Au sommaire

- √ Les spécificités BD par rapport à la programmation
  - ▶ Les apports du modèle ODL de l'ODMG par rapport au modèle UML de l'OMG
  - ▶ Le langage de requêtes OQL.
- √ Les réponses de l'OMG.
  - ▶ Introduction de services standards CORBA
  - ▶ Les besoins non résolus (les vues).
- √ La fédération de systèmes d'information
  - ▶ Description de l'architecture
  - ▶ CORBA et la fédération

Bases de données à Objets dans un Environnement Distribué

Olivier.Caron@lifl.fr

Point de départ

√ **Nécessité d'établir une norme.**

Pas de norme ⇒ frein à l'expansion des BDOO.

√ **Difficulté d'établir une norme.**

Il existe déjà des gestionnaires de bases de données orientées objet
⇒ Il faut prendre le meilleur de chaque implémentation.

Nécessité de s'accorder avec des normes qui existent déjà (SQL, OMG, langages OO ...). ODMG est affiliée à OMG.

√ **Les membres du groupe ODMG :** Ils représentent presque toute l'industrie des bases de données à objets (SunSoft, Sybase, Bell, HP-Versant, Object Design, O_2, POET Software, GemStone, Objectivity, ONTOS . . .)

OBJECT DATABASE MANAGEMENT GROUP — Objects, Languages, Databases

√ ODMG a été fondé en 1991 par Rick Cattell (SunSoft/JavaSoft) et des fabricants de bases de données objets.

√ Versions 1.0 1993, 1.1 1994, 1.2 1995, 2.0 1997.

√ Les plus :

- Bonne définition des besoins BD (version 1.0)
- Intégré par les éditeurs **SGBDOO**

√ Les moins :

- trop orienté Java ? (version 2.0)
- absence des ténors SGBD (Oracle,. . .)
- Cohérence de la norme ?, évolution de la norme ? liens OMG ?

Qu'est-ce qu'un système de gestion de BDOO ?

En anglais OODBMS.

C'est un langage orienté objet + une base de donnée.

Intégration "transparente" (voir schéma)

- √ Les données sont des objets.
- √ Le DML est intégré dans le langage de programmation orienté objet.
- √ Persistance des objets.
- √ Contrôle de la concurrence des accès aux objets.
- √ Recouvrement des données.
- √ Langage de requête.
- √ ...

Buts

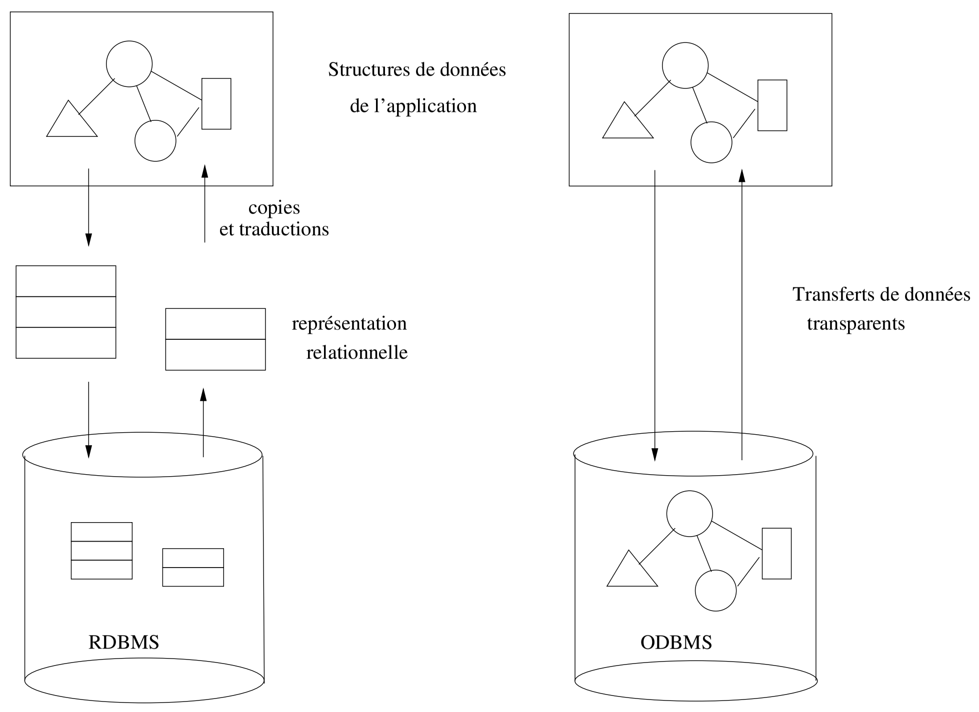
Décrire un ensemble de standards pour permettre aux développeurs d'applications bases de données objets d'écrire des applications portables (sur d'autres systèmes de gestion de BDOO).

- √ schéma de données
- √ interfaçage avec des langages de programmation
- √ langage de manipulation des données (DML)
- √ langage de requêtes

A plus long terme : interopérabilité des bases de données, communication entre bases distribuées hétérogènes à travers l'OMG ORB.

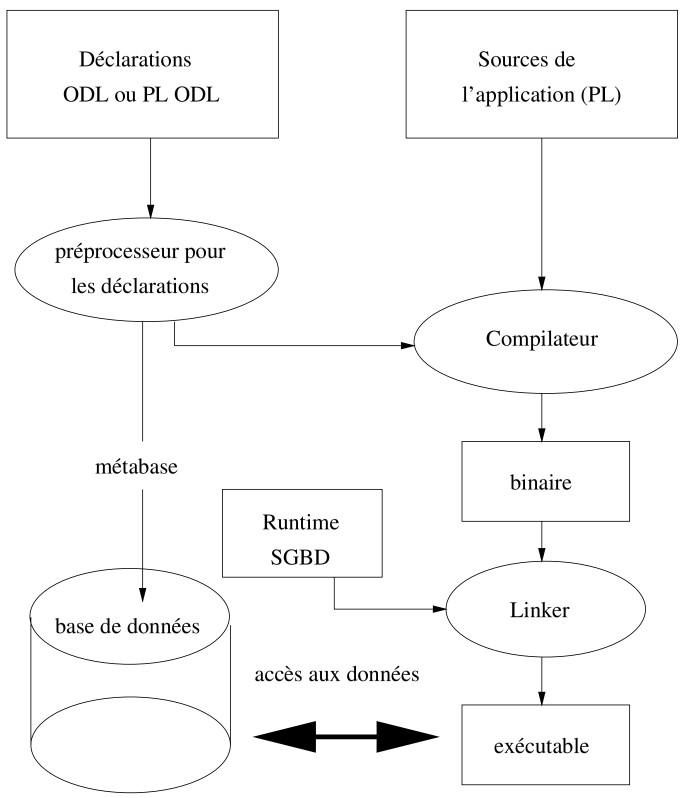
Que contient la norme ODMG ?

- √ **Le modèle objet**
- √ **Langage de définition des objets (ODL).** inspiré de l'OMG IDL.
- √ **Langage de requête OQL.**
  inspiré de SQL. Dans l'avenir SQL et OQL doivent converger.
- √ **Liens avec C++.**
  Comment écrire du code C++ portable qui manipule les objets persistants (C++ OML).
  Version de l'ODL qui utilise la syntaxe C++.
  Comment appeler des requêtes OQL dans du code C++.
  Opérations sur la base . . . et les transactions.
- √ **Liens avec Smalltalk.**
  Reprise de ce qui existe dans la norme OMG pour le passage IDL/Smalltalk.
  Même chose que pour C++ : opérations propres aux bases de données.
- √ **Liens avec Java (incomplet)**



Le Modèle Objet

- √ Les primitives de base sont **les littéraux** et **les objets**. Chaque objet a un identifiant unique (object_ID). Les littéraux n'ont pas d'identifiant.
- √ L'état d'un objet est défini par un ensemble de **propriétés** qui sont soit des **attributs** soit des **relations**.
- √ Le comportement d'un objet est défini par un ensemble d'**opérations**
- √ Les objets et les littéraux sont classés selon leur **type**. Deux éléments ont le même type s'ils ont même comportement et mêmes propriétés.
- √ une **base de données** stocke des objets qui peuvent être partagés par plusieurs utilisateurs et plusieurs applications. Une base de données est basée sur un **schéma** défini en ODL et contient les instances des types définis dans ce schéma.



Héritage

- relation type – sous-type : relation ISA.

```
interface Employee{...};
interface Professor : Employee {...};
interface Associate_Professor : Professor {...};
```

- L'interface du sous-type peut définir des caractéristiques supplémentaires par rapport au super-type. On peut aussi avoir des opérations identiques avec des comportements différents.
- Le modèle supporte l'héritage multiple mais ne spécifie pas comment les conflits sont résolus (à définir dans l'implémentation).
- Certains types ne sont pas instanciables : les **types abstraits** ; par opposition aux types **concrets**. Cependant, l'ODL ne spécifie pas si un type est concret ou abstrait.

Interfaces et implémentations

- Un type a une spécification d'**interface** et une ou plusieurs spécifications d'**implémentation**.
- L'interface définit les caractéristiques externes des objets du type. C'est-à-dire les opérations que l'on peut invoquer et les variables d'état qui sont accessibles (les propriétés). Elle est définie en ODL.
- Une spécification d'implémentation consiste en un ensemble de méthodes et une représentation (SD). Le contenu de l'implémentation n'est pas visible des utilisateurs de l'objet.

Les Objets

- √ Tous les objets possèdent un identifiant appelé **object_id** qui permet de les distinguer dans le domaine de stockage. Un objet change de valeur durant sa vie mais jamais d'object_id.
- √ Un objet peut aussi être désigné par un (ou plusieurs) **nom(s)**.
  ODMG veut surtout utiliser les noms comme des **racines**, des points d'entrée à la base.
- √ **durée de vie de l'objet**
  Elle est spécifiée à la création de l'objet. Elle est de deux sortes : **transient (éphémère)** et **persistent (persistant)**. Un type peut avoir des instances persistantes (dans la base) et des instances éphémères (variables de l'application).
  ⇒ Souplesse par rapport au modèle relationnel.

Extensions, clefs

- √ On appelle **extension** d'un type l'ensemble des instances de ce type.
- √ Dans le modèle relationnel : une extension pour chaque table.
  Dans le modèle objet : le concepteur décide s'il faut gérer ou non une extension pour tel ou tel type.
  Il peut aussi définir un index pour une extension.
- √ On peut définir des **clefs** pour un type.
  Un type doit avoir une extension pour pouvoir posséder une clef. La valeur de la clef de chaque objet est alors unique dans l'extension du type.

√ **Les relations**

On ne considère que des **relations binaires**. Ces relations peuvent être `1-1`, `1-n` ou `n-n`.

Les relations dans le modèle objet sont similaires à celles rencontrées dans le modèle entités-relations. Elles sont représentées par des chemins transversaux ($\neq$ pointeurs).

```
interface Professor{ ...
  relationship Set<Course> teaches
    inverse Course::is_taught_by;  ...
}
interface Course{
  ...
  relationship Professor is_taught_by
    inverse Professor::teaches;
  ...
}
```

L'ODBMS doit maintenir l'intégrité référentielle des relations.

Comment modéliser l'état : les propriétés

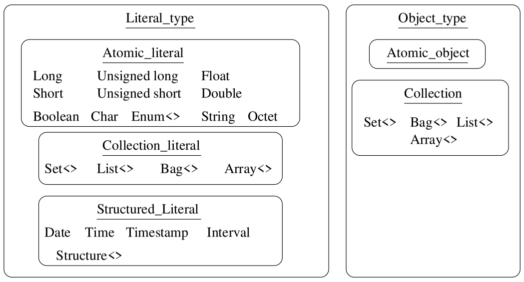
√ **Les attributs**

Ils définissent l'état abstrait d'un type.

```
interface Person{
  attribute Short age;
  attribute String name;
  attribute Enum gender {male,female};
  attribute Address home_address;
  attribute Set<Phone-no> phones;
  attribute Department dept;
};
```

Les méta-données

- √ Informations sur les objets de la base : le schéma, comment accéder aux données ...
- √ Hiérarchie de types prédéfinis.



- √ Règles de compatibilité de types. La compatibilité suit la relation de sous-type.
- √ Valeur nulle : pour tout type litéral, il existe un autre type supportant la valeur `nil`.
- √ type `table` par analogie au modèle relationnel (synonyme du type `Bag<Struct<>>`).

Comment modéliser le comportement : les opérations

- √ On définit un ensemble de **signatures d'opérations** :
  nom de l'opération,
  type des arguments,
  type de la valeur retournée,
  noms des exceptions déclenchées.
- √ Même spécification que OMG
- √ Surcharge.
- √ Exceptions.
- √ Pas de spécification de la sémantique des opérations : pré et post conditions, invariants.

Opération sur les transactions

```
interface Transaction{
  void begin();
  void commit();
  void abort();
  void checkpoint();
};
```

Opération sur la base

```
interface Database{
  void open(in string database_name);
  void close();
  void bind(in any an_object, in string name);
  any lookup(in string object_name);
};
```

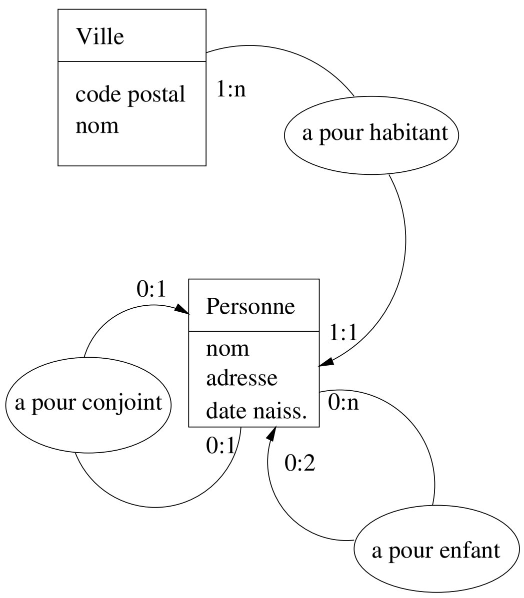

Le Modèle de Transactions

√ Un programme qui utilise des objets persistants est organisé en **transactions** ⇒ garantie des propriétés ACID.

- **Atomicité :** une transaction se fait entièrement ou pas du tout.
- **Consistance :** une transaction fait passer la base d'un état consistant à un autre état consistant.
- **Isolement :** l'exécution de transaction "en parallèle" donne le même résultat que l'exécution de ces transactions en séquence.
- **Durabilité :** Les effets d'une transaction terminée correctement sont conservés (même en cas de panne, problème mémoire, tremblement de terre ...)

√ Le contrôle de la concurrence se fait par des méthodes classiques de verrouillage. (Verrou sur un objet)

Un exemple



Object Definition Language

- √ C'est un langage de spécification qui permet de définir les interfaces des types.
  - √ Il permet la portabilité des schémas.
  - √ Il facilite l'intégration de schémas de sources multiples.
- √ Ce n'est pas un langage de programmation.
- √ ODL est indépendant des langages de programmation.
- √ Il est compatible avec l'OMG IDL.

OQL :Object Query Language

OQL est un langage déclaratif d'accès aux objets, et utilise le modèle objet décrit précédemment.

- √ C'est un sur-ensemble de SQL (uniquement la partie requêtes)
  - ▹ Ce n'est pas un langage complet
  - ▹ OQL est un langage fonctionnel où l'on peut composer les opérateurs.
- √ Il fournit des primitives pour manipuler des ensembles d'objets (`Set<>`) mais aussi tous les autres constructeurs.
- √ OQL peut être appelé à l'intérieur d'un programme C++, smalltalk... et réciproquement, OQL peut appeler des opérations écrites dans ces langages.
- √ Il n'y a pas d'opérateurs explicites pour les mises-à-jour (utiliser les méthodes).


```
interface Personne
( extent les_gens)
{ attribute String nom;
  attribute Struct Adresse {
     Unsigned Short numero,
     String rue,
     String nom_ville} adresse;
  relationship Personne epoux
     inverse Personne::epoux;
  attribute Date date_naiss ;
  relationship Set<Personne> enfants
     inverse Personne::parents
     {order_by date_naiss} ;
  relationship List<Personne> parents
     inverse Personne::enfants;
};
  void naissance (in String nom) ;
```

```
  Boolean mariage
    (in String nom_personne)
    raises (personne_inconnue);
  Unsigned Short ancetres
    (out Set<Personne> les_ancetres)
    raises (personne_inconnue);
  void demenagement
    (in String nouvelle_adresse);
};

interface Ville
( extent les_villes
  key code_postal)
{ attribute Unsigned Short code_postal;
  attribute String nom;
  attribute Set<Personne> population;
};
```

OQL Navigation

```
select c.address
from Persons p,
     p.children c
```

Remarque : pour extraire une propriété d'un objet on utilise indifféremment le `‘‘.’’` ou la `‘‘->’’`.

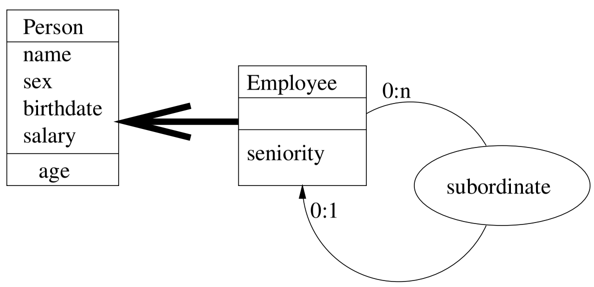
OQL :Prédicats

```
select c.address
from Persons p
     p.children c
where p.address.street = ‘‘Penny Lane’’
      and count(p.children)>=2
      and c.address.city != p.address.city
```


OQL : les noms

OQL permet d'interroger la base à partir des objets nommés. Un nom peut désigner n'importe quel type d'objets (atomique, collection ...).

Exemple :



noms : `Persons` (extension de la classe `Person`), `Chairman` (employé particulier).

Les noms sont aussi des requêtes.

```
select distinct x.age
from Persons x
where x.name = ‘‘Pat’’
```

OQL : polymorphisme

L'extension `Persons` comporte des objets de trois classes : `Person`, `Employee`, `Student`.

La méthode `activities` a 3 incarnations : une pour chaque classe.

```
select p.activities
from Persons p
where p.name=''Paul''
```

Dans l'exemple suivant, on s'intéresse aux objets de la classe `Student` dans l'extension `Persons` :

```
select ((Student)p).grade
from Persons p
where ``course of study'' in p.activities
```


OQL :Jointure

```
select p
from Persons p
     Flowers f
where p.name = f.name
```

OQL :appel de méthode

```
select max(select c.age from p.children c)
from Persons p
where p.name = ``Paul''

select p.oldest_child.address
from Persons p
where p.lives_in(``Paris'')
```

OQL : group by

```
select department,
   avg_salary: avg(select x.e.salary from partition x)
from Employees e
group by department: e.deptno
having avg(select x.e.salary from partition x)> 30000
```

Le type du résultat est :

`bag<struct(department: integer, avg_salary: float)>`

12 | 30501.6	5 | 30045.2	...	23 | 30950.5

OQL : quantification

```
for all x in Students: x.student_id > 0

exists x in Doe.takes: x.taught_by.name = ''Turing''
```

Remarque : on peut utiliser la syntaxe SQL pour les requêtes imbriquées (`where exists (select ...)`)

OQL : expressions ensemblistes

Opérateurs `union`, `intersect`, `except`.

Définis pour les types `Set` et `Bag`.

OQL : order by

On peut trier la collection résultat d'une requête selon différents attributs, par ordre croissant ou décroissant.

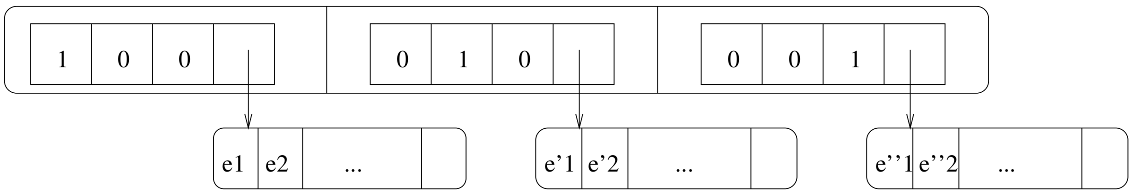
```
select p from Persons p
order by p.age, p.name

select * from Persons
order by age desc, name asc, department
```



```
select *
from Employees e
group by low:    salary < 1000,
         medium: salary >= 1000 and salary < 10000
         high:   salary >= 10000
```

Résultat de type

```
set<struct(low:boolean,
           medium:boolean,
           high:boolean,
           partition: bag<struct(e:Employee)>)>
```



Liaison avec les langages orientés objets

- √ Liens avec ODL, OML, OQL.
- √ Preprocesseur nécessaire pour passer d'une description en ODL vers le langage cible (C++, Smalltalk, Java...).
- √ Un seul système de typage. Instances persistantes ou éphémères.
- √ Dans tous les cas, on respecte la syntaxe et la sémantique des langages : ils n'ont pas à être modifiés pour implémenter ces liens.
- √ On peut mixer le langage de base, OQL et OML.

Requêtes nommées

Pour simplifier l'écriture des requêtes, on peut **nommer** des requêtes intermédiaires. C'est l'équivalent des vues dans les bases de données relationnelles.

```
define salary_map as
  select street, average_salary:avg(select e.salary from partition)
  from Employee e
  group by street: e.address.street

define sorted_salary_map as
  select s from salary_map s
  order by s.average_salary

first(sorted_salary_map).street
```

La dernière requête permet de trouver la rue où se trouvent les employés de plus petits salaires (en moyenne).

ODMG : comparaison avec SQL3

- √ les ADT
  - ▹ En relationnel, le schéma est décrit par des tables, dont chaque colonne est d'un type choisi parmi les types prédéfinis. En SQL3 : On peut définir soi-même des types appelés ADT (abstract data type).
  - ▹ Un ADT est défini par une liste d'attributs et d'opérations. Il peut y avoir une relation de sous-type entre ADT.
- √ Les Row Types
  - ▹ Les lignes ont un type (liste des colonnes) qui peut être nommé. Dans ce cas, il est utilisé comme un type d'objets : il peut y avoir des references directes sur une ligne (row_id = object_id)
  - ▹ Notion de sous-table (?).

ODMG : Conclusion

Comparaison avec le modèle objet de l'OMG

Le modèle objet de l'OMG est un sous-ensemble de celui de l'ODMG.

- √ Les propriétés sont mieux décrites (attributs, relations)
- √ La hiérarchie de sous-types (objets, littéraux) est plus détaillée.

Bus CORBA + services CORBA = SGBDOO ? (1)

√ OUI :

- ▷ La persistence des données : le service de persistence des objets, le service cycle de vie des objets.
- ▷ Le service relations (avec intégrité référentielle)
- ▷ L'extension d'une classe : le service Collection.
- ▷ Les requêtes : le service Query (lié au service Collection)
- ▷ Exécution d'un traitement : le service de transactions, le service de concurrence, le service sécurité.

ODMG :Conclusion

√ Cette norme fournit un modèle objet et un langage de description d'objets pour les bases de données, indépendant des langages existants.

√ Elle explique aussi comment projeter ce modèle dans certains langages orientés objets.

√ Elle fournit un langage de requête compatible avec SQL2 (et SQL3).

√ Point faible : la sécurité, la cohérence.

- ▷ Pas de privilèges, de rôles, comme dans les SGBD relationnels.
- ▷ Pas de contraintes.
- ▷ Les données ne sont pas encapsulées.

√ Point fort : maintien des références quand on utilise les relations. Les extensions sont gérées par le SGBD.

√ ODMG a l'intention d'accentuer les rapprochements avec OMG.

EJB et CORBA

- √ EJB repose sur CORBA.
- √ l'architecture logicielle des EJB définit :
  - ▷ modèle de composants coté serveur
  - ▷ l'activation, la désactivation d'un EJB
  - ▷ transactions (en utilisant le service CORBA).
  - ▷ gestion des évènements
- √ quelques produits industriels (ex : BD gemstone).
- √ génération de code automatique (ex : développement NetBeans, JBuilder)
- √ pas encore simple !

Bus CORBA + services CORBA = SGBDOO ? (2)

- √ NON :
  - ▷ La notion de clés.
  - ▷ La notion de vues ou schéma externe.
  - ▷ La complexité
  - ▷ L'efficacité

Dans la réalité :

- √ CORBA +JDBC, ODBC. le serveur BD assure les services : transactions, sécurité, . . .
- √ Les EJB : Enterprise java Beans

SI fédérés : la distribution

- √ Plusieurs types de réseau : plusieurs BDs sur un réseau local, plusieurs BDs sur un réseau grande distance,...
  - ▷ interopérabilité
- √ Possibilité de duplication des données (du fait que les BDs composantes existent avant le SI fédéré → pb de cohérence
  - ▷ consultation : où chercher l'information
  - ▷ mise à jour ?

La fédération de systèmes d'informations

- √ 3 principaux critères :
  - ▷ La distribution des données.
  - ▷ L'hétérogénéité.
  - ▷ L'autonomie.

SI fédérés : l'autonomie

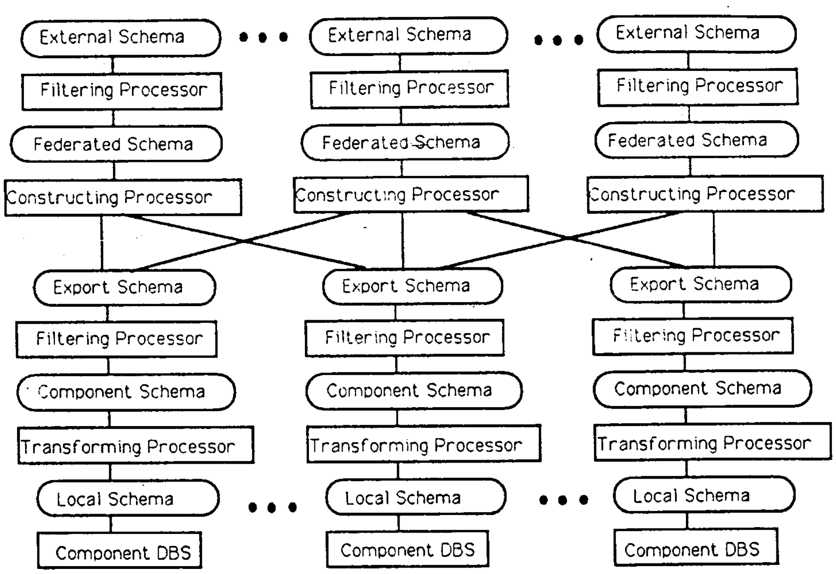
- √ peut être classée selon 4 types :
  - ▷ L'autonomie de conception.C'est la capacité de la BD composante à choisir son modèle de données, le nommage des données, l'interprétation sémantique des données, les contraintes, les fonctionnalités du système, l'implémentation.
  - ▷ L'autonomie de communication. C'est la capacité du DBMS à décider quand et comment répondre à une requête d'un autre DBMS.
  - ▷ L'autonomie d'exécution. C'est la capacité du DBMS à exécuter des transactions locales, à décider de l'ordre des requêtes externes
  - ▷ L'autonomie d'association. C'est la capacité du DBS à décider du niveau d'implication (accès aux données) dans une ou plusieurs fédérations.

On parle alors de fédération faiblement couplée ou fortement couplée

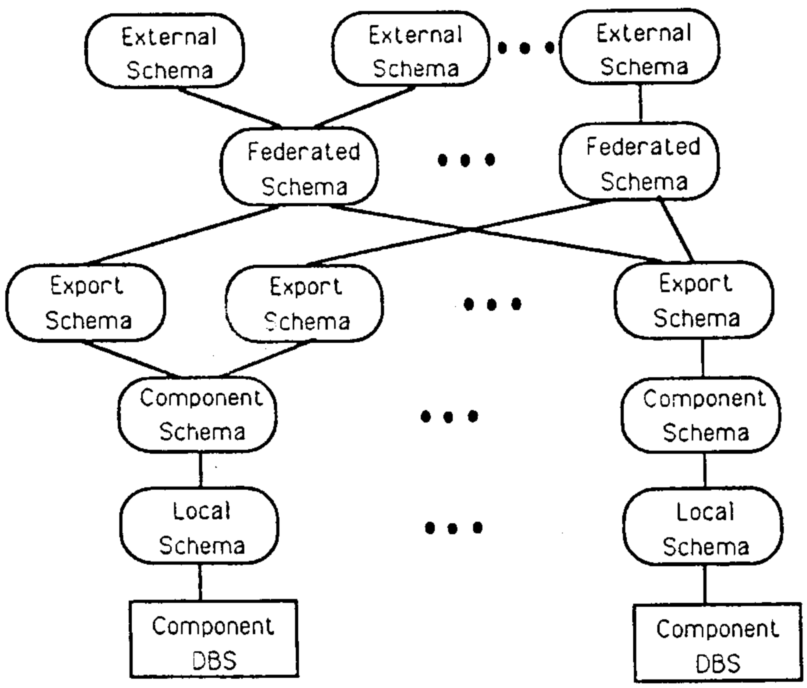
SI fédérés : l'hétérogénéité

- √ SGBD différents :
  - ▷ différence de modèle : modèle objet (objet), modèle relationnel (table), modèle CODASYL (record). Présence héritage ou pas, . . .
  - ▷ différence langages de reqêtes : OQL, SQL , QUEL, QBE, SQL3,. . .
  - ▷ différence gestion des transactions.
- √ différence des contraintes des données, différence de systèmes d'unités (échelles)
- √ l'hétérogénéité sémantique : (différence d'interprétation des données) la plus difficile à détecter et à résoudre.

SI fédérés : Construction de l'architecture



SI fédérés : Architecture



CORBA et SI fédérés

- √ CORBA peut résoudre :
  - ▷ L'interopérabilité, la fédération de bus ORB (protocole GIOP)

    Un modèle canonique et un langage canonique : IDL
  - ▷ La gestion des transactions fédérées : fédération de transactions.
- √ CORBA ne permet pas de résoudre :
  - ▷ Intégration de schéma. Une intégration complète automatisée est impossible. Seuls des outils d'aide permettent d'automatiser ce processus : langage de comparaison de schémas, modèle sémantique riche,. . .→ UML ?

    les vues ? la représentation multiple ?

La fédération : pourquoi faire ?

- √ Faire cohabiter l'ancien et le nouveau : évolution des SI
- √ Aide à la décision , tableaux de bord (LabOGP)
- √ rejoint la coopération.