OFL : un modèle pour paramétrer la sémantique opérationnelle des langages à objets

Application aux relations inter-classes

Pierre Crescenzo

jeudi 20 décembre 2001, Sophia-Antipolis

Plan

- Introduction
  - Problématique
  - La solution envisagée
- Le modèle OFL
  - Définition
  - Exemples d'applications
  - Résultats
- Bilan et Perspectives

Contexte

- Langages à objets avec classes
  - Java
  - Eiffel
  - C++
- Génie Logiciel
  - Contrôles prépondérants
  - Approche pragmatique

Définition du sujet

- Inclure un système de vues dans les langages à objets
- ↪ Les vues, une dérivation de l'héritage
- ↪ Séparer la gestion des vues des autres applications de l'héritage
- ↪ Créer une relation de vue indépendante
- ↪ Définir un modèle métaobjet permettant de créer la relation de vue, et d'autres...

Les objectifs

- Améliorer le code en spécifiant plus précisément les usages des relations
  - pour plus de lisibilité
  - pour plus de contrôles
  - pour une documentation plus adéquate
- Offrir cette spécification sans l'imposer au programmeur
- Pouvoir adapter un langage de programmation dans ce but

Approche : métaprogrammation

- Créer de nouveaux types de relation
- Créer de nouveaux types de classe
- Pouvoir les intégrer à un langage
  - pour créer un nouveau langage ou,
  - plus probablement, pour adapter un langage existant

Métaprogrammation : principes d'OFL

- → Réifier les relations autant que les classes
- → En faire des *éléments de langage* intégrables
- → Offrir un moyen simple de créer ces composants et limiter l'écriture d'un code complexe pour les mettre en œuvre
- → Si l'accès au code est nécessaire, localiser les interventions

OFL : *Open Flexible Languages*

- Trois éléments essentiels
  - Composant-description
    - ≈ métaclasse paramétrée
  - Composant-relation
    - ≈ métarelation paramétrée
  - Composant-langage
    - ≈ métalangage, composition

Hypergénéricité : paramètres

- Chaque composant est notamment décrit à l'aide d'un ensemble de paramètres.
- Paramètre ≈ partie de la sémantique
- Création d'un composant par spécialisation d'un composant générique
- Responsabilité du métaprogrammeur
- Contrôle assertionnel des composants

Hypergénéricité : actions

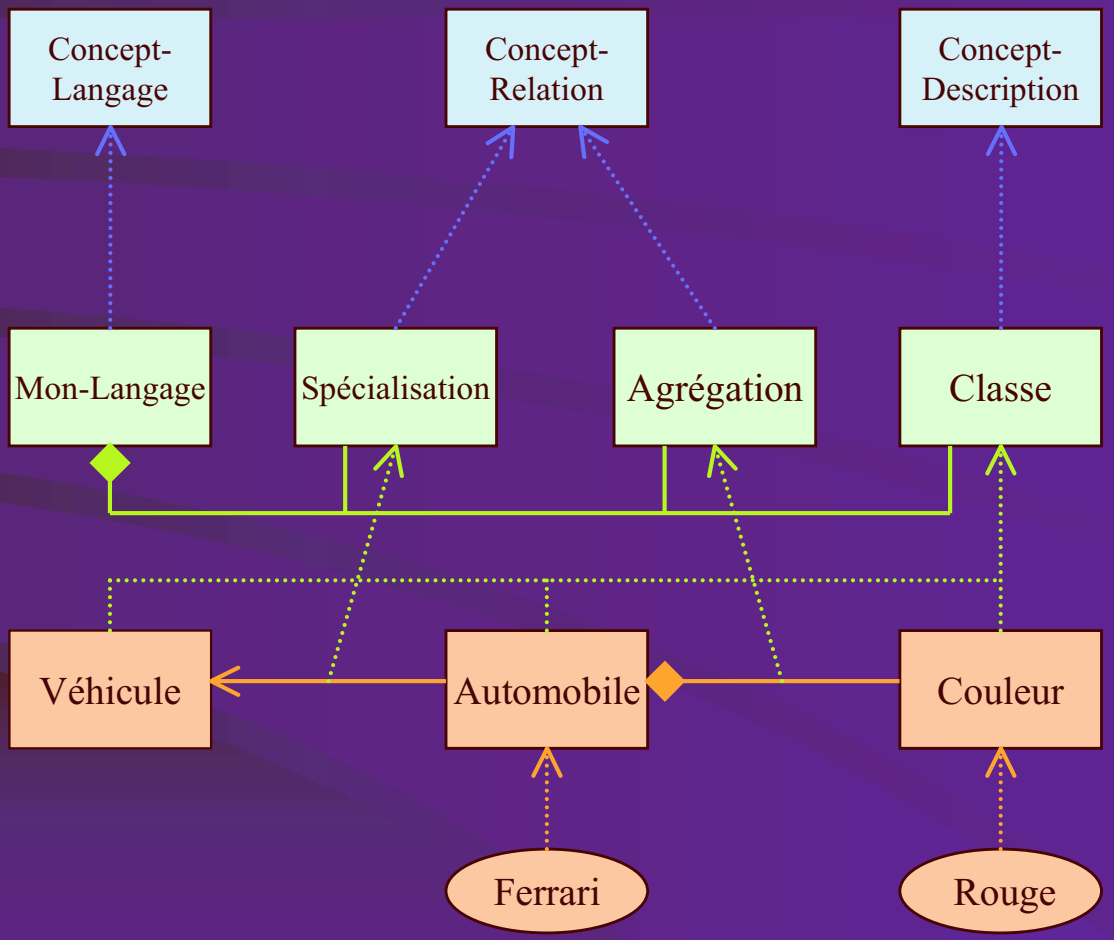
- ➡ Il faut prendre en compte la définition d'un composant, avec ses paramètres valués, pour décrire la partie opérationnelle du système.
- OFL dispose d'un système de 53 *actions*.
- Action ≈ algorithme définissant une partie de l'exécution d'une application
- Chaque action prend en compte la valeur des paramètres qui influencent son exécution.
- Exemples d'actions :
  - envoi d'un message
  - recherche d'une primitive (liaison dynamique)
  - affectation d'une instance ou d'une valeur à un attribut

Architecture d'OFL

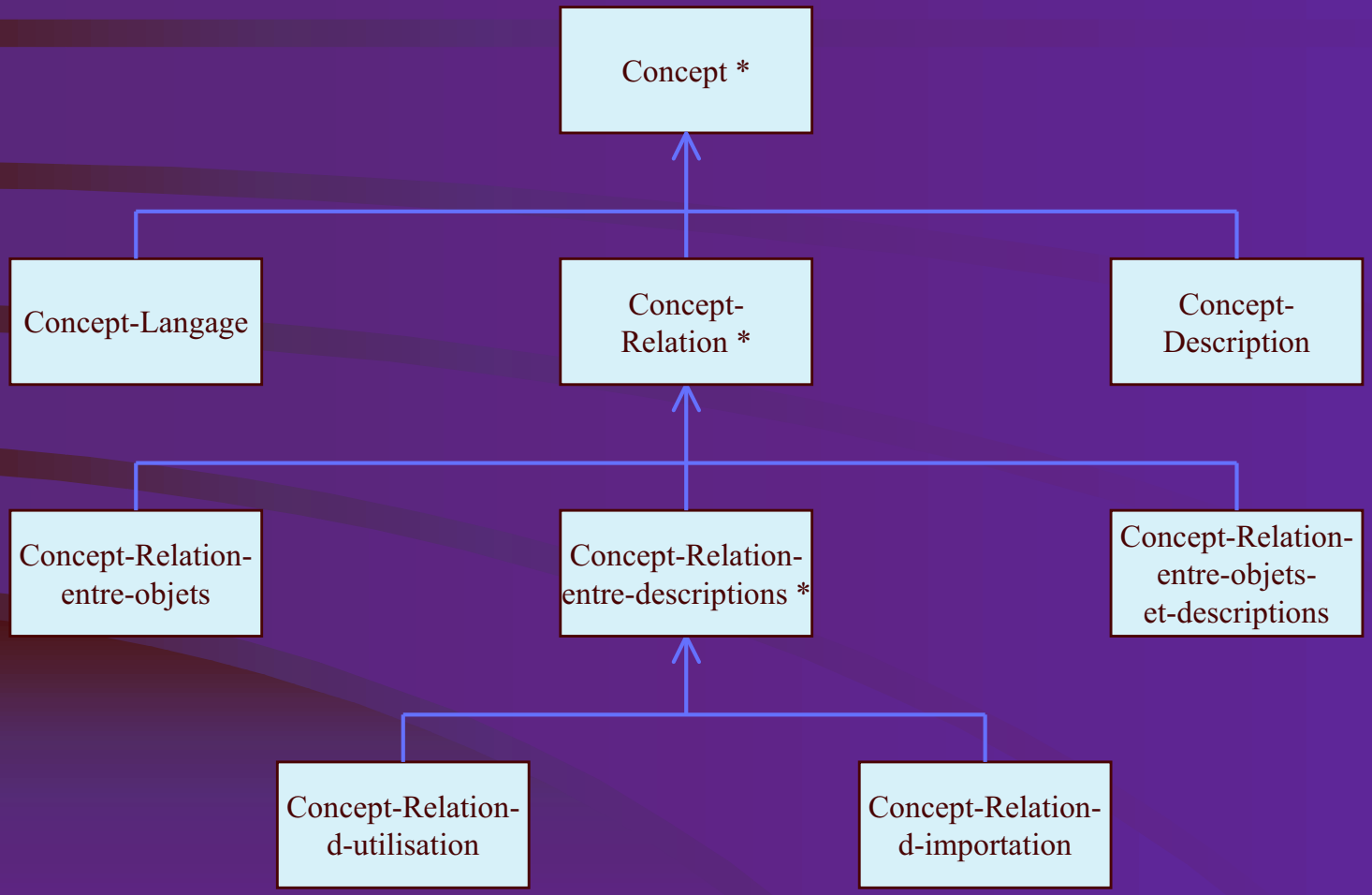
des concepts génériques

des composants de langage

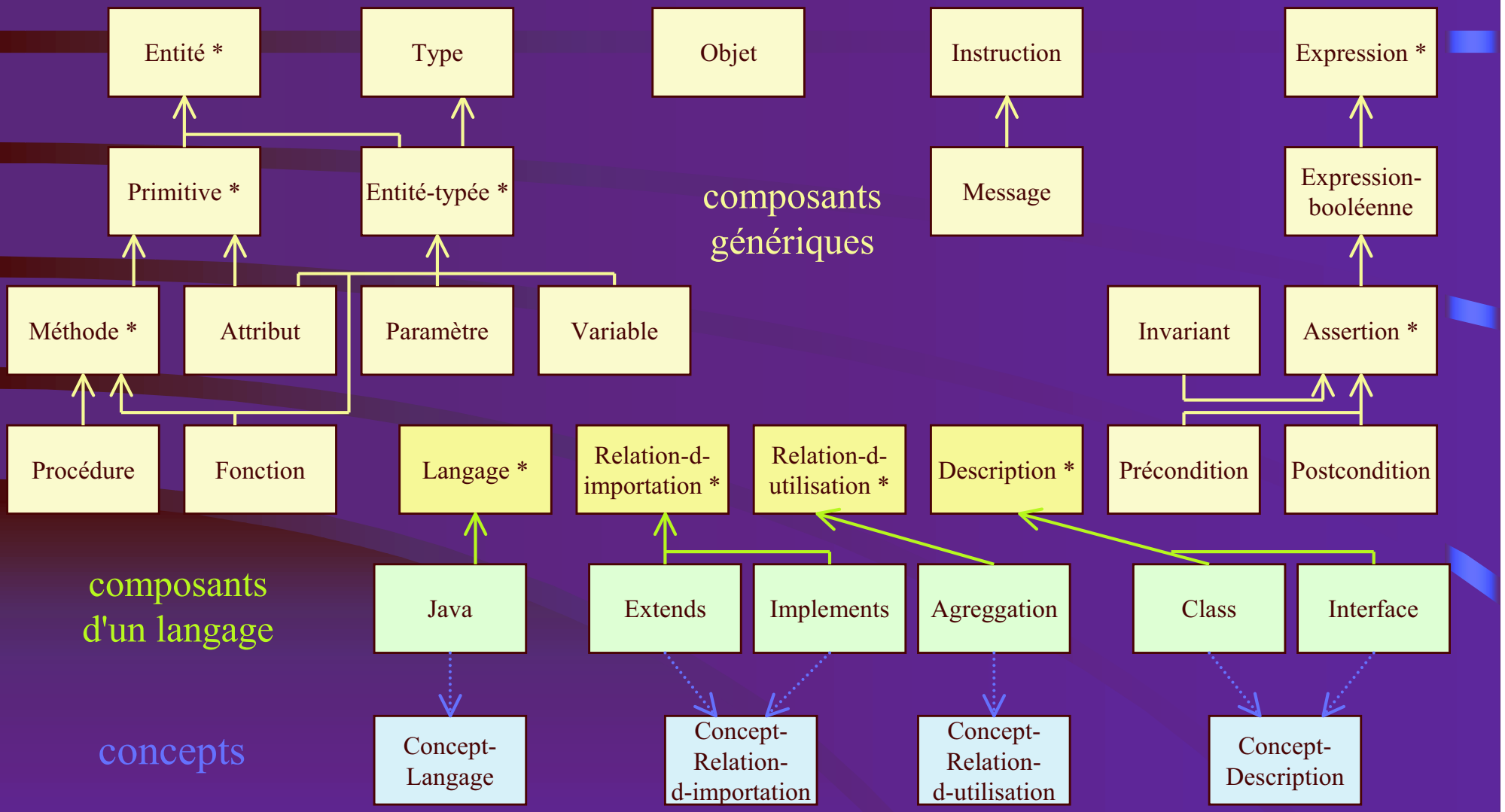
une application



Hiérarchie des concepts



Hiérarchie partielle des composants de base



Quelques-uns des 13 paramètres de concept-description

- Name ("Class", "Interface" ou ...)
- Generator (true ou false)
- Sharing_control (description, instance ou unique_instance)
- Attribute (allowed ou forbidden)
- Method (allowed ou forbidden)
- Overloading (allowed ou forbidden)
  - attributs,
  - résultats de fonction,
  - nombre de paramètres de méthode et
  - type de paramètres de méthode

Quelques-uns des 28 paramètres de concept-relation

- Name ("Inheritance", "Code_Reuse", "View", ...)
- Cardinality (1-1, 1-∞, ou ...)
- Repetition (allowed ou forbidden pour les sources et les cibles)
- Circularity (allowed ou forbidden)
- Opposite ("Generalisation", "View", ...)
- Polymorphism_direction (none, up, down ou both)
- Polymorphism (hiding ou overriding pour les attributs et les méthodes)
- Feature_variance (covariant, contravariant, nonvariant ou non_applicable pour les paramètres de méthode, les résultats de fonction et les attributs)
- Renaming (mandatory, allowed ou forbidden)

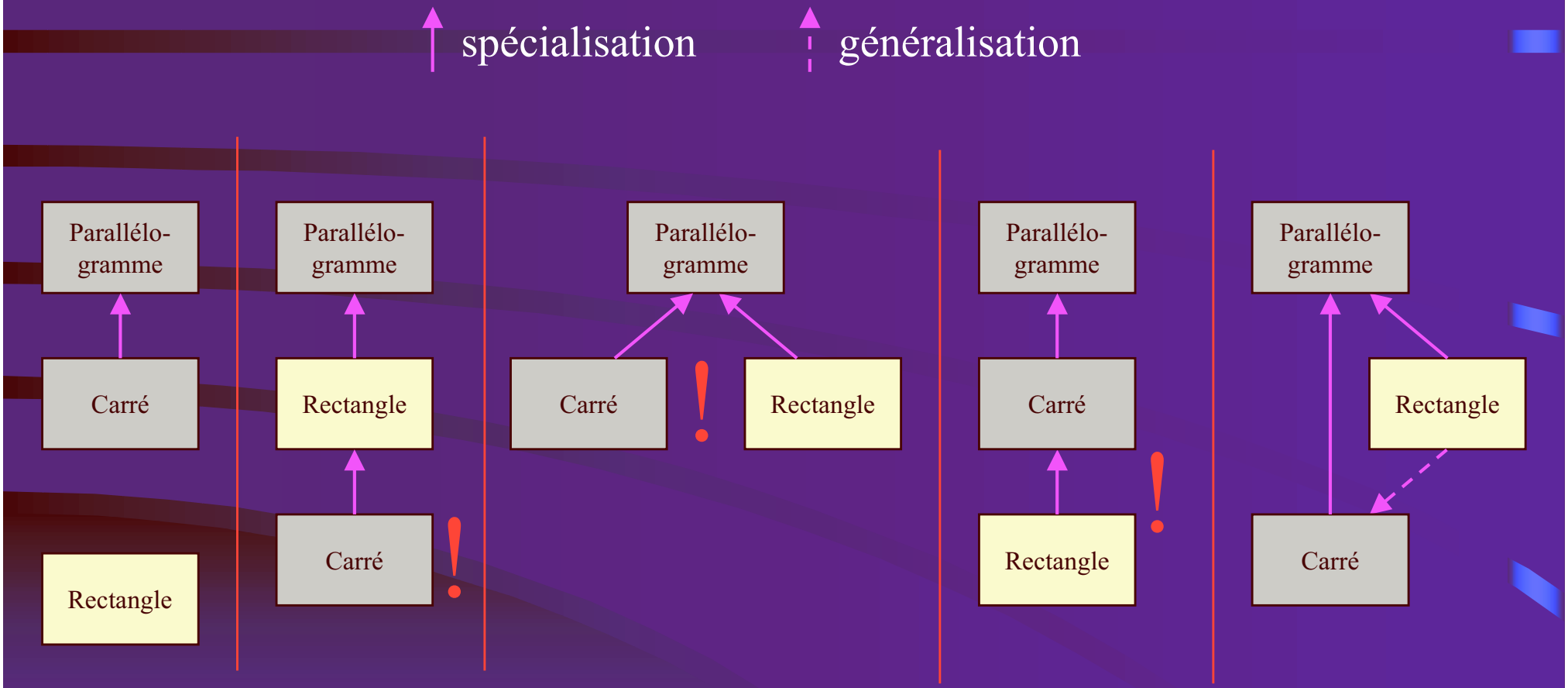
Une action parmi 53

- Feature lookup(M : Message)
  - Exemples de paramètres de concept-description pris en compte :
    - Generator
    - Overloading
  - Exemples de paramètres de concept-relation pris en compte :
    - Cardinality
    - Circularity
    - Opposite
    - Polymorphism_direction
    - Polymorphism
    - Feature_variance
    - Renaming

OFL : principes d'utilisation

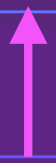
1. Création des composants-descriptions
   a. Instanciation du concept-description et spécialisation d'un composant-description
   b. Valuation des paramètres pour indiquer la sémantique
2. Création des composants-relations
   a. Instanciation d'un concept-relation et spécialisation d'un composant-relation
   b. Valuation des paramètres pour indiquer la sémantique
3. Création du composant-langage
   a. Instanciation du concept-langage et spécialisation d'un composant-langage
   b. Intégration des composants-descriptions et composants-relations

Pourquoi la Généralisation en plus de la Spécialisation ?



Réutiliser du code

```
class Personne {

    int annéeNaissance;

    int âge() {
        return Date.annéeCourante() - annéeNaissance;
    }
}
```

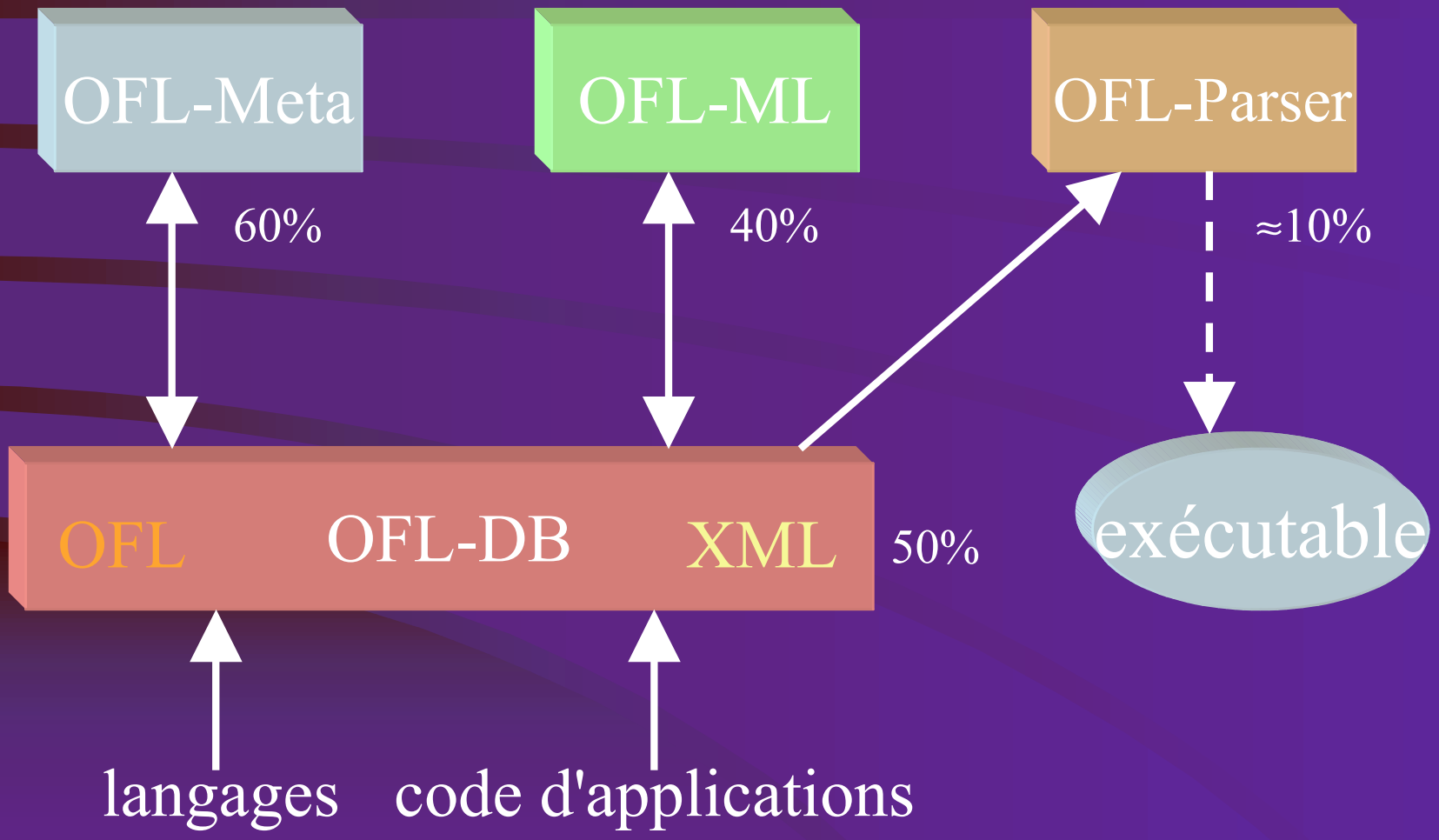


```
class Automobile code_reuses Personne {

    int annéeMiseCirculation;

    int âge() from Personne where annéeNaissance is
            annéeMiseCirculation;
}
```

OFL : les points clés

- La *relation entre description* et non la *description* au centre du modèle
- Paramétrage hypergénérique
- Système d'actions redéfinissables
- Contrôles assertionnels de cohérence du modèle et de ses usages
- Métaprogrammation et programmation différenciée
- Pas de langage support choisi *a priori*
- Langage décrit sous forme d'éléments intégrables
- Approche non réflexive

État de l'implémentation



Bilan

- Possibilité de spécification plus fine des relations interdescriptions
- Fonctionnalités supplémentaires offertes au programmeur, non imposées
- Capacité d'adaptation d'un langage

notre réponse : OFL

Perspectives

- Développement et diffusion des outils logiciels
  - Ajout d'un type de relation à un langage standard
  - Spécialisation des actions vers un problème ciblé
- Formalisation de la définition du modèle
- Étude et usage du modèle dans un contexte de *séparation des préoccupations*
- Système d'intégration de services (persistance, concurrence, évolution, …)
- Usage d'OFL comme plate-forme d'expérimentation d'adaptation et évolution des langages et applications