Le Modèle Objet de l 'ODMG

Object Database Management Group

Didier DONSEZ

Université de Valenciennes

Institut des Sciences et Techniques de Valenciennes

donsez@univ-valenciennes.fr

Sommaire

- L'ODMG
- Les standards ODMG
- L 'architecture
- ODL, OQL, OML C++, Smaltalk et Java
- Relation avec CORBA
- Conclusion & Bibliographie

ODMG
Object Database Management Group

http://www.odmg.org

- Composition
  - Président
    - Rick Cattell, SUN
  - Membres
    - GemStone Systems, IBEX Computing SA, Object Design, Objectivity, O2 Technology, POET Software, UniSQL, Versant Object Technology
- But
  - Définir un standard permettant la portabilité des schémas de base et des programmes développés sur des SGBDs-OO différents
    - un standard ODMG-93 (94)
    - et son evolution ODMG-II (Mars 1997)

Concepts du modèle objet de l 'ODMG

- Objet, attribut, association
- opérations (méthodes), exceptions
- Héritage Multiple
- Extent et Clé
- Identité, nommage et durée de vie des objets
- Valeurs (litéraux) atomiques, structurées, collection
- Collections list, set, bag, array
- Transactions, Contrôle de Concurrence, Verrouillage

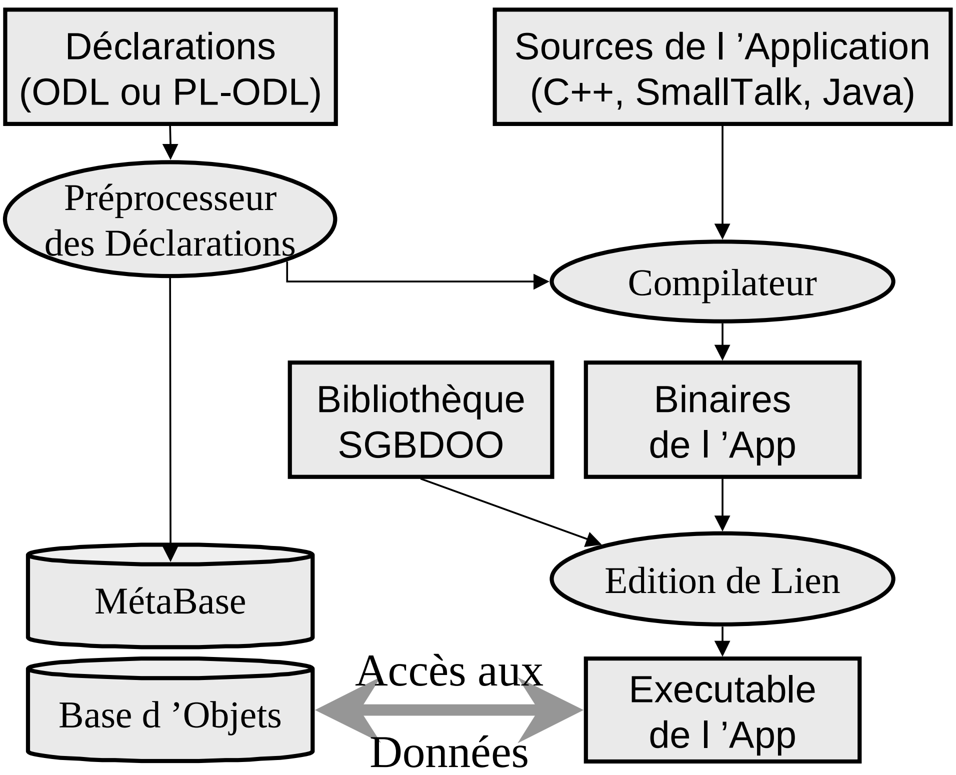
Les standards ODMG-93 et ODMG-II

- Définition
  - Spécification
    - ODL - Object Definition Language
    - OIF - Object Interchange Format
  - Manipulation
    - Non-Procédural

      OQL - Object Query Language
    - Procédural

      OML - Object Manipulation Language
      - » extensions (binding) Persistance et Transaction
      - » langages généraux : C++, Smalltalk puis Java (06/97)
    - Imbrication des langages

L 'Architecture



ODL - Object Definition Language

- Etend la syntaxe IDL de CORBA
  - interface = spécification d 'un type

```
interface Employee : Person {
   attribute int numemp;
   attribute float basesalary;
   attribute Struct Addr { string street, string city } address;
   relationship Dept dept  inverse Dept::members;
   float salary();
}
```

  - Héritage simple
  - classe = interface + une implantation du type
- Types prédéfinis
  - Collections paramétrées Set, List, Bag, Array
  - Exception
  - Type (introspection par la métabase)
  - Transaction

OQL - Object Query Language

- Langage déclaratif d 'interrogation
  - compatibilité avec l 'ordre SELECT de SQL-92
  - proposé par O2 Technology
- Fonctionnalités générales
  - tous les types ODMG et identité d 'objet
  - requête select ... from ... where ...
  - polymorphisme de collections sources
  - imbrication de requêtes, requêtes nommées
  - appel d 'opérateurs et de méthodes
  - opérateurs ensemblistes (union, intersect, except)
  - opérateur universel (for all)
  - quantifieur existentialiste (exists)
  - order by, group by, fonctions d 'agrégat (count, sum, min, max, avg)

Un exemple en OQL

- Exemple

```
select cl.name, paid
from    (        select x from Companies c, c.clients x
                 where c.address.city = "New York"
        ) cl
where count (cl.orders) > 2
order by paid: sum ( select o.price from cl.orders o )
```

- Exercice:
  - A quelle question répond cette requête ?

OML - *Object Manipulation Language*

- Principe
  - étendre les langages procéduraux standards au support des objets persistants
    - persistance
    - navigation
    - collection
    - transaction
    - accès à des requêtes OQL
  - pas de transformation de la syntaxe du langage cible
- 3 bindings spécifiés
  - OML/C++ *(ODMG-93 et ODMG-II)*
  - OML/SmallTalk *(ODMG-93 et ODMG-II)*
  - OML/Java *(ODMG-II)*

Types de Persistance

- Persistance Directe
  - l 'objet est crée Persistant ou Transitoire (Transient)
    - au commit de la transaction, les nouveaux objets persistants sont effectivement créés
    - les références sur un objet détruit sont dites pendantes (dangling)
  - cas de OML/C++
- Persistance par atteignabilité
  - un objet crée n 'est persistant au commit que s 'il est accessible par une racine de persistance (i.e. un objet nommé)
  - cas de OML/SmallTalk et de OML/Java
    - ces langages utilisent de base des ramasses miettes

OML - C++ (i)
Le binding vers le C++

- Binding des types utilisateurs
  - les interfaces utilisateurs T sont transformées en classes d 'objets persistants dérivant de la classe racine d_Object
- Le binding utilise intensivement les templates C++
  - Pour la référence sur les objets persistants
    - d_Ref<T>, d_Ref_Any
  - Pour les associations
    - d_Rel_List<T,MT>,d_Rel_Ref<T,MT>,d_Rel_Set<T,MT>
  - Pour les collections prédéfinies
    - super-classe d_Collection<T>
    - d_Set<T>, d_Bag<T>, d_Varray<T>, d_Dictionary<K,V>, d_Extent<T>, d_Iterator<T>, d_List<T>

OML - C++ (ii)
Compléments

- Autres classes
  - `d_Database, d_Date, d_OQL_Query, d_String, d_Time, d_Timestamp, d_Interval, d_Transaction, d_Error`
- Métaclasses
  - `d_Access_Kind, d_Alias_Type, d_Attribute, d_Class, d_Collection_Type, d_Constant, d_Enumeration_Type, d_Exception, d_Inheritance, d_Keyed_Collection_Type, d_Meta_Object, d_Module, d_Operation, d_Parameter, d_Primitive_Type, d_Property, d_Ref_Type, d_Relationship, d_Scope, d_Structure_Type, d_Type`

OML - Imbrication des Langages

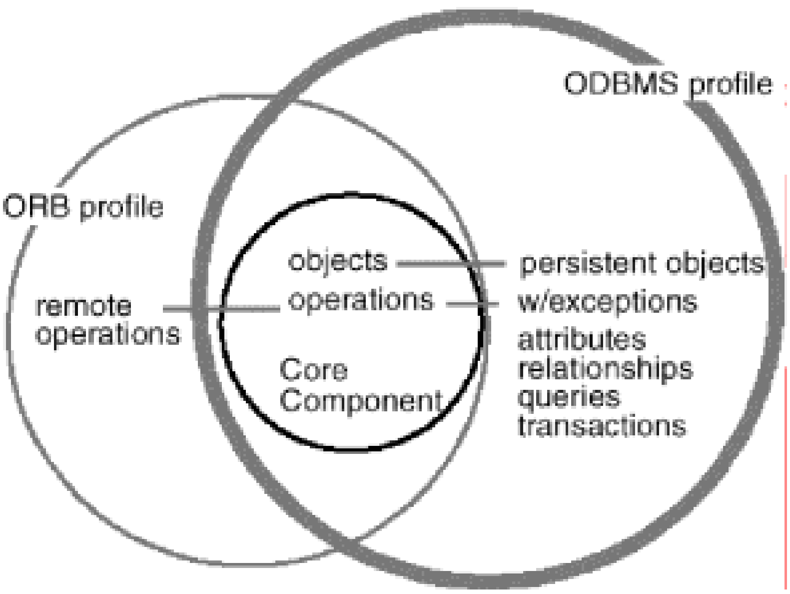
- Appel de méthodes depuis un autre langage
- Procédurale et déclaratif OQL_Query
  - exécution d 'une requête OQL
  - parcours procédural de la collection résultat

```
Transaction inscription;
inscription.begin();
   d_Ref<Etudiant>    jean = new(database) Etudiant("Jean");
   Etudiants.insert(jean);
   d_OQL_query le_cours(
                  "element(select c from Les_Cours c where c.sujet=$1)");
   le_cours << "Math-sup";
   d_Ref<Cours> c; d_OQL_execute(c,le_cours);
   jean->suit.insert(c);
inscription.commit();
```

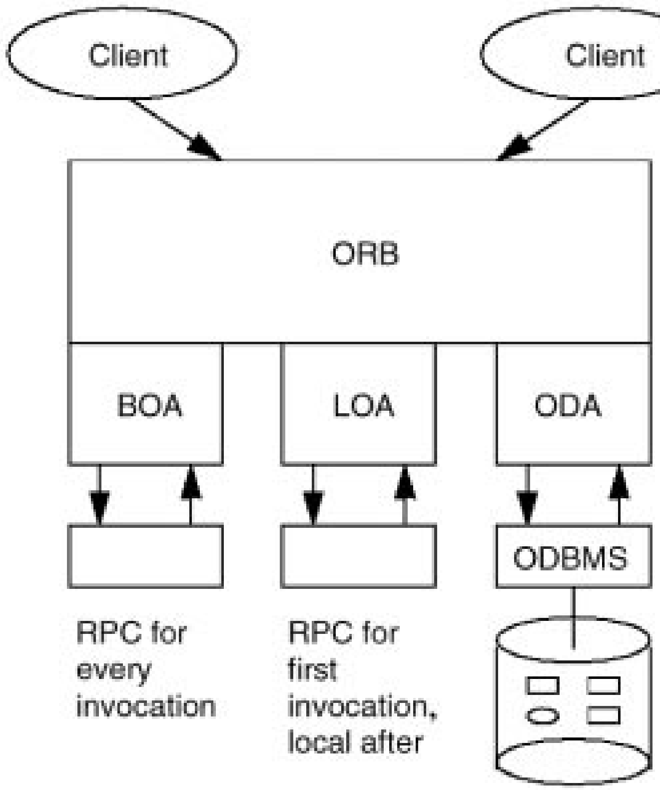
Relations avec OMG/CORBA

- ODL
  - extension de l 'IDL de CORBA
- ODA (Object Database Adaptor)
  - Granularité importante des objets CORBA
  - des millions d 'objets persistants dans une BDOO
  - Adapteur (BOA) permettant à un sous ensemble des objets persistants de publier des services (leur méthodes) sur un bus CORBA
    - Annexe B du livre de l 'ODMG II
- OQS (Object Query Service)
  - Requêtes OQL sur des collections d 'Objets CORBA.

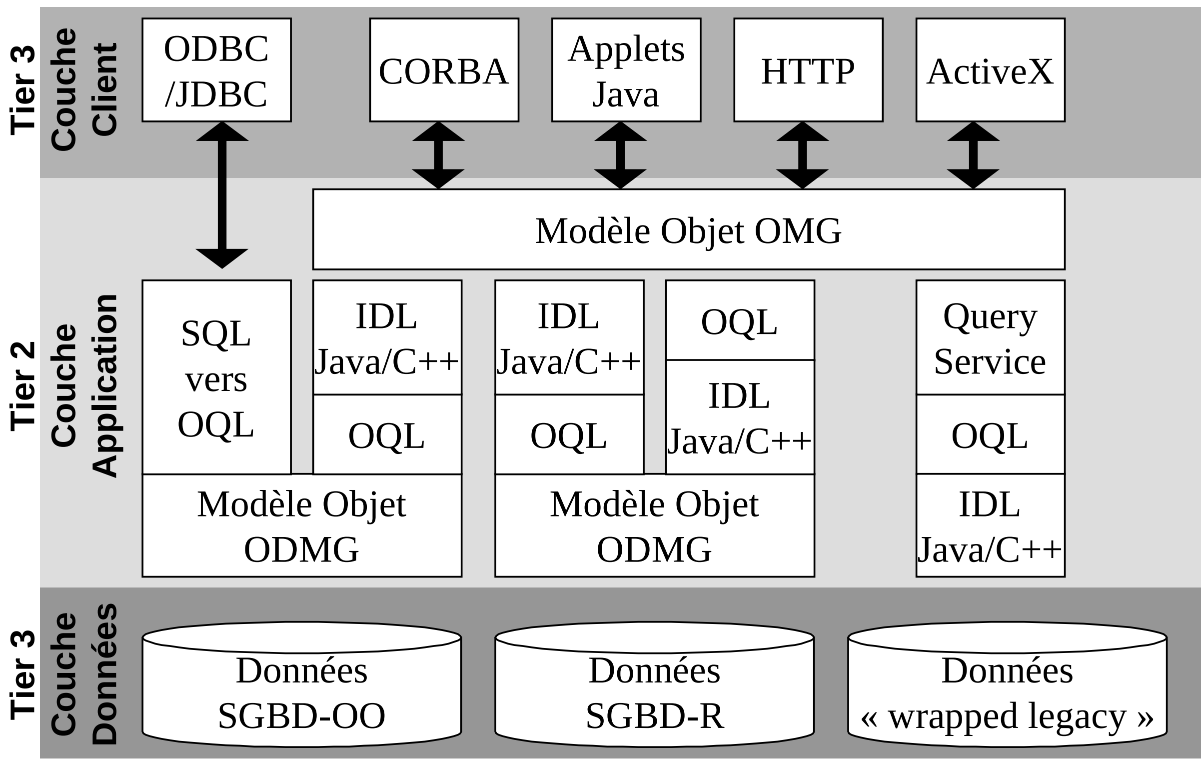
Relations avec l 'OMG/CORBA IDL et ODL



Relations avec l 'OMG/CORBA
ODA Object Database Adapter



L 'architecture 3 tiers selon l 'ODMG



Object Definition Language
(ODMG)

Didier DONSEZ
Université de Valenciennes
Institut des Sciences et Techniques de Valenciennes
donsez@univ-valenciennes.fr

Motivations

- ODMG
  - normalisation des langages d 'interface des SGBD-OO
- Principe : indépendance entre le LDD et les LMD
  - LDD : ODL
  - LMD : C++, SmallTalk, Java
    - binding des définitions ODL vers les LDD des langages cibles
- ODL étend la syntaxe de l 'IDL de OMG/CORBA

Types ODL

- Distinction entre Littéral et Objet

Littéral

- Types atomiques
  - int, real, string
- Constructeurs de type
  - énumération enum
  - structures struct, union
  - collections génériques : set, bag, list, array
- *chaque littéral est « caractérisé » par sa valeur*
  - *L1 et L2 sont égaux si leurs valeurs sont égales*

Objet

- définition de l 'interface
- *chaque objet est identifié par son identité*
  - *O1 et O2 sont égaux si leurs identifiants sont égaux*

Définition d 'une interface

- interface = spécification d 'un type
  - super (Héritages simple et multiple)
  - extent, clés candidates
  - attributs

    attribute <type> <nomattr>;
  - associations et associations inverses

    relationship <type> <nomasso>
    inverse <nom-d-interface>::<nomasso>;
  - méthodes

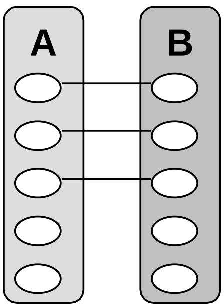
    <type-retourné> <nommeth> (<type-paramêtre> : <type>, ...)
    raise (<type-d-exception>);
    - » <type-paramêtre> : in, out, inout
- classe
  - interface + une implantation particulière du type
    - dans un des LMD disponibles

Exemple

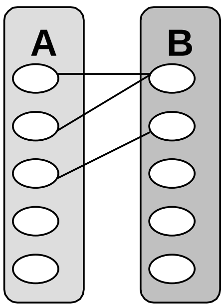

```
interface Employee {
    attribute int numemp;            attribute string name;
    attribute float basesalary;      attribute Date birthday;
    attribute Struct Addr { string street, string city, int zip }   address;
    attribute enum TypeEmp { worker,manager}                         typeEmp;
    attribute Set<string>                                            phones;
    relationship Dept dept  inverse Dept::members;
    float salary();
 }
interface Dept {
   attribute string name;
   attribute Struct Employee::Addr address;
   relationship Set<Employee> members inverse Employee::dept
   float totsalary() raise(Uncalculable);
}
```


Cardinalités des Assocations

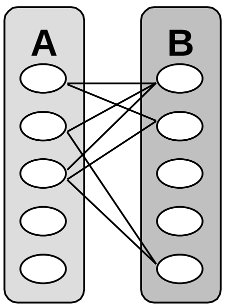
Multiplicity of Relationships



One-to-one
1-1



Many-to-one
N-1



Many-to-many
N-M

```
interface A {
 relationship B rb
       inverse B::ra;
}
interface B {
 relationship A ra
       inverse A::rb;
}
```

```
interface A {
 relationship B rb
       inverse B::ra;
}
interface B {
 relationship Set<A> ra
       inverse A::rb;
}
```

```
interface A {
 relationship Set<B> rb
       inverse B::ra;
}
interface B {
 relationship Set<A> ra
       inverse A::rb;
}
```

Rôles

- Nom des associations

Rôles asymétriques

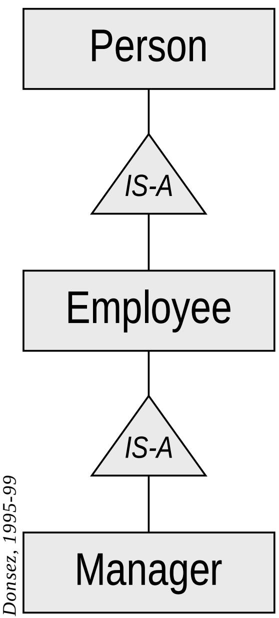
```
interface Person { ...
    relationship Person epoux  inverse Person::epouse;
    relationship Person epouse  inverse epous;
    relationship Dept dept  inverse Dept::members;
```

Rôles symétriques

```
...
    relationship Person conjoint inverse Person::conjoint;
}
```


Héritage

Héritage simple




```
interface Person {
    attribute string name;
    attribute date_t birthday
    attribute Addr addr;
}
interface Employee : Person {
    attribute int basesalary;
    float salary();
}
interface Manager : Employee {
    attribute int bonus;
    float salary();
}
```

Héritage multiple

- laisser au concepteur

Identité d 'objets

- OID : Identifiant d 'objet
  - chaque objet est identifié de manière unique par son identifiant d 'objet OID
  - Propriétés de l 'OID
    - Unique
    - Invariable

      i.e ne dépend pas de la valeur de l 'objet comme les clés de SQL
    - N 'est pas accessible au développeur (contrairement à SQL3)

Clés

- Clé = groupe d 'attributs

```
interface Person ( key numemp){
    attribute string numemp;
    attribute string numss;
    attribute string name;
    attribute date_t birthday
    attribute Addr addr;
}
```

- Clés candidates
  - spécification de plusieurs clés candidates

```
interface Person ( key numemp; (numss); (name,birthday) ) { ... }

interface Enseignement( key (filiere,intitule);(salle,heure) ) { ... }
```

Extent

- extent
  - ensemble des objets instancés pour l 'interface

```
interface Person
  extent Persons
{
    attribute string name;
    ...
}
```

```
interface Employee : Person
  extent Employees
 {
    attribute int basesalary;
    ...
}
```

  - utilisé par OQL (clause FROM)

Divers

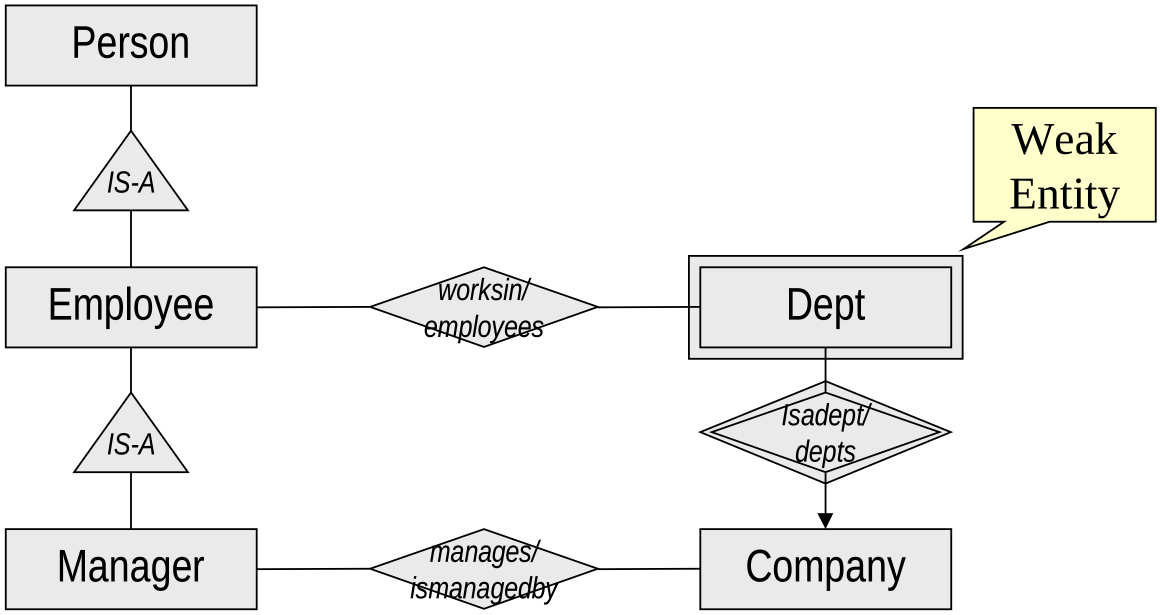
- typedef : Définition de types litéraux

```
typedef int Franc, Euro;
typedef struct { string street, string city, int zip } Addr ;
typedef enum Couleur {rouge, rose, blanc};
```

- module
  - regroupement dans le même espace de nom

```
module DRH {
    typedef struct { ... } Addr ;
   interface Person { ...; Addr addr; ... };
};
module Marketing {
    typedef struct { ... } Addr ;
   interface Customer { ...; Addr deliv_addr; Addr bill_addr; ...   };
};
```

Exemple



Exemple

```
interface Person {
    attribute string name;
    attribute date_t birthday
    attribute Addr addr;

    int age();
}
```

```
interface Employee : Person {
    attribute int basesalary;
    relationship Dept worskins
       inverse Dept::employees;

    float salary();
}
```

```
interface Manager : Employee {
    attribute int bonus;
    relationship Dept manages
       inverse Dept::manager;
    float salary();
}
```

Exemple

```
interface Company {
    attribute string name;
    attribute Addr fiscaddr;

    relationship Set<Dept> depts
        inverse Dept::company;

    int allsalaries();
}
```

```
interface Dept {
    attribute string name;
    attribute Addr postaddr;

    relationship Company company
        inverse Company::depts;
    relationship Set<Employee> employees
        inverse Employee::worksin
   relationship Manager manager
        inverse Manager::manages

    int allsalaries();
}
```


Object Query Language
(ODMG)

Didier DONSEZ

Université de Valenciennes

Institut des Sciences et Techniques de Valenciennes

donsez@univ-valenciennes.fr

Motivation

- proposé par O2 Technology (Ardent Software) à l 'ODMG 93

- Langage déclaratif d 'interrogation
  - compatibilité avec le SELECT de SQL-92
- Eviter les problème d 'Impedance Mismatch
  - Embedded-SQL in C (ou C++) ou SQL/CLI
  - Typage et Valeurs nulles
- Repose donc sur le LDD ODL

Fonctionnalités générales

- Fonctionnalités générales
  - tous les types ODMG et identité d 'objet
  - requête select ... from ... where ...
  - polymorphisme de collections sources
  - imbrication de requêtes, requêtes nommées
  - appel d 'opérateurs et de méthodes
  - opérateurs ensemblistes (union, intersect, except)
  - opérateur universel (for all)
  - quantifieur existentialiste (exists)
  - order by, group by, fonctions d 'agrégat (count, sum, min, max, avg)
  - imbrication dans les binding C++, Smalltalk, Java

Le Typage dans OQL

- reprend le système de type d 'ODL

Types atomiques

- int, real, string

Constructeurs de type

- structures struct
- collections : set, bag, list, array

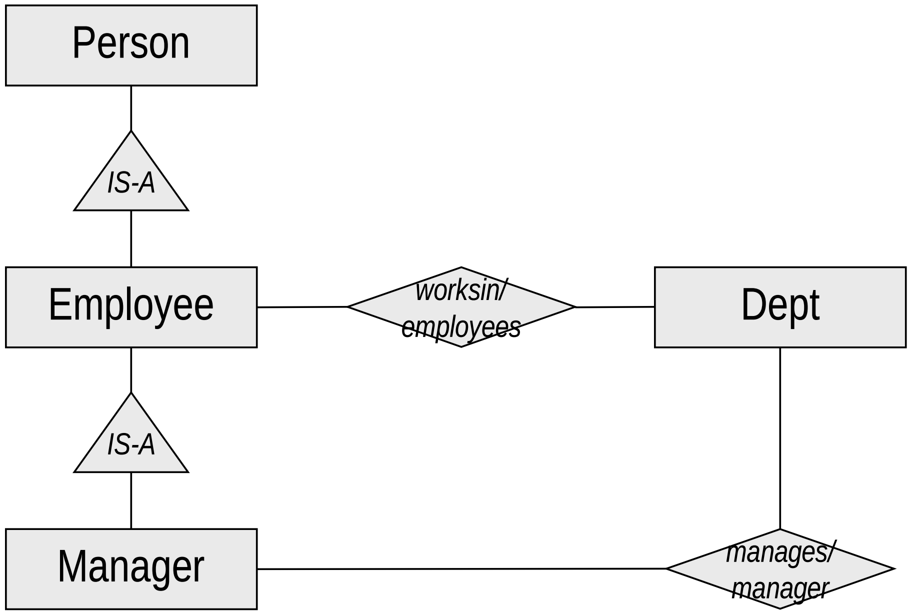
Objets

- Buildin : date, ...
- User-Defined

Remarque

- Set(Struct()) et Bag(Struct()) jouent un rôle spécial par rapport aux tables

Exemple Personne-Employé-Manager



Exemple

```
interface Person {
    attribute string name;
    attribute date_t birthday
    attribute Addr addr;

    int age();
}
```

```
interface Employee : Person {
    attribute int numemp;
    attribute int basesalary;
    relationship Dept worksin
        inverse Dept::employees;
    float salary();
}
```

```
interface Manager : Employee {
    attribute int bonus;
    relationship Dept manages
        inverse Dept::manager;
    float salary();
}
```


Exemple

```
interface Dept {
   attribute string name;
   attribute Addr postaddr;

  relationship Set<Employee> employees
        inverse Employee::worksin
  relationship Manager manager
        inverse Manager::manages

   int allsalaries();
}
```

Extent

- Collection d 'objets instanciés dans une classe
  - optionnel
- Exemple

interface Person	(extent **Persons**)	{ ... }
interface Employee : Person	(extent **Employees**)	{ ... }
interface Manager : Employee	(extent **Managers**)	{ ... }
interface Dept	(extent **Depts**)	{ ... }

Accès aux éléments

- Soit `c` un objet de la classe `C`

  soit `a` un attribut de `C`

  `c.a` est la valeur de l 'attribut a de l 'objet c

  soit `r` une relation de `C`

  `c.r` est la valeur de collection
  ou une référence vers des objets

  soit `m` une méthode de `C`

  `c.m`, `c.m()` est la valeur du résultat
  de l 'invocation de la méthode `m` sur c

- Exemple

  Soit e un objet de la classe `Employee`

```
e.birthday, e.name
e.salary, e.salary(), e.age, e.age()
e.worksin.name, e.worksin.allsalaries
e.worksin.manager.name
```

La requête Select-From-Where

- Syntaxe
  - SELECT <*listes de valeurs*>
    FROM <*listes de collections ou de requêtes imbriquées* >
    WHERE <*condition*>

- Renommage des champs et des collections

  SELECT **nom:** *e.name,*
  salaire: *e.salary(),*
  addrtravail: *e.worksin.addrpost*
  FROM *Employees* **e**
  WHERE *e.worksin.name="R&D"*

Type du résultat

- SELECT ... FROM ... WHERE ...

  un multi-ensemble (bag) de structure dont les champs sont la liste de valeurs (clause SELECT)

  SELECT *e.name, e.worksin.addrpost* FROM *Employees e* WHERE *e.worksin.name="R&D"*

  est du type

  Bag(Struct(name : string, addrpost: Addr))

- SELECT DISTINCT... FROM ... WHERE ...

  Set(Struct(name : string, addrpost: Addr))

- SELECT ... FROM ... WHERE ... ORDER BY ...

  List(Struct(name : string, addrpost: Addr))

Collections dans la clause FROM

- Extent, Collection

  SELECT *e.name, d.addrpost*
  FROM ***Employees e, e.worksin d***
  WHERE *d.name="R&D"*

- Requête imbriquée

  SELECT m.name, m.salary()
  FROM **(SELECT d.manager FROM Depts d) m**
  WHERE m.salary > 1000000

Accès aux éléments de la réponse

- ELEMENT pour une liste à 1 élément
  - exemple agrégat

  sal = ELEMENT(SELECT *e.salary*
  FROM ***Employees e*** WHERE *e.numemp=102)*

- [i] pour une liste à N éléments
  - [i] : accès au i+1 ème élément

  listsal = SELECT e. name, e.salary FROM Employees e
  ORDER BY s.salary ASC
  listsal[0].name est l 'employé le moins payé

EXISTS et FOR ALL

- Quantifieur existentialiste exists ... in

```
SELECT d.name
FROM Depts d
WHERE EXISTS e  IN d.employees : e.salary>1000000
```

- Opérateur universel for all ... in

```
SELECT d.name
FROM Depts d
WHERE FOR ALL emp  IN (     SELECT e
                            FROM d.employees e
                            WHERE e.age<30)
            : emp.salary>10000
```


Agrégat

- 5 fonctions d 'agrégat (count, sum, min, max, avg)

```
allsalRnD = ELEMENT(  SELECT SUM(e.salary)
                      FROM Depts d, d.employees e
                      WHERE d.name="R&D")

allsalRnD = SUM(      SELECT e.salary
                      FROM Depts d, d.employees e
                      WHERE d.name="R&D")
```

Groupage

- Exemple

  SELECT departement : d.name, massesal : SUM(e.salary)
  FROM Depts d, d.employees e
  GROUP BY d.name

- Usage de Partition
  - partition désigne l 'ensemble des objets ou litéraux de la partition courante du groupage

  SELECT codepostal, AVG(SELECT p.e.salary FROM partition p)
  FROM Employees e
  GROUP BY codepostal : e.addr.codpost

Résultat Temporaire : define

- Exemple

```
define CarteSalaire as
  SELECT      codepostal,
              salmoy : AVG(SELECT e.salary FROM partition)
  FROM Employee e
  GROUP BY codepostal : e.addr.codpost

define CarteTrieeSalaire as
  SELECT c FROM CarteSalaire c
  ORDER BY c.salmoy

CarteTrieeSalaire[0].codepostal
```

Coercion

- Exemple

```
SELECT p.name
FROM Persons p
WHERE p NOT IN ( SELECT (Person)e FROM Employees e)
```

Didier Donsez, 1995-99

Un petit exercice

- Vous vous souvenez de cette requête

```
select cl.name, paid
from    (       select x from Companies c, c.clients x
                where c.address.city = "New York"
        ) cl
where count (cl.orders) > 2
order by paid: sum ( select o.price from cl.orders o )
```

- Donnez la métabase en ODL
  - un client est une personne

Didier Donsez, 1995-99

Un petit exercice

- Vous vous souvenez de cette requête

```
interface Person { attribute String name; }
interface Client : Person {
        relationship Set<Company> companies inverse Company::clients;
        relationship Set<Order> orders inverse Order::client;
}
interface Company (extent Companies) {
        attribute Struct Addr {string city } address;
        relationship Set<Company> clients inverse Client::suppliers;
}
interface Order {
        relationship Client client inverse Client::orders;
        float price();
}
```


Bibliographie

- "The Object Database Standard, ODMG 2.0", Ed Morgan Kaufmann, 1997, 288 pages, ISBN 1-55860-463-4, Edité par R.G.G. Cattell, Douglas Barry, Dirk Bartels, Mark Berler, Jeff Eastman, Sophie Gamerman, David Jordan, Adam Springer, Henry Strickland, and Drew Wade
  - la description du modèle de référence de l 'ODMG
- David Jordan, "C++ Object Databases, Programming with the ODMG Standard", Ed Computer & Engineering Publishing Group, Object Technology Series, ISBN: 0-201-63488-0
  - programmation en C++ de SGBD OO conforme à l 'ODMG
- Jeffrey D. Ullman, Jennifer Widom, "A First Course in Database Systems", 1ère édition, Ed. Prentice Hall Engineering, Science & Math, Avril 1997, ISBN 0-13-861337-0, 470 pp.
  - décrit l 'ODL et le compare à SQL3 et au modèle E/R

Sites

- ODMG - http://www.odmg.org
  - le site de l 'ODMG
- JDBMS - http://www.jdbms.org
  - propose une implantation d 'OQL en Java