De UML au langage C

© Olivier Caron

Polytech'Lille

Traduction du schéma conceptuel

√ Avant la phase de traduction :

- Toutes les données sont identifiées
- Les données sont structurées
- La notation est *indépendante* d'une implémentation

Traduction du schéma conceptuel

√ Avant la phase de traduction :
- Toutes les données sont identifiées
- Les données sont structurées
- La notation est *indépendante* d'une implémentation

√ Mise au point de la traduction :
- Des règles de traduction sont fixées
- Des systèmes automatisés peuvent exister (définition de profils).
- Pas de profil standard pour C :-((

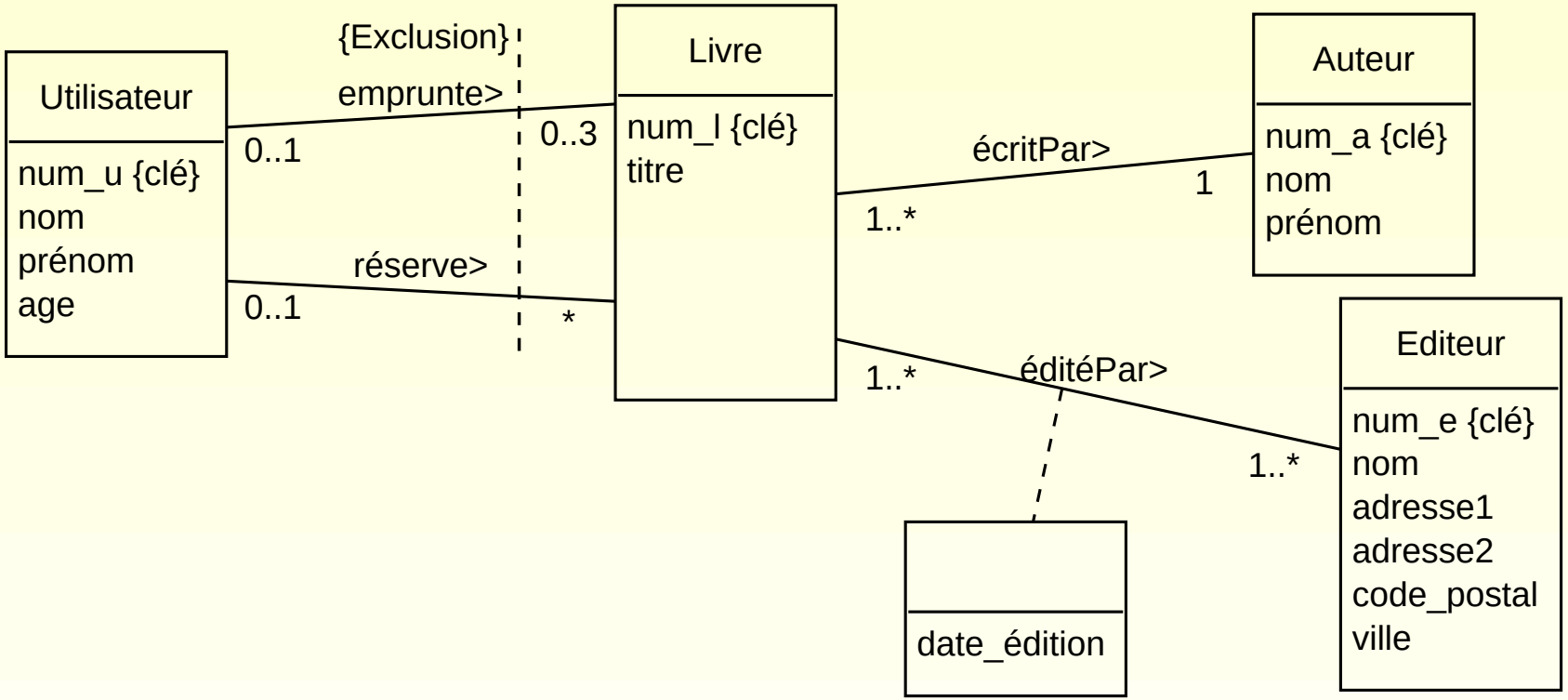
Traduction du schéma conceptuel

√ Avant la phase de traduction :
- Toutes les données sont identifiées
- Les données sont structurées
- La notation est *indépendante* d'une implémentation

√ Mise au point de la traduction :
- Des règles de traduction sont fixées
- Des systèmes automatisés peuvent exister (définition de profils).
- Pas de profil standard pour C :-((

√ Après la phase de traduction :
- Les données sont structurées dans un modèle compréhensible selon l'implantation

Un exemple fil rouge



Passage du schéma conceptuel au langage C : les entités

$\surd$ Une entité $\longrightarrow$ une structure

Passage du schéma conceptuel au langage C : les entités

- $\surd$ Une entité $\longrightarrow$ une structure
- $\surd$ Une propriété $\longrightarrow$ un champ d'une structure

Passage du schéma conceptuel au langage C : les entités

- √ Une entité $\longrightarrow$ une structure
- √ Une propriété $\longrightarrow$ un champ d'une structure

```
typedef struct auteur {
  char nom[MAX_CHAINE] ;
  char prenom[MAX_CHAINE] ;
} Auteur ;
```

- √ Une structure a sa propre identité (son adresse), les identifiants sont inutiles

Les associations

$\surd$ Une association $\longrightarrow$ des champs (pointeurs) d'une structure

Les associations

$\surd$ Une association $\longrightarrow$ des champs (pointeurs) d'une structure

- Une cardinalité 1 à une extrémité $\longrightarrow$ un champ de type pointeur de cette structure
- Une cardinalité * à une extrémité $\longrightarrow$ un champ de type pointeur de liste de cette structure
- Une cardinalite précise n (avec $n > 1$) un tableau de structures

Exemple Association 0..1

```
typedef Auteur * P_Auteur ;

typedef struct livre {
  char titre[MAX_CHAINE] ;
  P_Auteur auteur ;
}
```

√ Les noms de rôles permettent de préciser le nom de la propriété

√ Des pointeurs dans chaque structure, sauf si navigation UML.

Exemple Association *

```
typedef Editeur  * P_Editeur ;

typedef struct cellule_editeur {
  P_Editeur unEditeur ;
  struct cellule_editeur *suiv ;
} Cellule_Editeur ;

typedef Cellule_editeur *Editeurs ;

typedef struct livre {
  char titre[MAX_CHAINE] ;
  P_Auteur auteur ;
  Editeurs les_editeurs ;
} ;
```

Les associations avec propriétés

√ Si cardinalité 1 à une des extrémités

$\longrightarrow$ ajout propriété asso dans une structure composante de l'association

Les associations avec propriétés

√ Si cardinalité 1 à une des extrémités

$\longrightarrow$ ajout propriété asso dans une structure composante de l'association

√ Ajout structure avec pointeurs vers les structures de l'association

```
typedef struct reserve {
  char date_reservation[MAX_CHAINE] ;
  P_Utilisateur util ;
  P_Livre livre ;
} Reserve ;
```

Manipulation des associations

- √ Toujours 3 fonctions liées à une association
  - ajout d'un lien (add)
  - suppression d'un lien (remove)
  - consultation d'un lien (get)
- √ La signature des fonctions dépend de la cardinalité
- √ Selon la navigation, un ou deux couples des 3 fonctions

Exemple de gestion de manipulation des associations

√ cardinalité 1 :

```
public void addEcritPar (P_Livre livre, P_Auteur auteur) ;
public P_Auteur getEcritPar(P_livre livre) ;
public void removeEcritPar(P_livre livre) ;
```

Exemple de gestion de manipulation des associations

√ cardinalité 1 :

```
public void addEcritPar (P_Livre livre, P_Auteur auteur) ;
public P_Auteur getEcritPar(P_livre livre) ;
public void removeEcritPar(P_livre livre) ;
```

√ cardinalité * :

```
public void addEditePar (P_Livre livre, P_Auteur auteur) ;
public Editeurs getEditePar(P_livre livre) ;
public void removeEcritPar(P_livre livre, P_Editeur editeur) ;
```

√ Utilisation du nom de l'association, ou du noms des rôles

Les extensions

√ Une extension d'une entité : ce sont toutes les instances (occurrences) d'une entité

Les extensions

- √ Une extension d'une entité : ce sont toutes les instances (occurrences) d'une entité
- √ Pointeur de liste chaînée

Les extensions

- √ Une extension d'une entité : ce sont toutes les instances (occurrences) d'une entité
- √ Pointeur de liste chaînée
- √ Mais pas de langage de requêtes en C, tout est à programmer !

Un profil UML pour langages procéduraux

√ Imposer les noms de rôles aux associations

Un profil UML pour langages procéduraux

- √ Imposer les noms de rôles aux associations
- √ Imposer les noms d'associations lorsque pas de navigation

Un profil UML pour langages procéduraux

- √ Imposer les noms de rôles aux associations
- √ Imposer les noms d'associations lorsque pas de navigation
- √ Clés inutiles

Un profil UML pour langages procéduraux

- √ Imposer les noms de rôles aux associations
- √ Imposer les noms d'associations lorsque pas de navigation
- √ Clés inutiles
- √ Un processus complet de génération de code est alors possible !

Un profil UML pour langages procéduraux

- √ Imposer les noms de rôles aux associations
- √ Imposer les noms d'associations lorsque pas de navigation
- √ Clés inutiles
- √ Un processus complet de génération de code est alors possible !