Standard ODMG

ALBERTO BELUSSI

SECONDA PARTE

ANNO ACCADEMICO 2011/'12

ODMG Object Definition Language (ODL)

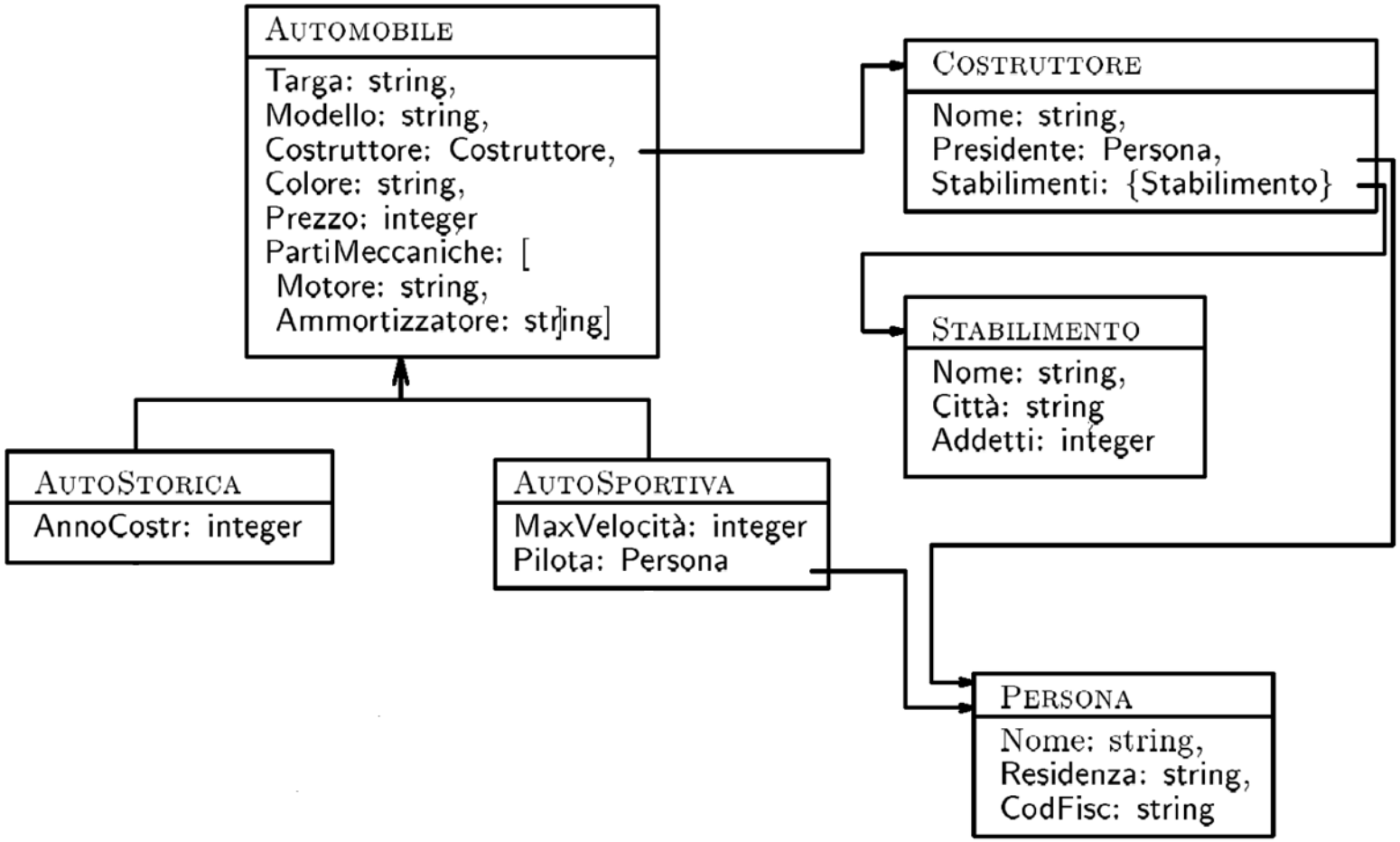
ODMG Object Definition Language

- Linguaggio per la definizione astratta di interfacce, letterali, classi, transazioni, eccezioni, …
  - ODL supporta i costrutti semantici di ODMG OM
- ODL non è legato ad alcun linguaggio di programmazione
- ODL non è inteso come un linguaggio di programmazione, ma un linguaggio di definizione per la specifica di oggetti

ODMG Object Definition Language

- ODL è un Data Definition Language per tipi di oggetti
  - ODL permette di definire le caratteristiche dei tipi: proprietà ed operazioni
  - ODL permette di definire solo la segnatura delle operazioni
- ODL consente di definire tipi di oggetti che possono essere implementati con linguaggi di programmazione diversi

ODL: Esempio 1



ODL: Esempio 1


```
struct T_PartiMec
            {String Motore,
             String Ammortizzatore};
class Automobile
( extent Automobili  key Targa )
 {     attribute String Targa;
       attribute String Modello;
       attribute String Colore;
       attribute String Prezzo;
       attribute T_PartiMec PartiMeccaniche;
       relationship Costruttore Costruttore
             inverse Costruttore::Costruisce;
}
```

ODL: Esempio 1


```
class AutoStorica extends Automobile
 {      attribute short AnnoCostr;
  }

class AutoSportiva extends Automobile
 {      attribute Integer MaxVelocità;
        relationship Set<Persona> Piloti
                inverse Persona::Ha_Pilotato;
  }
```

ODL: Esempio 1


```
class Costruttore
( extent Costruttori key Nome )
 {     attribute String Nome;
       relationship Persona Presidente
             inverse Persona::Presidenza;
       relationship Set<Automobile> Costruisce
             inverse Automobile::Costruttore;
       relationship Set<Stabilimento> Stabilimenti
             inverse Stabilimento::StabCostruzione;

}
```

ODL: Esempio 1


```
class Stabilimento
 {      attribute String Nome;
        attribute String Città;
        attribute long Addetti;
        relationship Costruttore StabCostruzione
              inverse Stabilimento::Stabilimenti;
}

class Persona
 {      attribute String Nome;
        attribute String Residenza;
        attribute String CodiceFiscale;
        relationship Costruttore Presidenza
              inverse Costruttore::Presidente;
        relationship Set<AutoSportiva> Ha_pilotato
              inverse AutoSportiva::Pilota;
}
```

ODL: Esempio 2

- Facciamo riferimento ad una schema che descriva:
  - Documenti: con chiave “titolo”. Ogni documento ha un titolo, un insieme di autori che sono impiegati ed è descritto dallo stato e dal contenuto
  - Articoli: sono tipi particolari di documenti per cui si memorizza la rivista su cui sono stati pubblicati e la data di pubblicazione
  - Progetti: con chiave “nome”. Ogni progetto ha un nome e viene descritto da un insieme di documenti. Inoltre, è composto da una insieme di task. Ad ogni progetto sono assegnati un insieme di impiegati ed un capo (sempre impiegato).
  - Task: ogni task viene descritto dai mesi uomo necessari per completarlo, dalla data di inizio e di fine e dal nome dell’impiegato responsabile del task.

ODL: Esempio 2


```
class Documento
   (extent documenti key titolo)
   {
        attribute string titolo;
        attribute List<Impiegato> autori;
        attribute string stato;
        attribute string contenuto;
   }

class Articolo extends Documento
   (extent articoli)
   {
        attribute string rivista;
        attribute date data_pubbli;
   }
```

ODL: Esempio 2


```
class Progetto
(extent progetti key nome)
 {      attribute string nome;
        attribute Set<Documento> documenti;
        attribute Set<Task> tasks;
        relationship Set<Impiegato> assegnati
               inverse Impiegato::progetti;
        relationship Impiegato capo
               inverse Impiegato::dirige;
}
```

ODL: Esempio 2


```
class Task
(extent task)
 {      attribute unsigned short mesi_uomo;
        attribute date data_in;
        attribute date data_fine;
        attribute Impiegato responsabile;
        relationship Set<Impiegato> partecipanti
                inverse Impiegato::tasks;

  }
```

ODL: Esempio 2


```
class Impiegato
(extext impiegati key codicefiscale)
{       attribute string nome;
        attribute string residenza;
        attribute string codicefiscale;
        relationship Set<Progetto> progetti
                inverse Progetto::assegnati;
        relationship Task dirige
                inverse Task::capo;
        relationship Set<Task> tasks
                inverse Task::partecipanti;
}
```

ODMG Object Query Language (OQL)

ODMG Object Query Language (OQL)

- Il linguaggio OQL, originariamente sviluppato per il sistema O2, è stato adattato dal comitato ODMG, con varie modifiche, ed è considerato il linguaggio standard di interrogazione degli OODBMS

- OQL si può ritenere SQL-like
  - Usa le stesse parole chiave di SQL
  - Come SQL consente un accesso dichiarativo ai dati
  - Come SQL non è un linguaggio computazionalmente completo (non include, ad esempio strutture di controllo)

ODMG Object Query Language (OQL)

- OQL consente di invocare metodi
  - I metodi possono includere interrogazioni
- OQL permette di accedere agli oggetti attraverso il nome ad essi associato
- Una caratteristica peculiare di OQL rispetto a SQL e necessaria, vista la possibilità di strutturare gli oggetti, è la possibilità di specificare path di accesso ai dati tramite la notazione a “punto” (dot notation)

ODMG Object Query Language (OQL)

- OQL non prevede primitive per modificare gli oggetti nella base di dati
  - coerentemente al modello object-oriented le modifiche devono avvenire solo tramite i metodi specificati nelle classi
- I comandi OQL possono essere immersi in un linguaggio di programmazione

ODMG Object Query Language (OQL)

- OQL fornisce primitive per considerare insiemi di oggetti
- OQL è un linguaggio funzionale dove gli operatori possono essere liberamente composti
  - il risultato è un oggetto il cui tipo appartiene all'insieme dei tipi di ODMG OM

ODMG Object Query Language (OQL) – Schema di riferimento


```
Interface Iperson {
  exception   NoSuchPerson{};
  void        birth (in string name);
  boolean              marriage(in string person_name)
                      raises(NoSuchPerson);
  unsigned short       ancestors(out set<Person>
                                      all_ancestors);
  void                 move(in string new_address);
};
```

Aggiunge un nuovo figlio ad una persona

ODMG Object Query Language (OQL) – Schema di riferimento


```
class Person: Iperson
(extent people)
{
  attribute string                    name;
  attribute string                    surname;
  attribute unsigned short            age;
  attribute enum
       Gender {male, female}          gender;
  attribute struct Address
   {unsigned short   number,
    string           street,
    City             city}           address;

  relationship Person spouse
       inverse Person::spouse;
  relationship set<Person> children
       inverse Person::parents;
  relationship list<Person> parents
       inverse Person::children;
};
```

ODMG Object Query Language (OQL) – Schema di riferimento


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

ODMG Object Query Language (OQL)

- Query OQL che estrae l'età delle persone di nome "Mario"


```
SELECT distinct x.age
FROM people x
WHERE x.name = "Mario"
```

Tipo del risultato:

`set<unsigned short>` (literal)

ODMG Object Query Language (OQL)

- Query OQL che estrae l'età e il sesso delle persone di cognome "Rossi":

```
SELECT distinct struct(a: x.age, s: x.gender)
FROM people x
WHERE x.surname="Rossi"
```

Tipo del risultato:

`set<struct(unsigned short, enum)>` (literal)

ODMG Object Query Language (OQL)

Query OQL che, data una persona `p`, estrae il nome della città dove vive il coniuge di ogni oggetto p della classe persona

```
SELECT p.spouse.address.city.name
FROM people p
```

La query segue la relazione "`spouse`" usando una path expression e raggiunge un oggetto `person` che rappresenta il coniuge di `p`, quindi accede al letterale `address` che ha una struttura nella quale l'attributo `city` è un oggetto di tipo `City` nel quale esiste l'attributo `name`.

ODMG Object Query Language (OQL)

Query OQL che estrae il nome dei figli delle persone presenti nell'estensione `people` (relazione uno a molti):

```
SELECT c.name
FROM people p, p.children c
```

Tipo del risultato:

`bag<string>` (`bag` perché non c'è `distinct`)

ODMG Object Query Language (OQL)

Query OQL che estrae il nome dei figli delle persone presenti nell'estensione `people` (relazione uno a molti):

```
SELECT distinct c.name
FROM people p, p.children c
```

Tipo del risultato:

`set<string>` (perché c'è `distinct`)

ODMG Object Query Language (OQL)

Query OQL che estrae l'indirizzo dei figli di ogni persona contenuta nel DB:

```
SELECT c.address
FROM people p, p.children c
```

Tipo del risultato:

```
bag<Address>
```

ODMG Object Query Language (OQL)

Query OQL che estrae l'indirizzo dei figli delle persone che vivono in Via Roma ed hanno almeno 2 figli. Si richiedono gli indirizzi dei figli che vivono in una città diversa dai loro genitori:

```
SELECT c.address
FROM people p, p.children c
WHERE p.address.street ="Via Roma" AND
          count(p.children) >= 2 AND
          c.address.city != p.address.city
```

Tipo del risultato:

```
bag<Address>
```

ODMG Object Query Language (OQL)

Query OQL che estrae le persone che hanno il nome di un fiore (connessione computata tramite un join perché gli oggetti considerati non sono direttamente connessi):

```
SELECT p
FROM Persons p, Flowers f
WHERE p.name = f.name
```

ODMG Object Query Language (OQL) – Schema di riferimento

Facciamo riferimento ad uno schema con tipi

- **Impiegato** (Nome, Nascita:(Citta, Data), StipendioAnnuo, Subordinati, Età(), AssegnaSubordinato(...))

  con estensione **impiegati**

- **Dipartimento** (NomeDip, Sedi:{(NomeSede, CittaSede)}, Direttore)

  con estensione **dipartimenti**

ODMG Object Query Language (OQL) – Schema di riferimento


```
typedef struct Nascita_t {
  attribute string Citta;
  attribute date Data; }

class Impiegato
( extent impiegati key Code)
{ attribute string Code;
  attribute string Nome;
  attribute Nascita_t Nascita;
  attribute unsigned long StipendioAnnuo;
  relationship Set<Impiegato> Subordinati
       inverse Impiegato::Capo;
  relationship Impiegato Capo
       inverse Impiegato::Subordinati;
  short Eta ();
  void AssegnaSubordinato (in Impiegato Sub);
}
```

ODMG Object Query Language (OQL) – Schema di riferimento


```
typedef Struct Sede_t {
  attribute string NomeSede;
  attribute string CittaSede;
}

class Dipartimento
( extent dipartimenti )
{ attribute string NomeDip;
  attribute Set<sede_t> Sedi;
  attribute Impiegato Direttore;
  void AssegnaDirettore (in Impiegato Dir);
}
```

ODMG Object Query Language (OQL)

- Seleziona gli stipendi degli impiegati di nome Paolo

```
select distinct x.StipendioAnnuo
from x in impiegati
where x.Nome = "Paolo"
```

- Seleziona nome e età degli impiegati di nome Paolo

```
select distinct
        struct(N: x.Nome, E: x.Eta())
from x in impiegati
where x.Nome = "Paolo"
```

ODMG Object Query Language (OQL)

- Attraversamento di una relazione

```
select struct (Dip: x.NomeDip,
               Dir: x.Direttore.Nome)
from x in dipartimenti
```

- Equivalente a

```
select struct (Dip: x.NomeDip,
               Dir: y.Nome)
from x in dipartimenti, y in impiegati
where x.Direttore = y
```

Identità

ODMG Object Query Language (OQL)

- Per ciascun impiegato, seleziona il nome e l'insieme dei subordinati con stipendio annuo maggiore di 100000

```
select distinct struct(
            Nome: x.Nome,
            Sub: (select y
                  from y in x.Subordinati
                  where
                   y.StipendioAnnuo>100000))
from x in impiegati
```

ODMG Object Query Language (OQL)

La clausola `select` può comparire anche entro la clausola `from`:

```
select struct(N: x.Nome, E: x.Eta())
from x in (select y
           from y in impiegati
           where y.Stipendio > 100000 )
where x.Nome = "Paolo"
```

ODMG Object Query Language (OQL)

- Attributi contenenti il valore indefinito (NIL). Il risultato dell'accesso a una proprietà indefinita è il letterale UNDEFINED. Esistono due predicati per manipolare tale letterale

```
is_undefined(UNDEFINED) = true
is_defined(UNDEFINED) = false
```

- Se un predicato di una WHERE ritorna UNDEFINED è considerato come fosse FALSE.

```
select x
from x in impiegati
where x.address.city = "Verona"
```

Se un impiegato non ha indirizzo x.address.city ritorna UNDEFINED e il predicato diventa FALSE.

ODMG Object Query Language (OQL)

- Creazione di viste — e accesso a un attributo composto

```
define Romani as
        select x
        from x in impiegati
        where x.Nascita.Citta = “Roma”
```

- Appartenenza

```
select x.Eta()
from x in impiegati
where x in Romani
```

ODMG Object Query Language (OQL)

Quantificatori (queste sono query booleane):

```
for all X in Impiegati:
          X.Stipendio > 200000

exists X in Impiegati:
          X.Eta() < 19
```

si può anche riscrivere così:

```
exists (select x from x in Impiegati
        where x.Eta() < 19)
```

ODMG Object Query Language (OQL)

- Subordinati di subordinati

```
select distinct struct(
            Imp: x, Subsub: z)
from x in impiegati, y in x.Subordinati,
     z in y.Subordinati
```

- Restituisce la *lista* degli impiegati, ordinata per nome

```
SELECT x
FROM x in impiegati
ORDER BY x.Nome
```

ODL: Esempio 1


```
class Automobile
(extent automobili  key Targa )
 {     attribute String Targa;
       attribute String Modello;
       attribute String Colore;
       attribute String Prezzo;
       attribute struct TPartiMeccaniche
             {String Motore,
              String Ammortizzatore} PartiMeccaniche;
       relationship Costruttore costruttore
             inverse Costruttore::costruisce;
       relationship Stabilimento stabilimentoCostr
             inverse Stabilimento::autoCostruite;
}
```

ODL: Esempio 1


```
class AutoStorica extends Automobile
(extent automobiliStoriche)
{       attribute unsigned short AnnoCostr; }

class AutoSportiva extends Automobile
(extent automobiliSportive)
{       attribute unsigned short MaxVelocità;
        relationship Set<Persona> pilota
                inverse Persona::ha_Pilotato;
}
```

ODL: Esempio 1


```
class Costruttore
(extent costruttori key Nome)
{       attribute String Nome;
        relationship Persona presidente
                inverse Persona::presidenza;
        relationship Set<Automobile> costruisce
                inverse Automobile::costruttore;
        relationship Set<Stabilimento> stabilimenti
                inverse Stabilimento::proprietario;
}
```

ODL: Esempio 1


```
class Stabilimento
(extent stabilimenti key Nome)
{       attribute String Nome;
        attribute String Città;
        attribute unsigned long Addetti;
        relationship Costruttore proprietario
                inverse Stabilimento::stabilimenti;
        relationship Set<Automobile> autoCostruite
                inverse Automobile::stabilimentoProd;

}

class Persona
(extent persone key CodiceFiscale)
{       attribute String Nome;
        attribute String Residenza;
        attribute String CodiceFiscale;
        relationship Costruttore presidenza
                inverse Costruttore::presidente;
        relationship Set<AutoSportiva> ha_pilotato
                inverse AutoSportiva::pilota;
}
```

OQL: Esempio 1

Estrarre le targhe delle automobili rosse

```
SELECT x.Targa
FROM x in Automobile
WHERE x.Colore = "Rosso"
```

Tipo del risultato:

```
set<string>
```

OQL: Esempio 1

Estrarre il nome delle persone che siano contemporaneamente piloti e costruttori delle stesse auto sportive:

```
SELECT y.Nome
FROM x in AutoSportiva,
     y in x.piloti
WHERE
y = x.costruttore.presidente
```

Identità

Tipo del risultato:

`bag<string>` (possono esistere piloti omonimi)

OQL: Esempio 1

Estrarre le auto sportive Ferrari che non sono state costruite a Maranello e che superano i 250 Km/h di velocità massima

```
SELECT a.Targa
FROM a in autoSportive,
     c in costruttori, s in stabilimenti
WHERE a.MaxVelocità > 250 AND
      c = a.costruttore AND
      c.Nome = "Ferrari" AND
      s = a.stabilimentoCostr AND
      s.Città != "Maranello"
```

OQL: Esempio 1

Estrarre il nome dei costruttori che vendono auto sportive a un prezzo superiore a 100000 euro; per ciascuno di essi, elencare le città e numero di addetti degli stabilimenti.

```
SELECT distinct struct(
  nome: x.costruttore.Nome,
  stab: (SELECT struct(Città: y.citta,
                       numAddetti: y.addetti)
         FROM y in Stabilimento
         WHERE y in
                x.costruttore.stabilimenti))
FROM x in AutoSportiva
WHERE x.Prezzo > 100000
```

Tipo del risultato:

```
set(struct(string,bag(struct(string,unsigned long))))
```

OQL: Esempio 1

Estrarre le auto partizionate in base al costruttore, riportando per ogni costruttore il numero di auto contenute nella base di dati.

```
SELECT Costr,
  numAuto: count(select x.a from x in partition)
FROM Automobile a
GROUP BY (Costr: a.costruttore)
```

Tipo del risultato prima della clausola select:

```
set<struct(Costr: string,
           partition: bag<struct(a: Automobile)>)>
```

Tipo del risultato finale:

```
set<struct(Costr: string,numAuto: unsigned long)>
```

OQL: Esempio 1

Estrarre le auto partizionate in base al prezzo.

```
SELECT *
FROM Automobile a
GROUP BY ( PrezzoBasso: a.Prezzo < 10000,
           PrezzoMedio: a.Prezzo >=10000
                        AND a.Prezzo <=25000,
           PrezzoAlto: a.Prezzo > 25000)
```

Tipo del risultato:

```
Set<struct(PrezzoBasso:boolean,
           PrezzoMedio: boolean,
           PrezzoAlto: boolean,
           partition: bag<struct(a: Automobile)>)
```

OQL: semantica delle interrogazioni

Forma base di un'interrogazione OQL

```
SELECT <espressione che produce gli oggetti/letterali
       del risultato - funzione di x1,…,xn >
FROM <collezione1> x1, …, <collezionen> xn
WHERE <espressione booleana - funzione di x1,…,xn >
```

Risultato prima della clausola SELECT:

```
bag<struct(x1: type(<collezione1>),
           …
           xn: type(<collezionen>))>
```

una ennupla $<x_1,\ldots,x_n>$ appartiene al risultato se rende vera l'espressione della clausola WHERE.

Risultato della clausola SELECT: dipende dall'espressione della stessa, ma è funzione delle ennuple prodotte nella fase precedente.

OQL: semantica delle interrogazioni

Forma di un'interrogazione OQL con il GROUP BY

```
SELECT <espressione che produce gli oggetti/letterali
        del risultato - funzione di
        partition e di g1,…,gm >
FROM <collezione1> x1, …, <collezionen> xn
WHERE <espressione booleana - funzione di x1,…,xn >
GROUP BY g1: <espressione1>, …, gm: <espressionem>
```

Risultato prima della clausola SELECT:

```
set<struct(g1: type(<espressione1>),
                …
           gm: type(<espressionem>),
           partition: bag<struct(x1: type(<collezione1>),
                                 …
                                 xn: type(<collezionen>))>
             )>
```

OQL: semantica delle interrogazioni

Forma di un'interrogazione OQL con il GROUP BY

```
SELECT <espressione che produce gli oggetti/letterali
          del risultato - funzione di x1,…,xn >
FROM <collezione1> x1, …, <collezionen> xn
WHERE <espressione booleana - funzione di x1,…,xn >
GROUP BY g1: <espressione1>, …, gm: <espressionem>
HAVING <espressione booleana su partition>
```

Una ennupla $<x_1, \ldots, x_n>$ contribuisce alla generazione dei gruppi se rende vera l'espressione della clausola `WHERE`.

Un gruppo va nel risultato se soddisfa la clausola `HAVING`.

Risultato della clausola `SELECT`: dipende dall'espressione della stessa, ma è funzione dei gruppi prodotte nella fase precedente.

ODL: Esempio 2


```
class Documento
(extent documenti key titolo)
{       attribute string titolo;
        attribute List<Impiegato> autori;
        attribute string stato;
        attribute string contenuto;
}

class Articolo extends Documento
(extent articoli)
{       attribute string rivista;
        attribute date data_pubbli;
}
```

ODL: Esempio 2


```
class Progetto
(extent progetti key nome)
{       attribute string nome;
        attribute Set<Documento> documenti;
        attribute Set<Task> tasks;
        relationship Set<Impiegato> assegnati
                inverse Impiegato::progetti;
        relationship <Impiegato> capo
                inverse Impiegato::dirige;
}
```

ODL: Esempio 2


```
class Task
(extent tasks)
{       attribute unsigned short mesiUomo;
        attribute date dataInizio;
        attribute date dataFine;
        attribute Impiegato responsabile;
        relationship Set<Impiegato> partecipanti
                inverse Impiegato::tasks;
}
```

ODL: Esempio 2


```
class Impiegato
(extext impiegati key codicefiscale)
{       attribute string nome;
        attribute string cognome;
        attribute string residenza;
        attribute string codicefiscale;
        attribute unsigned long stipendio;
        relationship Set<Progetto> progetti
                inverse Progetto::assegnati;
        relationship <Task> dirige
                inverse Task::capo;
        relationship Set<Task> tasks
                inverse Task::partecipanti;
}
```

OQL: Esempio 2

Determinare i task con almeno 20 mesi uomo il cui responsabile guadagna almeno 2000

```
SELECT t
FROM tasks t
WHERE t.mesiUomo > 20 AND
      t.responsabile.stipendio > 2000
```

il risultato è di tipo `bag<Task>`

OQL: Esempio 2

Determinare la data di inizio dei task con almeno 20 mesi uomo:

```
SELECT distinct t.dataInizio
FROM Tasks t
WHERE t.mesiUomo > 20
```

il risultato è un set di letterali: `set<date>`

OQL: Esempio 2

Determinare la data di inizio e la data di fine dei task con almeno 20 mesi uomo

```
SELECT distinct struct(
                    di: t.dataInizio,
                    df: t.dataFine)
FROM tasks t
WHERE t.mesiUomo > 20
```

il risultato è di tipo:

```
  set<struct(di:date;df:date)>
```

OQL: Esempio 2

Raggruppare gli impiegati in base ai progetti a cui partecipano, selezionare i progetti con più di 10 impiegati e riportare il nome del progetto, il numero di impiegati e lo stipendio medio degli impiegati (1/2)

```
SELECT struct(progetto: ,
              numImpiegati: count(…),
              stipendioMedio: avg(…) )
FROM impiegati i
GROUP BY (prog: i.progetti.nome)
HAVING count(select x.i from partition x) > 10
```

CONTINUA ->

OQL: Esempio 2

Raggruppare gli impiegati in base ai progetti a cui partecipano, selezionare i progetti con più di 10 impiegati e riportare il nome del progetto, il numero di impiegati e lo stipendio medio degli impiegati (2/2)

```
SELECT struct(progetto: prog,
    numImpiegati: count(select x.i
                        from partition x),
    stipendioMedio: avg(select x.i.stipendio
                        from partition x)
       )
FROM impiegati i …
```

OQL: Esempio 2

Trovare i documenti che non sono stati scritti da impiegati di Verona, riportando il titolo e lo stato del documento

```
SELECT struct(titolo: d.titolo, stato: d.stato)
FROM documenti d
WHERE NOT (exists i in d.autori:
           i.Residenza = 'Verona')
```

oppure

```
SELECT struct(titolo: d.titolo, stato: d.stato)
FROM documenti d
WHERE for all i in d.autori:
           i.Residenza != 'Verona')
```

OQL: Esempio 2

Trovare i progetti che non hanno task il cui responsabile abbia uno stipendio maggiore di 10000 euro, riportando il nome del progetto e il cognome del capoprogetto:

```
SELECT struct(progetto: p.nome,
              capo: p.capo.cognome)
FROM progetti p
WHERE NOT (exists t in p.tasks:
           t.responsabile.stipendio > 10000)
```

OQL: Esempio 2

Trovare gli impiegati che partecipano ad almeno tre task, non dirigono nessun task e partecipano a meno di tre progetti con un budget maggiore di 100000 euro, riportando il nome e il cognome dell'impiegato:

```
SELECT struct(nome: i.nome,
              cognome: i.cognome)
FROM impiegati i
WHERE count(i.tasks) > 3 AND i.dirige.is_undefined()
  AND count(SELECT x FROM i.progetti x
            WHERE x.budget > 100000) < 3
```

Riferimenti

- **The Object Data Standard: ODMG 3.0.**

  Roderic Geoffrey Galton Cattell, Douglas K. Barry, Mark Berler

  Morgan Kaufmann, 2000

UML per la rappresentazione grafica di uno schema ODL

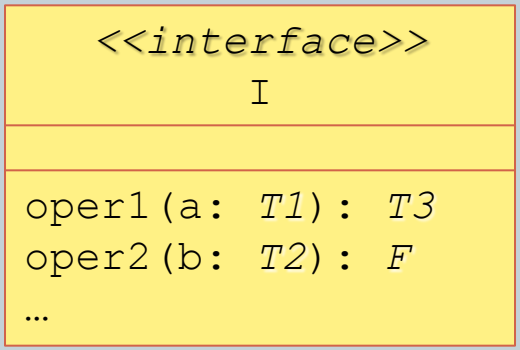
E' possibile utilizzare il modello UML e la sua sintassi grafica per specificare graficamente lo schema di una base di dati a oggetti in modo che possa essere facilmente tradotto in uno schema ODL.

Supponiamo di voler rappresentare graficamente in UML:

- Le dichiarazioni di *interface*, contenenti operazioni
- Le dichiarazioni di *typedef*, contenenti attributi
- Le dichiarazioni di *classi*, contenenti attributi e relazioni tra classi.

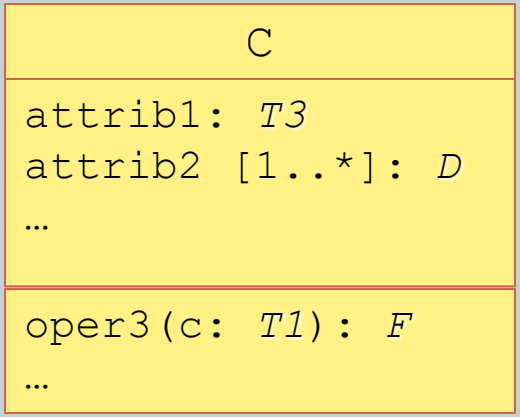
UML per la rappresentazione grafica di uno schema ODL

Rappresentazione di una *Interface I* contenente due operazioni *oper1* e *oper2* con due parametri di tipo base *T1* e *T2* dove la prima operazione restituisce un tipo base *T3* mentre la seconda restituisce un oggetto della classe *F*.



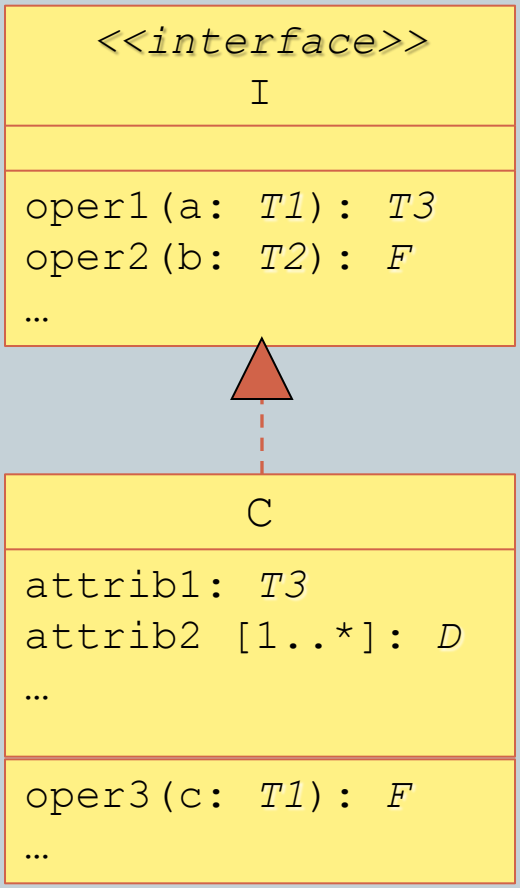
UML per la rappresentazione grafica di uno schema ODL

Rappresentazione di una *Classe C* con due attributi attrib1 e attrib2, il primo di tipo base T3 e il secondo che punta ad un set di oggetti di tipo D. E' inclusa anche l'operazione oper3 che ha un parametro di tipo base T1 e restituisce un oggetto della classe F.



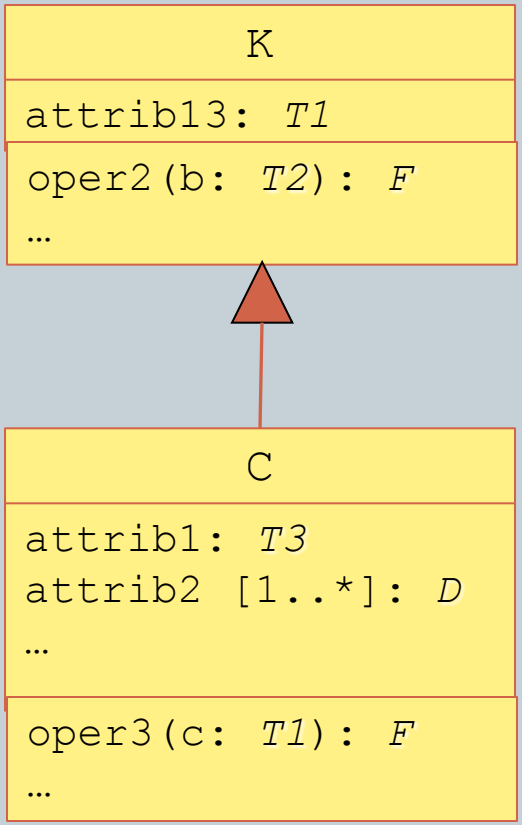
UML per la rappresentazione grafica di uno schema ODL

Una *Classe C* può anche ereditare operazioni da una *interface I*.



UML per la rappresentazione grafica di uno schema ODL

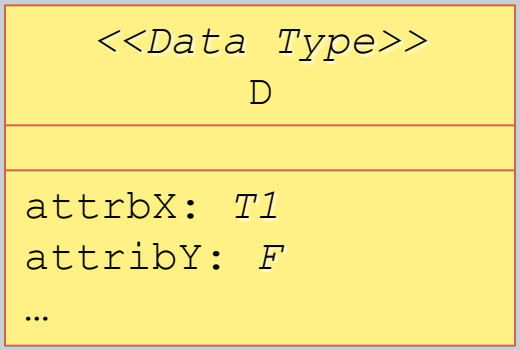
Una *Classe C* può anche ereditare un'altra *classe K*.



UML per la rappresentazione grafica di uno schema ODL

Rappresentazione di una *tipo strutturato* per letterali.

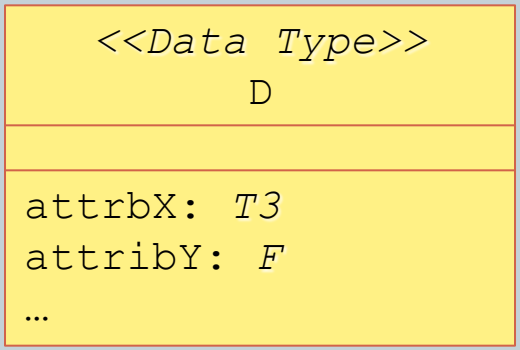
Le definizioni esterne di tipi strutturati, vale a dire quelle realizzate con il costrutto *Typedef struct*, possono essere rappresentate in UML come *DataType*.



UML per la rappresentazione grafica di uno schema ODL

Rappresentazione di una *tipo strutturato* per letterali.

Le definizioni interne di tipi strutturati, vale a dire quelle realizzate con il costrutto *struct* direttamente nella specifica di classe o interfaccia, possono essere rappresentate comunque come *DataType* esterni alla classe.



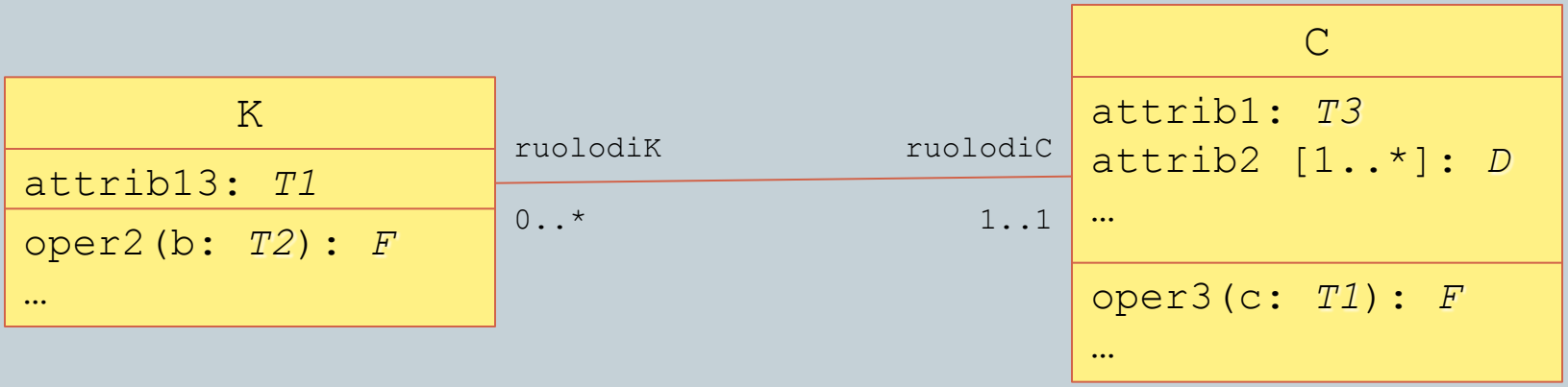
UML per la rappresentazione grafica di uno schema ODL

Rappresentazione di una *relationship* tra classi.

Per rappresentare le relationship binarie tra classi è possibile utilizzare il costrutto associazione di UML. Tale costrutto consente di precisare i ruoli e anche le cardinalità delle associazioni. Tuttavia, le cardinalità non consentono di precisare la tipologia di collezione che implementa l'associazione, tale precisazione diventa un dettaglio dello schema ODL.

UML per la rappresentazione grafica di uno schema ODL

Relationships



Class K
(extent Ks)
{ attribute T1 attrib13;
  F oper2(in b: T2);
  relationship C ruolodiC
        inverse C::ruolodiK
}

Class C
(extent Cs)
{ attribute T3 attrib1;
  attribute set<D> attrib2;
  F oper3(in c: T1);
  relationship set<K> ruolodiK
        inverse K::ruolodiC
}

UML per la rappresentazione grafica di uno schema ODL

Ulteriori dettagli della specifica ODL:

- *extent*: potrebbe diventare un tag value sulla classe
- *key*: potrebbe essere usato uno stereotipo `<<key>>` per etichettare gli attributi/ruoli che fanno parte della chiave della classe.