Basi di dati a oggetti

ALBERTO BELUSSI

SISTEMI INFOMATIVI GEOGRAFICI E MULTIMEDIALI
ANNO ACCADEMICO 2009-'10

Aree applicative emergenti

Progettazione assistita da calcolatore

- CASE (Computer-Aided Software Engineering)
- CAD (Computer-Aided Design)
- CAM (Computer-Aided Manufacturing)

- Gestione di documenti complessi
  - testi e automazione d'ufficio
  - dati ipertestuali
  - **dati multimediali**
- Applicazioni intrinsecamente complesse
  - scienza e medicina
  - sistemi real time
  - sistemi per la gestione di **dati spazio-temporali**

Caratteristiche delle nuove aree applicative

Oltre alle caratteristiche consuete di persistenza, condivisione e affidabilità, le nuove aree applicative sono caratterizzate da

- dati a struttura complessa
  - dati non-alfanumerici: immagini, dati spaziali, sequenze temporali, ...
  - tipi pre-definiti e tipi definiti dall'utente (e riutilizzati)
  - relazioni esplicite ("semantiche") tra i dati
  - aggregazioni complesse
- operazioni complesse
  - specifiche per i diversi tipi di dato — es. multimedia
  - associate anche ai tipi definiti dall'utente
  - transazioni di lunga durata

Basi di dati a oggetti

Alcune delle precedenti caratteristiche suggeriscono l'introduzione di nozioni dal paradigma orientato agli oggetti nel mondo delle basi di dati.

A partire dalla metà degli anni '80, sono stati realizzati numerosi sistemi di gestione di basi di dati a oggetti (ODBMS) — di tipo prototipale o commerciale.

I sistemi sono stati sviluppati indipendentemente, senza nessuna standardizzazione circa il modello dei dati o i linguaggi da utilizzare.

Dopo un periodo di "evoluzione," inizia ad esserci una convergenza su modello e linguaggio, per far sì che sistemi realizzati da produttori diversi possano almeno interoperare.

Basi di dati a oggetti: un modello dei dati

Una base di dati a oggetti è una collezione di *oggetti*:

- Ciascun oggetto ha un **identificatore**, detto OID, uno **stato**, e un **comportamento.**
  - L'OID garantisce l'individuazione in modo univoco dell'oggetto, e permette di realizzare riferimenti tra oggetti.
  - Lo stato è l'insieme dei valori assunti dalle proprietà dell'oggetto — è in generale un valore a struttura complessa.
  - Il comportamento è descritto dall'insieme dei metodi che possono essere applicati all'oggetto.

Basi di dati a oggetti: un modello dei dati

Gli oggetti sono associati a:

- un tipo (aspetto intensionale),
- una classe (implementazione) e
- all'estensione di una classe (aspetto estensionale)

Un tipo è una astrazione che permette di descrivere (1) lo stato e (2) il comportamento di un oggetto.

Una classe descrive l'implementazione di un tipo – struttura dei dati e implementazione di metodi tramite programmi. Ogni tipo è associato ad una sola classe.

Infine, gli oggetti vengono raggruppati in collezioni che costituiscono estensioni di classi.

Tipi

Un tipo descrive le proprietà di un oggetto (la parte statica) e l'interfaccia dei suoi metodi (la parte dinamica).
Relativamente alla parte statica, i tipi vengono costruiti a partire da:

- un insieme di tipi atomici (numeri, stringhe, ...)
- un insieme di costruttori di tipo, tra loro ortogonali
  - `record-of(`A_1`:`T_1`, ..., `A_n`:`T_n`)` (ennuple di valori/oggetti di tipo $T_1 \times ... \times T_n$)
  - `set-of(T)` (insieme di valori/oggetti di tipo T – duplicati non ammessi),
  - `bag-of(T)` (collezione di valori/oggetti di tipo T – duplicati ammessi),
  - `list-of(T)` (collezione ordinata di valori/oggetti di tipo T – duplicati ammessi)
  - `*T` (indica un riferimento ad oggetti di tipo T)

Tipi

Si ipotizza che, come consuetudine nelle basi di dati a oggetti, ogni dichiarazione di tipo inizi sempre con un costruttore di tipo record.

Dato un oggetto `x` di tipo T=`record-of(`A_1`:`T_1`,...,`A_n`:`T_n`)` possiamo parlare degli attributi $A_1, \ldots, A_n$ come **proprietà** di `x`.

Ad eccezione della regola sopra riportata, i costruttori di tipo sono completamente ortogonali.

Tipi

Esempio di tipo

```
Automobile: record-of(
      targa: string, modello: string,
      costruttore: record-of(
      nome: string,
      presidente: string,
      stabilimenti: set-of(
                  record-of( nome: string,
                              citta: string,
                              addetti: integer))),
      colore: string, prezzo: integer,
      partiMeccaniche: record-of( motore: string,
                              ammortizzatore: string))
```

Oggetti e valori

Esempio di valore

```
V1: [ targa: “MI67T891”, modello: “uno”,
      costruttore: [nome: “FIAT”,
                    presidente: “Agnelli”,
                    stabilimenti: {
                        [nome: “Mirafiori”,
                         citta: “Torino”,
                         addetti: 10000],
                        [nome: “Trattori”,
                         citta: “Modena”,
                         addetti: 1000]}],
      colore: “blu”, prezzo: 15.5M,
      partiMeccaniche: [motore: “1100CV”,
                        ammortizzatore: “Monroe”]]
```

Oggetti e valori

L'uso di tipi e valori complessi permette di associare ad un singolo oggetto una struttura qualunque.

Viceversa, nel modello relazionale alcuni concetti devono essere rappresentati tramite più relazioni.

Tuttavia, la rappresentazione proposta per automobile non è "normalizzata": vediamo come decomporla utilizzando dei riferimenti tra oggetti.

- Un **OGGETTO** è una coppia (**OID**, **VALORE**), dove **OID** (object identifier) è un valore atomico definito dal sistema e trasparente all'utente e **VALORE** è un valore di tipo complesso.
- Il valore assunto da una proprietà di un oggetto può essere l'**OID** di un altro oggetto (in tal caso si parla di riferimento ad altro oggetto)

Oggetti e valori

```
Automibile: record-of(
  targa: string, modello: string,
  costruttore: *costruttore,
  colore: string, prezzo: integer,
  partiMeccaniche: record-of(
      motore: string,
      ammortizzatore: string))
Costruttore: record-of(
  nome: string, presidente: string,
  stabilimenti: set-of(*stabilimento))
Stabilimento: record-of(
  nome: string, citta: string,
  addetti: integer)
```

Oggetti e valori

Un insieme di oggetti compatibili con lo schema:

```
O1: <OID1, [targa: "MI67T891", modello: "uno",
            costruttore: OID2, colore: "blu",
            prezzo: 15.5M, motore: "1100CV",
            ammortizzatore: "Monroe"]>
O2: <OID2, [nome: "FIAT", presidente: "Agnelli",
            stabilimenti: {OID3,OID4}]>
O3: <OID3, [nome: "Mirafiori", citta: "Torino",
            addetti: 10000]>
O4: <OID4, [nome: "Trattori", citta: "Modena",
            addetti: 1000]>
```

Identità e uguaglianza

Tra gli oggetti sono definite le seguenti relazioni

- **Identità** (O1=O2) — richiede che gli oggetti abbiano lo stesso identificatore
- **Uguaglianza superficiale** (O1==O2) — richiede che gli oggetti abbiano lo stesso stato, cioè stesso valore per proprietà omologhe
- **Uguaglianza profonda** (O1===O2) — richiede che le proprietà che si ottengono seguendo i riferimenti abbiano gli stessi valori (non richiede l'uguaglianza dello stato)

Uguglianza superficiale ⇒ Uguaglianza profonda

```
O1 = <OID1, [a, 10, OID3]>
O2 = <OID2, [a, 10, OID4]>
O3 = <OID3, [a,b]>
O4 = <OID4, [a,b]>
```

Riferimenti tra oggetti

Il concetto di riferimento presenta analogie con quello di **puntatore** nei linguaggi di programmazione, e con quello di **chiave esterna** in un sistema relazionale. Tuttavia

- i puntatori possono essere corrotti (dangling, appesi); i riferimenti a oggetti (in un buon ODBMS) vengono invalidati automaticamente in caso di cancellazione di un oggetto referenziato.
- le chiavi esterne sono visibili, in quanto realizzate tramite valori; gli identificatori d'oggetto non sono associati a valori visibili dall'utente
- modificando gli attributi di una chiave esterna, è possibile perdere riferimenti; modificando il valore di un oggetto referenziato, il riferimento continua ad esistere.

Metodi

Il paradigma OO deriva dal concetto di **Tipo di Dato Astratto**.

Un **metodo** è una procedura utilizzata per incapsulare lo stato di un oggetto, ed è caratterizzata da una **interfaccia** (o segnatura) e una **implementazione**:

- l'interfaccia comprende tutte le informazioni che permettono di invocare un metodo (il tipo dei parametri)
- l'implementazione contiene il codice del metodo
- La definizione di un tipo comprende, oltre alle proprietà, anche le interfacce dei metodi applicabili a oggetti di quel tipo

Ipotizziamo che i metodi siano assimilabili a funzioni, ovvero possano avere più parametri di ingresso ma un solo parametro di uscita.

Metodi

```
Automobile: record-of(
        targa: string, modello: string,
        costruttore: *costruttore,
        colore: string, prezzo: integer,
        partiMeccaniche: record-of(
                        motore: string,
                        ammortizzatore: string),
        public Init( // costruttore
                targa_par: string, modello_par: string, colore_par: string,
                prezzo_par: integer): automobile,
        public Prezzo(): integer, // accessore
        public Aumento( // trasformatore
                ammontare: integer)) // void
```

Classi e estensioni

I concetti di classe e di estensione non sono identici

- Una CLASSE è una implementazione di un tipo — significa che uno stesso tipo può avere implementazioni diverse. Questo è particolarmente importante non per assegnare semantiche diverse a oggetti di uno stesso tipo, ma ad esempio per implementare una (unica) semantica di un tipo in riferimento a piattaforme architetturali diverse (nel caso di una base di dati a oggetti distribuita).
- Una ESTENSIONE è una collezione di oggetti aventi lo stesso tipo; un oggetto può appartenere a più collezioni, essere rimosso da una collezione, ecc.

Ereditarietà

Tra i tipi (e le classi) di una base di dati a oggetti, è possibile definire una gerarchia di ereditarietà, con le usuali relazioni di sottotipo, l'ereditarietà dei metodi, la possibilità di overloading, overriding, late binding, ereditarietà multipla, ...

Tutto funziona come nei linguaggi di programmazione a oggetti. Esiste tuttavia una importante differenza: gli oggetti di un programma sono oggetti di breve durata, gli oggetti di una base di dati sono oggetti di lunga durata, con conseguenze non banali..

Migrazione di oggetti tra classi

Nel corso della propria esistenza, un oggetto deve mantenere la propria identità, in modo che sia possibile riferirsi ad esso in modo univoco.

Tuttavia, è possibile che un oggetto cambi tipo:

– una persona diviene uno studente, poi uno studente lavoratore, poi un lavoratore, poi una persona sposata, ...

Sembra quindi necessario un meccanismo di acquisizione e perdita di tipi, e sembra anche necessario permettere che un oggetto appartenga a più tipi tra loro non collegati — ad esempio, se gli studenti-lavoratori non hanno caratteristiche proprie, lo schema non deve necessariamente contenere un tale nuovo tipo, ma a un oggetto deve essere permesso di essere al tempo stesso sia studente sia lavoratore.