The Object Database Standard: ODMG

R.G.G. Cattell

1991 Reunión de vendedores

1993 ODMG-93

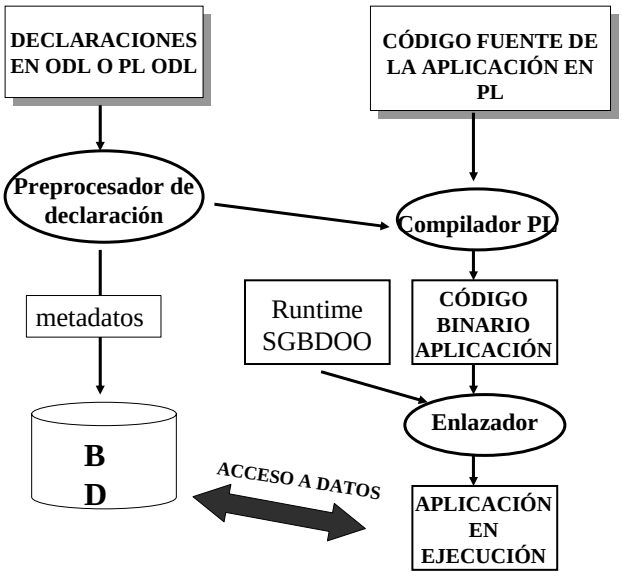
1995 ODMG-93 v 1.2

1997 ODMG v 2.0

1998 Revisión para JDK 1.2

2000 ODMG v 3.0

ARQUITECTURA:



MODELO DE OBJETOS

MODELO DE OBJETOS

La primitiva fundamental es el **objeto**: en ODMG todo son objetos

Distingue entre: **objetos** mutables e inmutables (**literales**). Un objeto tiene un identificador único y un literal no tiene identificador.

Modos de **identificar** un objeto:

- Identificador de objeto (OID).- único e inmutable
- Nombres de objeto.- un objeto puede tener varios; un nombre corresponde a un único objeto
- Descripciones de objetos.- predicados sobre atributos que permiten identificar unívocamente a un objeto

Los objetos se clasifican en **tipos**: todos los objetos de un mismo tipo tienen unas propiedades y un comportamiento común:

Comportamiento: conjunto de operaciones que se pueden ejecutar sobre un objeto

Estado: valores que toman sus propiedades (atributos y relaciones)

• Un **tipo** tiene una especificación externa y una o más implementaciones

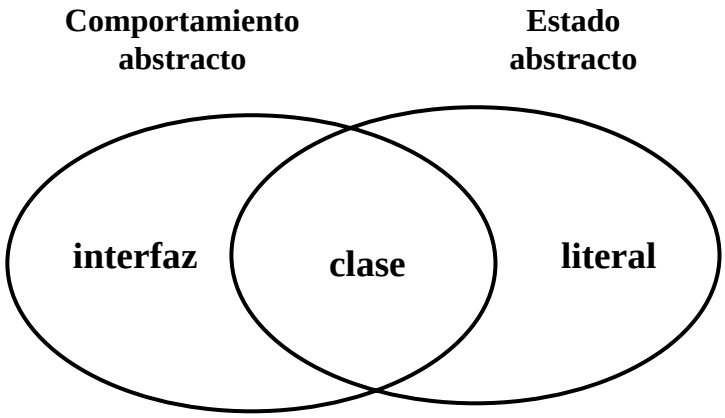
La especificación externa de un tipo es una descripción abstracta, independiente de la implementación, de las propiedades, operaciones y excepciones de un tipo.

•Un **interfaz** describe sólo el comportamiento abstracto de un tipo de objeto.

•Un **literal** define sólo el estado abstracto de un tipo literal.

• Una **clase** es una especificación que define el comportamiento abstracto y el estado abstracto de un tipo de objeto

Especificación de tipo



GENERALIZACIÓN

• Relación **ISA** (:), define la **herencia de comportamiento** entre tipos de objetos (interfaces o clases).

•Relación **EXTENDS** (*extend*), define la **herencia de estado** entre tipos de objetos (clases, no literales).

ISA: simple a múltiple herencia de comportamiento

EXTENDS: herencia **simple** entre clases

Los tipos de objeto se clasifican en **supertipos/subtipos**:

- Un subtipo **hereda** propiedades y operaciones del supertipo
- Un subtipo puede añadir propiedades y operaciones **propias**
- Un subtipo puede **redefinir** propiedades y operaciones del supertipo
- Se soporta **herencia múltiple** (solo de comportamiento: ISA)

OBJETOS vs. LITERALES

Todo objeto denotable tiene **identidad**:

Objetos.- representación de la identidad es una combinación de bits generada por el sistema (**OID**).

Literales.- representación de la identidad es la codificación en bits de su **valor**

El **estado de un objeto puede variar** modificando el valor de sus atributos o de sus relaciones

Los **literales** son *"objetos"* cuyas instancias **no pueden variar**

TIEMPO DE VIDA

Determina cómo se gestiona la memoria y el espacio reservados para un objeto:

- transitorio
- persistente

OBJETOS COMPUESTOS

COLECCIONES:

Número variable de elementos
Los elementos pueden ser objetos o literales
Todos los elementos del mismo tipo

TIPOS

LIST: colección ordenada de elementos
SET: colección de elementos desordenada que no admite duplicados
BAG: colección de elementos desordenada que admite duplicados
ARRAY: el modelo soporta arrays de una dimensión y de longitud variable
DICTIONARY: secuencia desordenada de pares <clave, valor> sin claves duplicadas

ESTRUCTURAS:

Número fijo de elementos

TIPOS

DATE. Fecha

INTERVAL. Intervalo de tiempo

TIME. Hora

TIMESTAMP. Fecha y hora

LITERALES ESTRUCTURADOS

Literales atómicos: long, short, unsigned long, unsigned short, float, double, boolean, octet, char, string, enum

Literales colecciones: set, bag, list, array, dictionary

Literales estructuras: date, intreval, time, timestamp

Literales NULL

LENGUAJE DE DEFINICIÓN DE OBJETOS (ODL)

PRINCIPIOS DEL ODL, Cattell (1994):

- *Soporta la semántica completa del modelo de objetos de ODMG*
- *No es un lenguaje completo*
- *Independiente del lenguaje de programación*
- *Extensible (nuevas funcionalidades, optimizaciones físicas)*

Notación BNF

<definición de tipo>::=
interface <nombre del tipo>(:<lista de supertipos>)
{
(<lista de propiedades del tipo>)
(<lista de propiedades>)
(<lista de operacione>)
}

<propiedad del tipo>::=
extent <nombre de la extensión>
|key(s) <lista de claves>

<especificación del atributo>::=
(attribute)
<tipo de dominio> (tamaño) <nombre del atributo>

<especificación de interrelación>::=
(relationship)
<destino del camino><nombre del camino>
inverse<camino inverso>
(order by <lista de atributos>)

<especificación de operación>::=
<tipo devuelto> <nombre de la operación>
(<lista de argumntos>) (raises (<excepciones levantadas>)

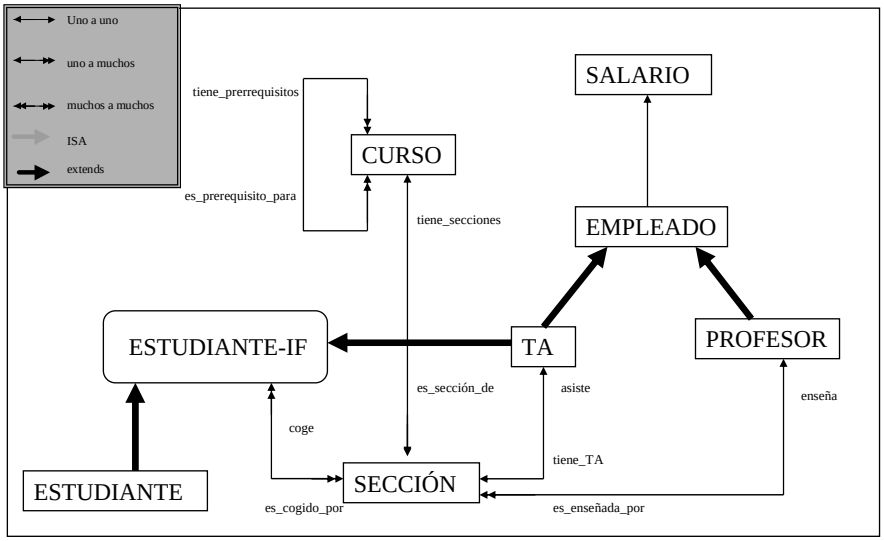
<argumento>::=
<papel> (<nombre del argumento>:)<tipo del argumento>

Ejemplo:

```
class Person (extent people)
{
attribute string name;
attribute struct Address {unsigned short number, string
        street, string city_name} address;
relationship Person spouse inverse Person::spouse;
        relationship set<Person> children inverse
Person::parents; relationship list<Person> parents inverse
Person::children; void birth (in string name);
boolean marriage (in string person_name)
        raises (no_such_person);
unsigned short ancestors (out set<Person> all_ancestors)
        raises (no_such_person);
void move (in string new_address);
};
```


Ejemplo:

```
class City (extent cities)
{
attribute unsigned short city_code;
attribute string name;
attribute set<Person> population;
};
```



Ejemplo en ODL

```
class Course (extent courses)
{
attribute string name;
attribute string number;

relationship list<Section> has_section
        inverse Section::is_section_of;
relationship set<Course> has_prerequisites
        inverse Course::is_prerequisite_for;
relationship set<Course> is_prerequisite_for
        inverse Course::has_prerequisites;

boolean offer (in unsigned short semester)
        raises (alredy_offered);
boolean drop (in unsigned short semester)
        raises (not_offered)};
```

```
class Section (extent sections)
{
attribute string number;

relationship Professor is_taught_by
        inverse Professor::teaches;
relationship TA has_TA
        inverse TA::assists;
relationship Course is_section_of
        inverse Course::has_sections;
relationship set<Student> is_taken_by
        inverse Student::takes;
};
```

```
class Salary
{
attribute float base;
attribute float overtime;
attribute float bonus;
};
```

```
class Employee (extent employees)
{
attribute string name;
attribute short id;
attribute Salary annual_salary;

void hire ();
void fire()
    raises (no_such_employee)
};
```



```
class Professor extends Employee (extent professors)
{
attribute enum Rank {full, associate, assistant} rank;

relationship set<Section> teaches
        inverse Section::is_taught_by;

short grant_tenure()
        raises (ineligible_for_tenure)};
```

```
interface Student-IF
{struct Address {string college, string roon_number}
attribute string name;
attribute string student_id;
attribute Address dorm_address;

relationship set<Section> takes
        inverse Section::is_taken_by;

boolean register_for_course (in unsigned short course,
                             in unsigned short Section)
        raises (unsatisfied_prerequisites, section_full, course_full);
boolean drop_course (in unsigned short course)
        raises (not_registered_for_that_course)
void assign_major (in unsigned short Departament);
short transfer (in unsigned short old_section,
                in unsigned short new_section)
        raises (section_full, not_registered_in_section)};
```



```
class TA extends Employee:Student-IF
        (extent Employee)
{
relationship Section assists
        inverse Section::has_TA;

attribute string name;
attribute string student_id;
attribute Address dorm_address;

relationship set<Section> takes
        inverse Section::is_taken_by;
};
```

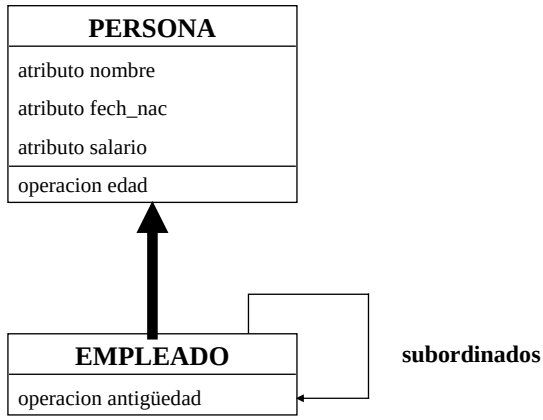
```
class Student:Student-IF
      (extent Students)
{
attribute string name;
attribute string student_id;
attribute Address dorm_address;

relationship set<Section> takes
        inverse Section::is_taken_by;
};
```

LENGUAJE DE CONSULTA DE OBJETOS (OQL)

PRINCIPIOS DEL OQL, Cattell (1994):

No es computacionalmente completo
Es un lenguaje declarativo
Tiene una sintaxis similar a la del SQL (OQL V 1.2)
Proporciona sintaxis para mezclar las consultas con C++, Smalltalk y Java
Proporciona primitivas de alto nivel para el tratamiento de todo tipo de colecciones
No proporciona operadores de actualización explícitos (se actualiza a través de operaciones definidas sobre los objetos)
Es facilmente optimizable (debido a que es declarativo)



Extensión de persona: ***personas***
Extensión de empleado: ***empleados***


```
select distinct x.edad
from x in personas
where x.nombre="Ana"
```

literal del tipo set <integer>

```
select distinct struct (a:x.edad, s:x.salario)
from x in personas
where x.nombre="Ana"
```

literal del tipo set <struct (a:integer, s:integer)>

```
select distinct struct (a:x.nombre, smp:
        (select y
         from y in x.subordinados
         where y.salario>300000)
from x in empleados
```

literal del tipo set <struct (nombre:string, smp:bag<empleado>)>

Para cada empleado: nombre del empleado y sus subordinados mejor pagados

Ejemplo de utilización del *select* en la *cláusula from:*

```
select struct (a:x.edad, s:x.salario)
from x in
        (select y
          from y in empleados
          where y.antigüedad="10")
```

literal del tipo bag <struct (a:integer, s:integer)>

select* ----------> *bag
select distinct* ----------> *set

Creación de objetos:

a) Objetos mutables

Persona (nombre: “María”, fech_nac:”11/2/69”, salario:100.000)

b) Literales

struct (a:10, b:”María”)

El resultado de una consulta es un literal. Para obtener un objeto mutable es necesario definir un tipo de objeto mutable:

```
type vectint: bag<integet>;
type estado attributes
        a:integer
        b: salario
end_type;
type estados: bag<estado>;
```

```
vectint (select distinct x.edad
        from x in personas
        where nombre=“Pedro”)
```

objeto mutable de tipo vectint

```
estados (select estado (a:x.edad, b:x.salario)
        from x in personas
        where nombre=“Pedro”)
```

objeto mutable de tipo estados

Definición de consultas:

En OQL es posible dar nombre al resultado de una consulta y utilizarla en otra

Ejemplo:

```
define juanes as select distinct x
                 from x in persona
                 where x.nombre="Juan";

select distinct x.salario
from x in juanes
```


Ejemplos de consultas mediante expresiones:

- 27
  devuelve 27
- nil
  devuelve el objeto nil
- personas
  devuelve el conjunto de todas las personas
- juanes
  devuelve el conjunto de personas que se llaman Juan

EXPRESIONES SOBRE COLECCIONES:

count, sum, min, max, avg
group...in...by...where, sort...in...by
for...all...in, exis...int, in
select...from...where

Ejemplo:
count(personas)

EXPRESIONES ARITMÉTICAS:

+, -, *, /, - unario, mod, abs

Ejemplo:
count(personas)-count(empleados)

EXPRESIONES DE COMPARACIÓN:
=, !=, <, <=, >, >=

EXPRESIONES LÓGICAS:
not, and, or

Ejemplo:
not(true)

EXPRESIONES SOBRE CONJJUNTOS:
intersect, union, except

Ejemplo:
bag(2,2,3,3,3) except bag(2,3,3,3)
resultado: bag(2)

CONVERSORES DE TIPO:

listtoset, element, flatten, nombre de clase

Ejemplo:

```
define juan as element(
        select distinct x
        from x in empleado
        where x.nombre="Juan");
```


OPERADORES DE ACCCESO:

• / -> (aplicados a un atributo, una operación o una relación)
first / last (primero / último elemento de una lista o un vector)

Ejemplo:

personas.nombre
personas->nombre se obtiene el nombre de todas las personas

juan.subordinado.nombre
juan->subordinado->nombre nombre de los subordinados de Juan

juan.edad
juan->edad edad de juan

* ***"The Object Database Standard"***
Release 2.0
Edited by R.G.G. Cattell and D. Barry
Morgan Kaufmann Publishers, Inc., San Francisco 1997
(Versión revisada para JDK 1.2 en 1998)

"Object Data Management. Object Oriented and Extended Relational Database Systems"
R.G.G. Cattell
Addison-Wesley Publishing Company, 1994