minuscule `<labalise>`, majuscule `<LABALISE>`, snake `<la_balise>`, Pascal `<LaBalise>`, Camel `<laBalise>`

➤ Les commentaires sont entre `<!--` et `-->`, ne peuvent contenir --

Espaces de noms (namespaces)

➤ On fait souvent référence à plusieurs documents définissant des balises, certaines peuvent avoir le même nom, on utilise donc des ns (*namespaces*) pour les différencier

- Le nom de balise est simplement préfixé par le ns suivi de « : »

`xmlns:.ecore="http://www.eclipse.org/emf/2002/Ecore"`

Le namespace `.ecore` est utilisé pour toute balise définie dans le fichier dont l'URI est `http://www.eclipse.org/emf/2002/Ecore`

`.ecore:EPackage`

Pas d'ambiguïté possible: on parle bien de la balise `EPackage` définie dans le fichier dont l'URI est `http://www.eclipse.org/emf/2002/Ecore`

DTD (Document Type Definition)

Définition des documents XML par DTD

XML – eXtensible Markup Language

13

Intérêts de définir la structure d'un document XML

- Ils sont évidents – écrire n'importe quoi n'importe comment ou écrire sous un format avec une sémantique donnée permettant d'utiliser des outils et d'échanger des informations ? (reste du slide laissé blanc)

XML – eXtensible Markup Language

14

DTD : la préhistoire d'XML

➤ La Document Type Definition (DTD)

- Le principe de base est simple

- <!ELEMENT nom-balise (#PCDATA ou sousbalise1, sousbalise2, sousbalise3, etc.)>

- Exemple de DTD

```
<!ELEMENT personne (nom, prenom, age)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT age (#PCDATA)>
```

- Un fichier XML conforme à cette DTD peut être

```
<personne>
    <nom>Mikky</nom>
    <prenom>Richard</prenom>
    <age>C'est son petit secret</age>
</personne>
```

XML – eXtensible Markup Language

DTD et cardinalité des éléments

➤ Le | exprime l'alternative entre des éléments, l'* exprime la présence multiple

```
<!ELEMENT roman (chapitre*)>
<!ELEMENT chapitre (titre|resume)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT resume (#PCDATA)>
```

- Un fichier XML conforme à cette DTD peut être

```
<roman>
    <chapitre>
        <titre>Cultiver les oignons en Poitou</titre>
    </chapitre>
    <chapitre>
        <resume>Savez-vous planter des oignons avec les pieds ?
        Moi non plus.</resume>
    </chapitre>
</roman>
```

XML – eXtensible Markup Language

DTD et attributs

- On utilise le mot-clé « ATTLIST » pour définir des règles sur les attributs
 - <!ATTLIST balise attribut type mode valeur_par_defaut >
 - **type** : pour décrire le type de l'attribut
 - **mode** : indication sur l'obligation de présence de l'attribut ou son unicité
 - ❖ ID signifie que la valeur doit être unique pour la balise donnée
 - ❖ IDREF fait référence à un autre identifiant unique
 - ❖ #REQUIRED signifie que l'attribut est requis, l'inverse est #IMPLIED
 - ❖ #FIXED signifie que l'attribut est immuable
 - On Trouve le | pour définir des valeurs possibles


```
<!ATTLIST pere nomcomplet ID mode >
<!ATTLIST fils nomcomplet ID mode sonpere IDREF mode >
```
 - Exemple d'XML valide


```
<pere nomcomplet='aaaa' />
<pere nomcomplet='bbbb' />
<fils nomcomplet='cccc' sonpere='bbbb' />
```

XML – eXtensible Markup Language

17

DTD et alias d'entités

- On peut créer un alias
 - <! ENTITY entityname "value">
- Et la référencer par &entityname;

```
<!DOCTYPE repertoire [
<!ELEMENT repertoire (établissement)+>
<!ELEMENT établissement (nom,type)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ENTITY ensm "école nationale supérieure de mécanique & d'ingénierie de Poitiers">
<!ENTITY up "université de Poitiers">
]>
<repertoire>
<établissement>
<nom>ensma</nom>
<type>école d'ing.</type>
</établissement>
<établissement>
<nom>&up;</nom>
<type>université</type>
</établissement>
</repertoire>
```



```
<repertoire>
- <établissement>
- <nom>
    école nationale supérieure de mécanique & d'aérotechnique
    </nom>
    <type>école d'ing.</type>
</établissement>
- <établissement>
    <nom>université de Poitiers</nom>
    <type>université</type>
</établissement>
</repertoire>
```

XML – eXtensible Markup Language

18

Exemple de DTD

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE formation [
    <!ELEMENT formation (module*)>
    <!ELEMENT module (code, nom)>
    <!ELEMENT code (#PCDATA)>
    <!ELEMENT nom (#PCDATA)>
]>
<formation>
    <module>
        <code>SETR</code>
        <nom>Systèmes embarqués temps réel</nom>
    </module>
    <module>
        <code>IDD</code>
        <nom>Ingénierie de données</nom>
    </module>
</formation>
```

```
<?xml version = "1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE formation SYSTEM "formation.dtd">
<formation>
    <module>
        <code>SETR</code>
        <nom>Systèmes embarqués temps réel</nom>
    </module>
    <module>
        <code>IDD</code>
        <nom>Ingénierie de données</nom>
    </module>
</formation>
```

```
<!ELEMENT formation (module*)>
<!ELEMENT module (code, nom)>
<!ELEMENT code (#PCDATA)>
<!ELEMENT nom (#PCDATA)>
```

XML – eXtensible Markup Language

Conclusion

- Quelques limites des DTD :
 - Pas de support pour les types de base comme les entiers, les dates, etc.
 - Pas de dérivation de type
 - Les liens id/idref ne sont pas fortement couplés
- A la limite de l'obsolescence

XML – eXtensible Markup Language

XSD (XML Schema Definition)

Définition des documents XML par XSD

XML – eXtensible Markup Language

21

Introduction

- En plus du contrôle de la structure des documents, les schémas XML ont plusieurs avantages :
 - Le typage de données : système de typage complet
 - Les contraintes (par exemple ID et IDREF) : contrôler précisément le nombre d'occurrences et la liaison entre attributs
 - Utilisation des espaces de noms
 - ==> XML Schéma est un formalisme qui permet de définir des contraintes en matière de syntaxe, de structure
- Le langage des schémas XML est un langage assez complexe
 - Les schémas XML s'écrivent grâce au XML
 - Les schémas des schémas XML sont des documents XML bien formés.
 - ==> Ce sont donc des documents XML qui décrivent d'autres documents XML.
 - ❖ Sont des MMM
 - C'est un standard W3C

XML – eXtensible Markup Language

22

Structuration d'un XSD

➤ Le document XML Schéma est un XML mais qui porte l'extension « .xsd »

- Un schéma XML se compose de 2 parties :

- ❖ Le prologue

```
<?xml version="1.0" encoding="UTF-8" ?>
```

- ❖ Le corps : ensemble de balises dont la racine est imposée

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
.....
```

```
</xsd:schema>
```

XML – eXtensible Markup Language

Premier exemple XSD

➤ DTD basique

```
<!ELEMENT address (name, street+, town, pc, country)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT town (#PCDATA)>
<!ELEMENT pc (#PCDATA)>
<!ELEMENT country (#PCDATA)>
```

➤ Représentation plus précise avec XSD

```
<xsd:element name= "address">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="street" type="xsd:string" minOccurs="1" maxOccurs="2"/>
      <xsd:element name="town" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="pc" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="country" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

XML – eXtensible Markup Language

Attributs, types simples et complexes XSD

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="book" type="bookType"/>
    <xsd:element name="comment" type="xsd:string"/>

    <xsd:complexType name="bookType">
        <!-- content model specification here -->
        <xsd:attribute name="publicationDate" type="xsd:date"/>
    </xsd:complexType>
</xsd:schema>

```

- L'élément book est de type complexe bookType
- Un attribut ne peut qu'être de type simple

Attributs requis, optionnels, prohibés et valeur requise ou par défaut

- L'élément « attribute » peut avoir deux attributs optionnels : **use**, **value**

```
<xsd:attribute name="publicationDate" type="xs:date" use="optional" value="2005-11-23"/>
```

use	value	Sémantique
required		l'attribut doit apparaître
required	2005-11-23	l'attribut doit apparaître avec la valeur exacte
optional		l'attribut peut apparaître
optional	2005-11-23	l'attribut peut apparaître, autrement la valeur est 2005-11-25
prohibited		l'attribut ne doit pas apparaître

Référence à des groupes d'attributs

```

<xsd:element name="book" type="bookType"/>
<xsd:element name="comment" type="xsd:string"/>
<xsd:complexType name="bookType">
    <!-- content model specification here -->
    <xsd:attribute ref="bookInfoAttrGroup"/>
</xsd:complexType>
<xsd:attributeGroup name="bookInfoAttrGroup">
    <xsd:attribute name="publicationDate" type="xsd:date"/>
    <xsd:attribute name="ISBN" type="xsd:ID"/>
</xsd:attributeGroup>

```

- Le groupe **bookInfoAttrGroup** peut être utilisé à différents endroits
- Permet de factoriser les définitions

Référence à des éléments définis

```

<xsd:element name="pages" type="xsd:positiveInteger" />
<xsd:element name="auteur" type="xsd:string" />
<xsd:element name="livre">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="auteur" /> <xsd:element ref="pages" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

```

- Là encore, on factorise
 - Avantage d'évolutivité
- Un élément est **global** s'il est défini comme fils de la racine `xsd:schema`, sinon il est considéré comme élément **local**

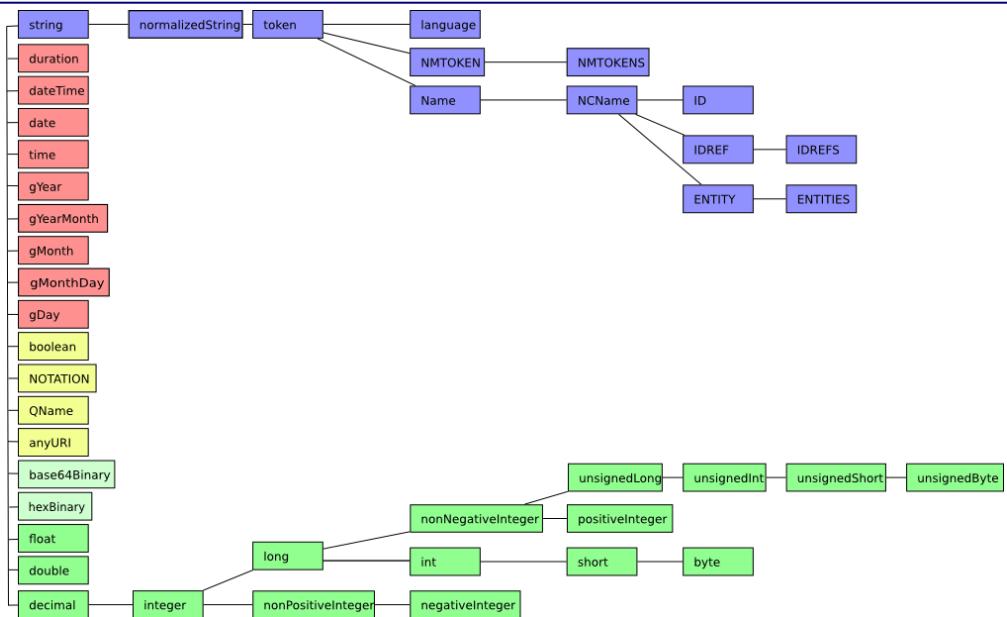
Types dans XSD

- Il existe plus de 40 types de données : string, integer, date, year, CDATA, float, double, binary, ENTITIES, token, byte, etc.
- XSD permet de définir des nouveaux types de données :
 - Créer un type de données totalement nouveau
 - Restreindre ou étendre un type de données existant
- On distingue deux catégories de types : simple et complexe

XML – eXtensible Markup Language

29

Types simple XSD



XML – eXtensible Markup Language

30

Définition de nouveaux types simples

➤ Il y a 3 façons de créer un nouveau type :

- **Restriction** : <restriction> permet de dériver un nouveau type sur la base d'un type existant en ajoutant de nouvelles contraintes
- **List** : <list> permet de créer un nouveau type liste
- **Union** : <union> permet de créer un nouveau type en faisant l'union de plusieurs types existants

XML – eXtensible Markup Language

Création de type de base XSD

Restriction

```

<xsd:simpleType name="SixCountries">
  <xsd:restriction base="CountryList">
    <xsd:length value="6"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Code">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[A-Z][a-z]"/>
  </xsd:restriction>
</xsd:simpleType>
  
```

```

<xsd:simpleType name="myInteger">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="10000"/>
    <xsd:maxExclusive value="99999"/>
  </xsd:restriction>
</xsd:simpleType>
  
```

```

<xsd:simpleType name="Country">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="France"/>
    <xsd:enumeration value="Italia"/>
    <!-- etc ... -->
  </xsd:restriction>
</xsd:simpleType>
  
```

List

```

<xsd:simpleType name="CountryList">
  <xsd:list itemType="Country"/>
</xsd:simpleType>
  
```

Union

```

<xsd:simpleType name="CountryOrCode">
  <xsd:union memberTypes="CountryList Code"/>
</xsd:simpleType>
  
```

XML – eXtensible Markup Language

Types complexes XSD

➤ Séquences

– Vues avant

➤ Choix

```
<xsd:element name="address" type="addressType"/>
<xsd:complexType name="addressType">
    <xsd:choice>
        <xsd:element name="fullAddress" type="xsd:string"/>
        <xsd:element name="email" type="xsd:string"/>
    </xsd:choice>
</xsd:complexType>
```

<address>
 <fullAddress>bla bla bla</fullAddress> Ou <address>
 <email>bla@bla.fr</email>
 </address>

Arité des éléments XSD

Définition	<i>minOccurs</i>	<i>maxOccurs</i>
0 to ? (* en DTD)	0	<i>unbounded</i>
1 to ? (+ en DTD)	1	<i>unbounded</i>
0 to 1 (? en DTD)	0	1
N to M	N	M

```
<xsd:element name="book" type="bookType"/>
<xsd:complexType name="bookType">
    <xsd:sequence>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="abstract" type="xsd:string"/>
        <xsd:element name="publisher" type="xsd:string"/>
        <xsd:element name="author" type="personType" maxOccurs="unbounded"/>
        <xsd:element name="chapterTitle" type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="personType">
    <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

Dérivation par restriction de types complexes XSD

```

<xsd:element name="book" type="bookType"/>
<xsd:complexType name="bookType">
    <xsd:sequence>
        <xsd:element name="title"      type="xsd:string"/>
        <xsd:element name="abstract"   type="xsd:string"/>
        <xsd:element name="publisher"  type="xsd:string"/>
        <xsd:element name="author"    type="personType"
                      maxOccurs="unbounded"/>           <xsd:complexType name="threeAuthorsBookType">
                                                    <xsd:complexContent>
                                                        <xsd:restriction base="bookType">
                                                            <xsd:sequence>
                                                                <xsd:element name="title"      type="xsd:string"/>
                                                                <xsd:element name="abstract"   type="xsd:string"/>
                                                                <xsd:element name="publisher"  type="xsd:string"/>
                                                                <xsd:element name="author"    type="personType"
                                  minOccurs="3" maxOccurs="3"/>
                                                            </xsd:sequence>
                                                        </xsd:restriction>
                                                    </xsd:complexContent>
                                                </xsd:complexType>
    </xsd:sequence>
</xsd:complexType>

```

- Note: il est nécessaire de répéter tous les champs conservés dans le type restreint. En effet, la restriction pourrait porter sur la suppression d'un élément.

Autres types de dérivation et références

- Nous ne le présentons pas, mais ils existent
- Dérivation par extension
 - De type spécialisation, généralisation
- On peut créer des types abstraits (non instanciable) qui devront être étendus pour être utilisés
- Les balises de type key et keyref permettent de demander au DOM de créer des tables avec les éléments référencés et des contraintes de clés étrangères
 - Usage plutôt bases de données

Commentaires XSD: les annotations

- L'annotation est un commentaire qui permet d'enrichir la description d'un schéma XML.
- La balise <annotation> contient deux sous éléments :
 - documentation : son contenu est dédié à l'utilisateur humain
 - appinfo : son contenu est dédié aux applications XML

```
<xsd:annotation>
<xsd:documentation>for human being purposes</xsd:documentation>
<xsd:appinfo>for XML application purposes</xsd:appinfo>
</xsd:annotation>
```

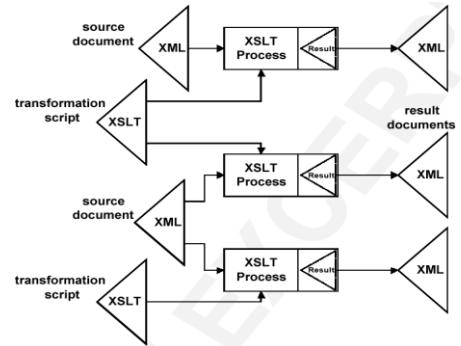
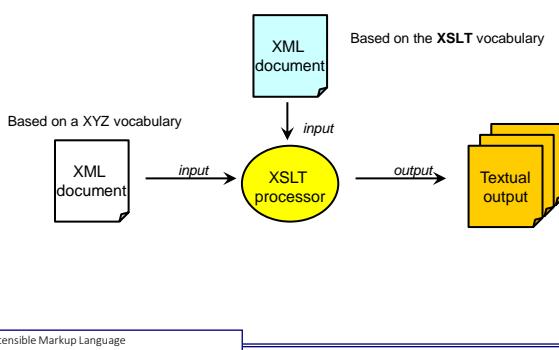
XML – eXtensible Markup Language

XSLT (eXtensible Stylesheet Language Transformation) et XPath Navigation dans et transformation de document XML

XML – eXtensible Markup Language

Qu'est-ce qu'XSLT ?

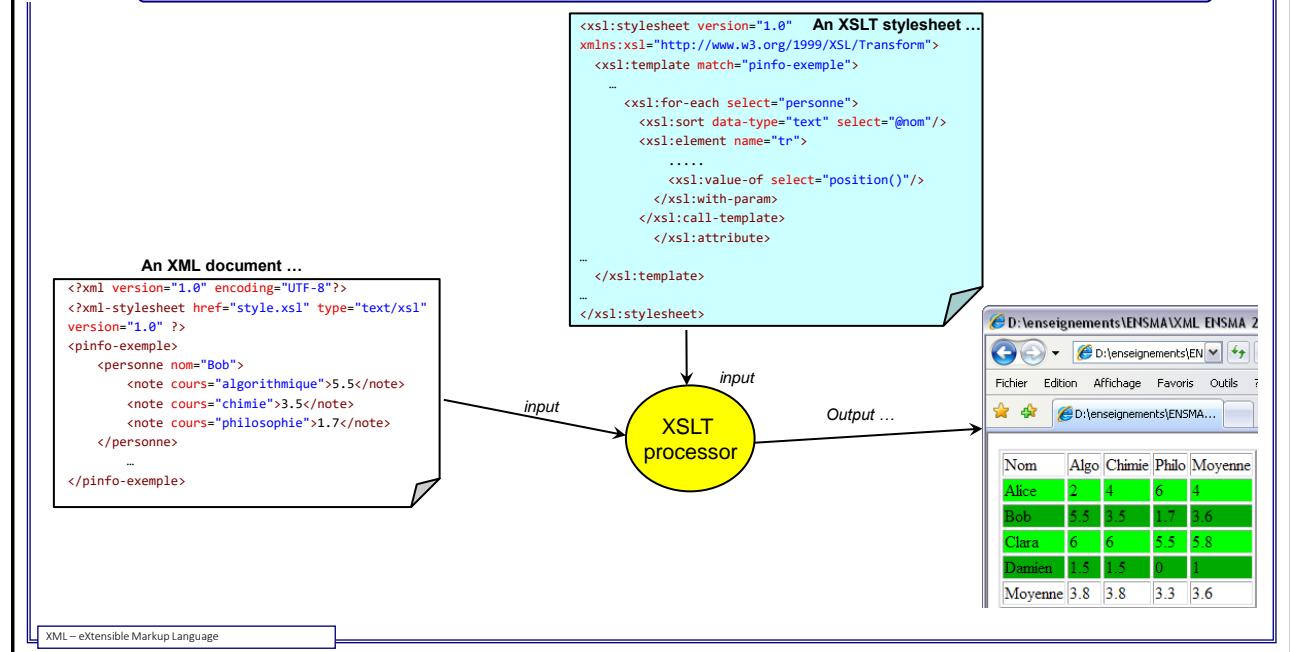
- XSLT (Extensible Stylesheet Language Transformation) est un langage de programmation qui sert à transformer le contenu d'un document XML source en un autre document dont le format et la structure diffère
- Un document XSLT est exprimée comme un document XML, pouvant aussi bien contenir des éléments définis par XSLT que d'autres éléments non définis par XSLT
- XSLT est un standard de la W3C
- Le langage XSLT permet de dériver des documents XML à partir de feuilles de style qui contiennent les règles de transformation (template rules).



Partie déclarative

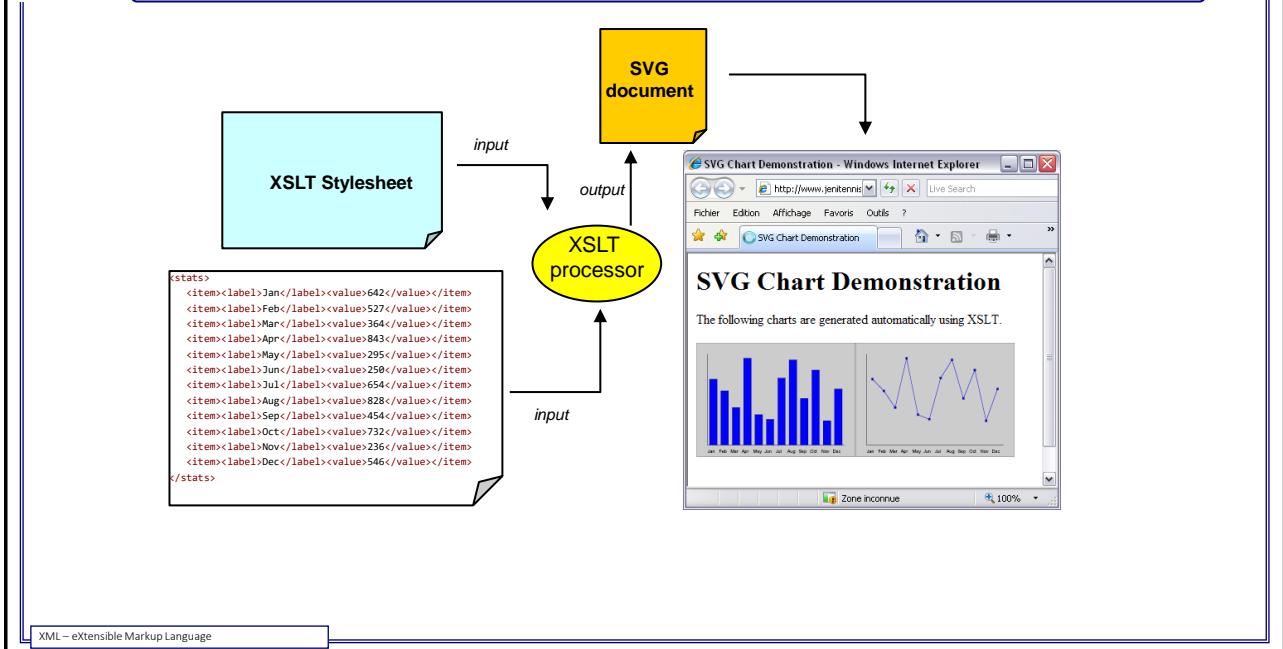
- Namespace : `xmlns="http://www.w3.org/1999/XSL/Transform"`
- Déclaration :
 - `<?xml version="1.0" ?>`
 - `<?xml-stylesheet type="text/xml" href="stylesheet.xml"?>`
- ou bien :
 - `<?xml version="1.0" encoding="ISO-8859-1"?>`
 - `<?xml-stylesheet type="text/xsl" href="feuilleDeStyle.xsl"?>`

Exemple d'usage de XSLT pour générer de l'HTML à partir d'XML



41

Exemple d'usage de XSLT pour générer du SVG à partir d'XML



42

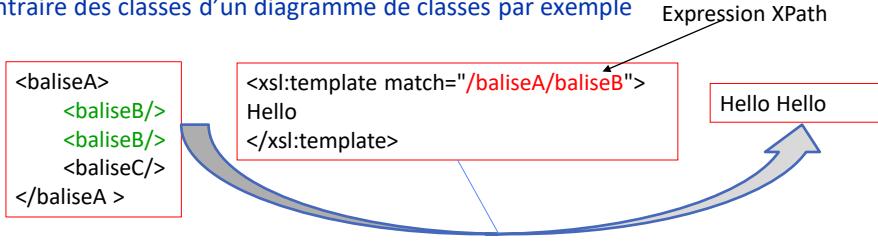
Principe de fonctionnement

➤ L'algorithme général de XSLT est :

- Sélectionner (opération match) les éléments XML du fichier source.
- Pour chaque élément reconnu, générer une sortie sur le fichier cible.

➤ Un programme XSLT est une suite de règles

- Chaque règle est indépendante des autres
- Chaque règle s'occupe de sélectionner un élément dans la source et d'effectuer une écriture dans la cible
- Assez proche des concepts sous-jacents au M2M déclaratif, cependant XML possède un ordre au contraire des classes d'un diagramme de classes par exemple



XML – eXtensible Markup Language

43

Structure d'un fichier XSLT

➤ La balise **<output>** est le 1er fils de la racine du document XSLT, cette balise indique :

- **method** : le format de sortie xml, html ou text.
- **indent=yes** : indique que le fichier généré sera automatiquement indenté.

```

<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html" indent="yes" encoding="iso-8859-1"/>
    <xsl:template match="XPATH">
        ...
    </xsl:template>
    <xsl:template match="XPATH">
        Instruction XSLT et/ou génération de texte sur la sortie
    </xsl:template>
    ...
</xsl:stylesheet>
    
```

XML – eXtensible Markup Language

44

Analogie XML – arborescence – la base d'XPath

```
<contenu>
<paragraphe>Ce paragraphe est le premier paragraphe du document.
</paragraphe>
<remarque>
<paragraphe>L'image suivante
<ressource URIsrc="image.jpg"
titre= "sans titre" type= "jpg"/> montre la couverture du livre.
</paragraphe>
</remarque>
</contenu>
```

```
/ 
| contenu
| | paragraphe
| | | text() = "Ce paragraphe..."
|
| | remarque
| | | paragraphe
| | | | text() = "L'image suivante"
| | | | ressource
| | | | | @URIsrc = "image.jpg"
| | | | | @titre = "sans titre"
| | | | | @type = "jpg"
| | | | | text() = "montre la... "
```

Introduction à XPath

- XPath (XML Path Langage) est un standard W3C
- XPath définit des règles grammaticales pour identifier des nœuds ou des ensembles de nœuds dans des documents XML.
 - langage d'expressions permettant de pointer sur n'importe quel élément d'un arbre XML depuis n'importe quel autre élément de l'arbre.
 - Une expression XPath décrit l'emplacement d'éléments et d'attributs XML comme un chemin similaire à des URL. Elle est nommée chemin de localisation
 - Une expression XPath est composée de segments séparés par « / ».
- Xpath joue un peu le rôle d'OCL dans l'extraction de parties d'un documents XML comme OCL permet d'extraire des parties d'un modèle, soit uniques, soit sous forme de collections
- Une expression XPath est évaluée au sein d'un contexte
 - Le **contexte** = le nœud par rapport auquel cette expression est évaluée
 - Une même expression retourne des résultats différents selon le contexte d'évaluation
- Une expression XPath peut-être absolue
 - Sa résolution est indépendante du contexte ou noeud courant
 - Elle commence dans ce cas par /
- Une expression XPath peut-être relative
 - Sa résolution est dépendante d'un contexte ou noeud courant
 - elle peut commencer par ./ (syntaxe développée)

Eléments renvoyés par XPath

- Une expression XPath renvoie
 - Un ensemble de nœud (sous arbre de l'arbre document), une chaîne de caractères, un booléen, ou un réel, etc.
- Un nœud peut être de type :
 - root, element, text, attribute, namespace, processing instruction, comment

```
<contenu>
<paragraphe>Ce paragraphe est le premier paragraphe du document.
</paragraphe>
<remarque>
    <paragraphe>L'image suivante
        <ressource URIsrc="image.jpg" titre= "sans titre" type= "jpg"/> montre
        la couverture du livre.
    </paragraphe>
</remarque>
</contenu>
```

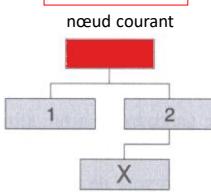
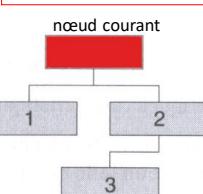
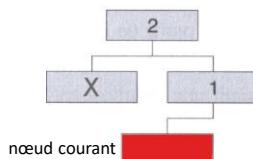
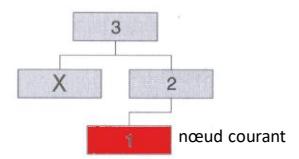
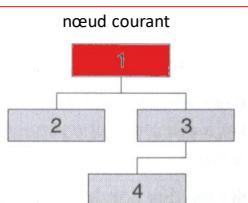
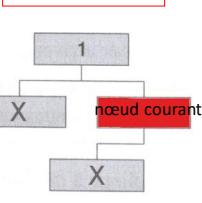
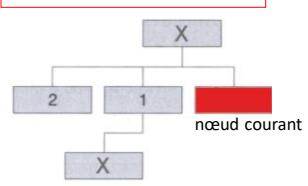
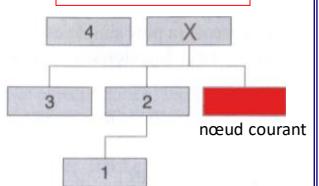
contenu/*
 contenu/remarque[1]
 ../paragraphe
 @type

Exemples XPath

Chemin de localisation XPath

- Une expression Xpath (chemin de localisation) est composée de plusieurs pas de localisation successifs séparés par des /
 - Chaque étape sélectionne à son tour des nœuds par rapport à l'étape précédente
 - Un pas est caractérisé par : un axe, un test de nœud et un prédictat
 - **axe::testnœud[prédictat]**
- Un chemin de localisation a deux syntaxes possibles :
 - Syntaxe abrégée : /elt1/elt2/elt3
 - Syntaxe non abrégée /axe::test[prédictat]/...../axe::test[prédictat]
 - ❖ Elle permet plus de possibilités de sélections (frères, commentaire, processing-instruction, ...)
- **axe** : spécifie la relation d'arborescence entre le nœud de contexte et les nœuds que l'étape de localisation doit sélectionner : parents, fils, ancêtres, voisins.
 - Exemple : child, parent, ancestor, self, etc.
- **test** : indique parmi tous les nœuds de l'axe spécifié, celui ou ceux à considérer comme des candidats. Il suit l'axe et doit être précédé de ::.
 - Exemple : parent::* , child::comment(), child::A
- **prédictat** (optionnel) : il consiste en des conditions sur les nœuds parcourus. Il est encadré par [].

Fonctionnement des axes

Axe « child »

Axe « descendant »

Axe « ancestor »

Axe « ancestor-or-self »

Axe « descendant-or-self »

Axe « parent »

Axe « preceding-sibling »

Axe « preceding »


XML – eXtensible Markup Language

Axe XPath

Axe	Description
ancestor::	Les ancêtres du nœud courant
ancestor-or-self::	Nœud courant et ses ancêtres
attribute::	Attributs du nœud courant
child:: /	Les fils de type <i>element</i> du nœud courant sans attributs et sans textes
descendant:: //	Les descendants de type <i>element</i> du nœud courant
descendant-or-self::	Nœud courant et ses descendants
following::	Tous les nœuds qui suivent le nœud courant dans l'arborescence (<i>ordre dans le document</i>), sauf ses nœuds descendants et attributs

XML – eXtensible Markup Language

Axe Xpath /suite

Axe	Description
following-sibling::	Tous les frères suivants du nœud courant (ordre dans le document), il ne comprend pas les autres frères qui apparaissent avant le nœud courant.
parent::	Le parent du nœud de contexte
preceding::	Tous les nœuds qui précèdent le nœud de contexte dans l'arborescence (ordre dans le document), sauf les nœuds ancêtres et attributs
preceding-sibling::	Tous les frères précédents du nœud courant (ordre dans le document), il ne comprend pas les autres frères qui apparaissent après le nœud courant.
self::	Nœud courant
.	

XML – eXtensible Markup Language

51

Exemples XPath

```

<A>
  <B>
    <C>test ccc 1</C>
    <C>test ccc 2</C>
  </B>
  <B>
    <C>test ccc 3</C>
    <C>test ccc 4</C>
    <D>test ddd 1</D>
  </B>
</A>

```

Exemple	Résultat
A/B[1]/following::*	 <C>test ccc 3</C> <C>test ccc 4</C> <D>test ddd 1</D> <C>test ccc 3</C> <C>test ccc 4</C> <D>test ddd 1</D>
A/B[2]/preceding::*	 <C>test ccc 1</C> <C>test ccc 2</C> <C>test ccc 1</C> <C>test ccc 2</C>
A/B[1]/following::D	<D>test ddd 1</D>
A/descendant::*	<C>test ccc 1</C><C>test ccc 2</C> <C>test ccc 1</C><C>test ccc 2</C> <C>test ccc 3</C><C>test ccc 4</C><D>test ddd 1</D> <C>test ccc 3</C><C>test ccc 4</C><D>test ddd 1</D>

 Amusez-vous <https://www.freeformatter.com/xpath-tester.html>

XML – eXtensible Markup Language

52

Test XPath

Type test	Description	Exemple
Comment()	retourne les nœuds de l'axe de type commentaire	following::comment()
Node()	retourne les nœuds de l'axe de n'importe quel type	preceding::node()
text()	retourne les nœuds de l'axe de type texte (contenu texte d'un élément)	child::text()
*	retourne tous les nœuds de l'axe	ancestor::*
name	retourne les nœuds de l'axe dont le nom est name	child::B

Prédicat XPath

- Les fonctions XPath permettent d'affiner les expressions par l'utilisation de prédictats
 - Conversion : string(object?), number(object?), boolean(object?)
 - Ensemble de nœuds : position(), last(), count()
 - Traitement de chaînes de caractères : concat(), contains(), startswith()
 - Numériques : sum(), floor(), round()
 - Booléennes : not(), true(), false()

- L'expression prédicat accepte :
 - les opérateurs de comparaison : <, >, <=, >=, =, !=
 - les opérateurs logiques : not(..), and, or
 - Les opérateurs arithmétiques : +, -, *, div

Exemple Xpath avec prédictats

```
<annuaire type = 'pages jaunes '>
<entree>
    <nom> firstName1 LastNameA </nom>
    <tel>06 06 06 06 06</tel>
</entree>
<entree>
    <nom> firstName2 LastNameB </nom>
    <tel>07 07 07 07 07</tel>
</entree>
</annuaire>
```

Exemple	Description
/annuaire/entree[1]	Sélectionner le 1 ^{er} fils 'entree' de l'élément annuaire
/annuaire/entree[last()]	Sélectionner le dernier fils 'entree' de l'élément annuaire
/annuaire/entree[last()-1]	Sélectionner l'avant dernier fils 'entree' de l'élément annuaire
/annuaire/entree[position()<3]	Sélectionner le deux 1 ^{er} fils 'entree' de l'élément annuaire
/annuaire/entree[position()=3]	Sélectionner le 3 ^{ème} fils 'entree' de l'élément annuaire
/annuaire/entree[nom='myname']/tel	Sélectionner les fils 'tel' des éléments 'entree' dont le contenu du fils 'nom' est myname
/annuaire/entree[@attribut='value']	Sélectionner les fils 'entree' dont la valeur de l'attribut 'attribut' est 'value'
/annuaire//entree[nom or tel]	Sélectionner les descendants 'entree' qui au moins un fils 'nom' ou 'tel'

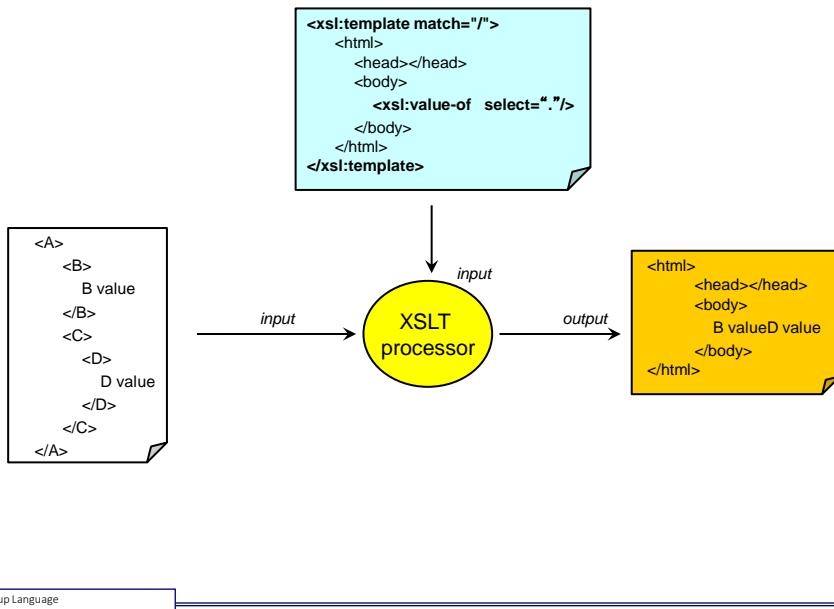
XML – eXtensible Markup Language

Retour à XSLT : utilisation d'XPath

Instruction	Description
<xsl:apply-templates select="XPATH"/>	Relance les règles du programme sur le sous-arbre pointé par le <i>select</i>
<xsl:value-of select="XPATH"/>	Génère le texte contenu dans le nœud ou attribut pointé par le <i>select</i>
<xsl:copy-of select="XPATH"/>	Génère le sous-arbre pointé par le <i>select</i>
<xsl:if test="XPATH">...</xsl:if>	Ne s'exécute que si <i>test</i> est vrai
<xsl:for-each select="XPATH">...</xsl:for-each>	S'exécute pour chaque sous-arbre renvoyé par le <i>select</i>

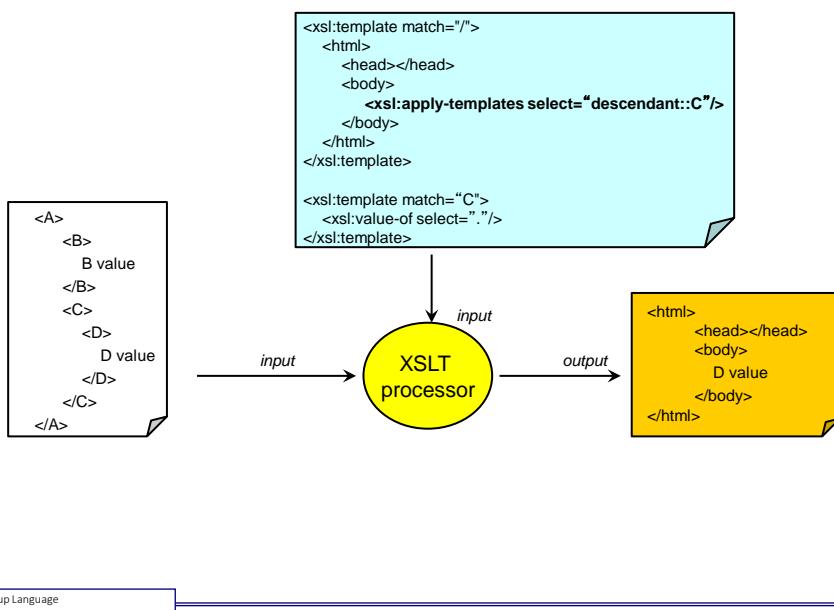
XML – eXtensible Markup Language

Value-of XSLT



57

Exemple apply-templates

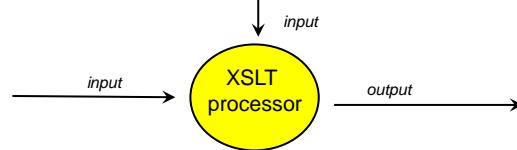


58

Règle de transformation XSLT

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="mysheet.xsl"?>
<writer>
<author>
<firstname> Carlos </firstname>
<lastname> Fuentes </lastname>
</author>
<author>
<firstname> Octavio </firstname>
<lastname> Paz </lastname>
</author>
<author>
<firstname> Juan </firstname>
<lastname> Rulfo </lastname>
</author>
<author>
<firstname> Amado </firstname>
<lastname> Nervo </lastname>
</author>
</writer>
```

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" version="4.0"/>
<xsl:template match="/">
<HTML>
<HEAD>
<TITLE> Mexican writers </TITLE>
</HEAD>
<BODY>
<UL>
<xsl:apply-templates/>
</UL>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```


RESULT

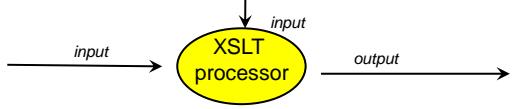
```
<BODY>
<HEAD>
<TITLE> Mexican writers </TITLE>
</HEAD>
<BODY>
<UL>
    <li> Carlos Fuentes Octavio Paz
        Juan Rulfo Amado Nervo
    </li>
</UL>
</BODY></HTML>
```

XML – eXtensible Markup Language

Templates répétés

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="mysheet.xsl"?>
<writer>
<author>
<firstname> Carlos </firstname>
<lastname> Fuentes </lastname>
</author>
<author>
<firstname> Octavio </firstname>
<lastname> Paz </lastname>
</author>
<author>
<firstname> Juan </firstname>
<lastname> Rulfo </lastname>
</author>
<author>
<firstname> Amado </firstname>
<lastname> Nervo </lastname>
</author>
</writer>
```

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" version="4.0"/>
<xsl:template match="/">
<HTML>
<HEAD>
<TITLE> Mexican writers </TITLE>
</HEAD>
<BODY>
<UL>
<xsl:apply-templates/>
</UL>
</BODY>
</HTML>
</xsl:template>
<xsl:template match="author">
<L>
<xsl:value-of select="lastname"/>,
<xsl:value-of select="firstname"/>
</L>
</xsl:template>
</xsl:stylesheet>
```


RESULT

```
<HTML>
<HEAD>
<TITLE> Mexican writers </TITLE>
</HEAD>
<BODY>
<UL>
    <li> Fuentes ,Carlos </li>
    <li> Paz ,Octavio </li>
    <li> Rulfo ,Juan </li>
    <li> Nervo ,Amado </li>
</UL>
</BODY></HTML>
```

XML – eXtensible Markup Language

Boucle explicite d'appel de templates XSLT

```
<?xml version="1.0"?>
<xmLstylesheet type="text/xsl" href="mysheet.xsl">
<writer>
<author>
<firstname> Carlos </firstname>
<lastname> Fuentes </lastname>
</author>
<author>
<firstname> Octavio </firstname>
<lastname> Paz </lastname>
</author>
<author>
<firstname> Juan </firstname>
<lastname> Rulfo </lastname>
</author>
<author>
<firstname> Amado </firstname>
<lastname> Nervo </lastname>
</author>
</writer>
```

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" version="4.0"/>
<xsl:template match="/">
<HTML>
<HEAD>
<TITLE> Mexican writers </TITLE>
</HEAD>
<BODY>
<UL>
<xsl:for-each select="//author">
<L>
<xsl:value-of select="lastname"/>,
<xsl:value-of select="firstname"/>
</L>
</xsl:for-each>
</UL>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```



RESULT

```
<HTML>
<HEAD>
<TITLE> Mexican writers </TITLE>
</HEAD>
<BODY>
<UL>
<L> Fuentes,Carlos </L>
<L> Paz,Octavio </L>
<L> Rulfo,Juan </L>
<L> Nervo,Amado </L>
</UL>
</BODY>
</HTML>
```

XML – eXtensible Markup Language

61

Utilisation de variables XSLT

```
<?xml version="1.0"?>
<xmLstylesheet type="text/xsl" href="mysheet.xsl">
<writer>
<author>
<firstname> Carlos </firstname>
<lastname> Fuentes </lastname>
</author>
<author>
<firstname> Octavio </firstname>
<lastname> Paz </lastname>
</author>
<author>
<firstname> Juan </firstname>
<lastname> Rulfo </lastname>
</author>
<author>
<firstname> Amado </firstname>
<lastname> Nervo </lastname>
</author>
</writer>
```

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:variable name="TITLE">
<xsl:text>Mexican writers</xsl:text>
</xsl:variable>
<xsl:template match="/">
<HTML>
<HEAD>
<TITLE> <xsl:value-of select="$TITLE"/> </TITLE>
</HEAD>
<BODY>
<H1> <xsl:value-of select="$TITLE"/> </H1>
<UL>
<xsl:for-each select="//author">
<L>
<xsl:value-of select="lastname"/>,
<xsl:value-of select="firstname"/>
</L>
</xsl:for-each>
</UL>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```



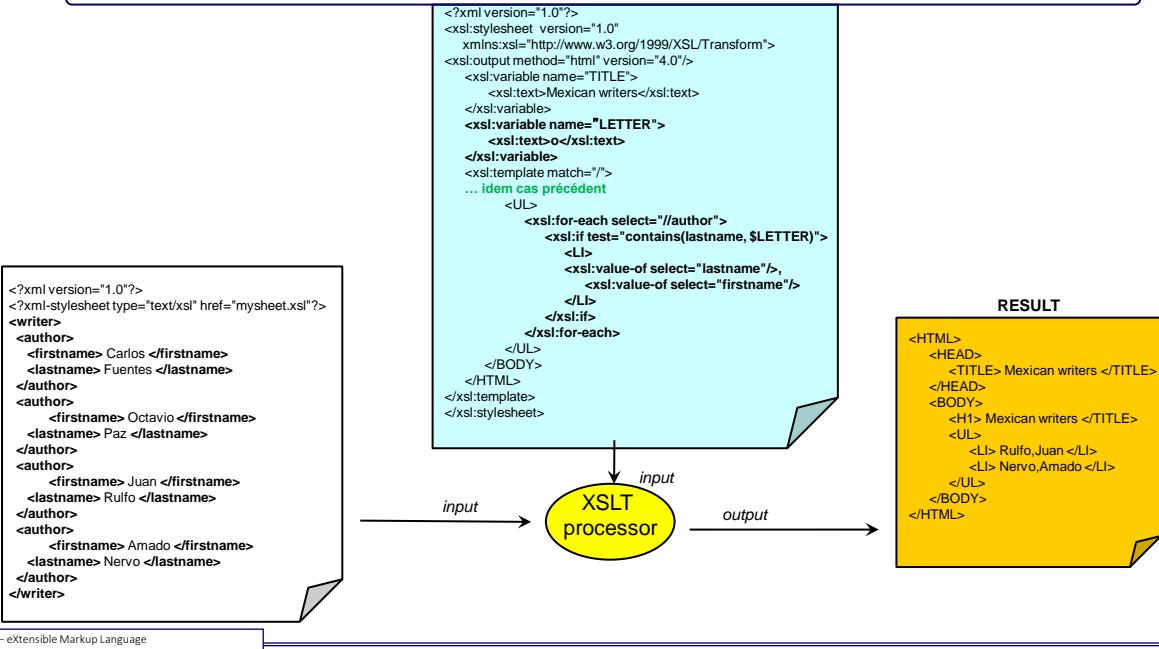
RESULT

```
<HTML>
<HEAD>
<TITLE> Mexican writers </TITLE>
</HEAD>
<BODY>
<H1> Mexican writers </H1>
<UL>
<L> Fuentes,Carlos </L>
<L> Paz,Octavio </L>
<L> Rulfo,Juan </L>
<L> Nervo,Amado </L>
</UL>
</BODY>
</HTML>
```

XML – eXtensible Markup Language

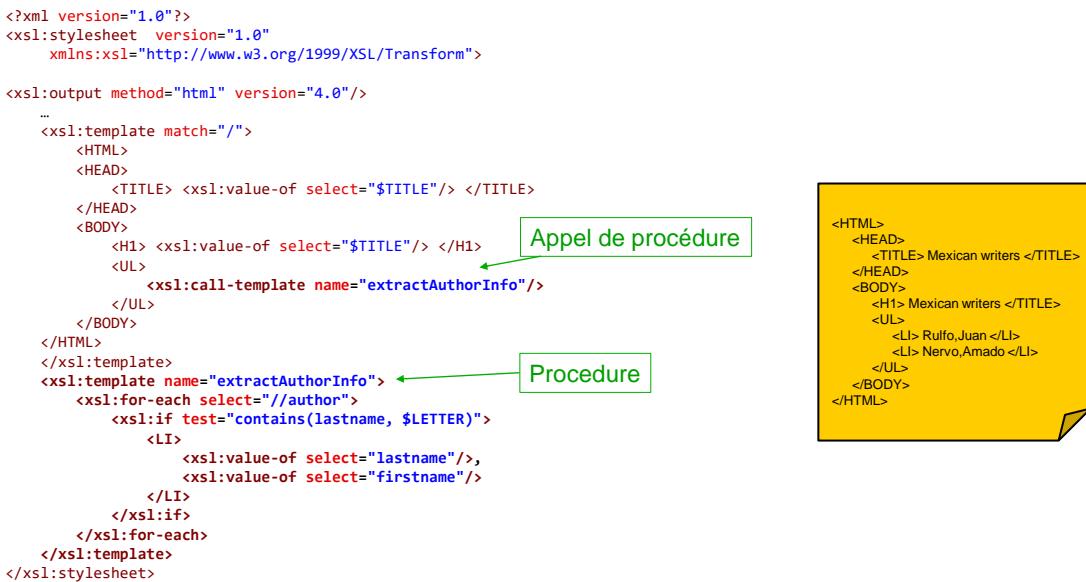
62

Conditionnelles XSLT



63

Procédure XSLT



64

Procédure avec paramètres XSLT

```

<xsl:stylesheet>
...
<xsl:template match="/">
<HTML>
...
<BODY>
    <H1> <xsl:value-of select="$TITLE"/> </H1>
    <UL>
        <xsl:call-template name="extractAuthorInfo">
            <xsl:with-param name="sep">***</xsl:with-param>
        </xsl:call-template>
    </UL>
</BODY>
</HTML>
</xsl:template>
<xsl:template name="extractAuthorInfo">
    <xsl:param name="sep"/>
    <xsl:for-each select="//author">
        <xsl:if test="contains(lastname, $LETTER)">
            <LI>
                <xsl:value-of select="lastname"/>
                <xsl:value-of select="$sep"/>
                <xsl:value-of select="firstname"/>
            </LI>
        </xsl:if>
    </xsl:for-each>
</xsl:template>
</xsl:stylesheet>

```

```

<HTML>
<HEAD>
    <TITLE> Mexican writers </TITLE>
</HEAD>
<BODY>
    <H1> Mexican writers </H1>
    <UL>
        <LI> Rulfo***Juan </LI>
        <LI> Nervo***Amado </LI>
    </UL>
</BODY>
</HTML>

```

XML – eXtensible Markup Language