Riflessione Computazionale

Un sistema **riflessivo** contiene strutture che rappresentano aspetti di se stesso e che permettono al sistema di supportare azioni su se stesso.

Le azioni sono effettuate per mezzo dei meccanismi di **introspezione** e **intercettazione**. Fornire questi meccanismi significa fornire **reificazione**.

Un sistema riflessivo è strutturato in due o più livelli.

- Le entità al primo livello (**baselevel**) sono chiamate entità base
- Le entità al secondo livello (**metalevel**) sono chiamate meta entità
- Ogni livello è un baselevel per il successivo (eccetto l'ultimo), ed un metalevel per il precedente (eccetto il primo)

Connessione tra entità base e meta:

- Le meta entità supervisionano il lavoro delle entità base tramite **intercettazione** delle azioni di queste ultime
- Ogni azione è catturata dalla meta entità che esegue delle operazioni e può permettere all'entità base di eseguire l'azione

Caratteristiche dei sistemi riflessivi:

- Transparency: le entità di un livello non conoscono le entità del livello superiore
- Separation of concerns: ogni livello è dedicato ad un differente aspetto del sistema (funzionalità, non-funzionalità)

Modelli Riflessivi - 1

Per la programmazione orientata agli oggetti esistono 4 modelli riflessivi: meta-classe, meta-oggetto, reificazione di messaggio e reificazione di canale.

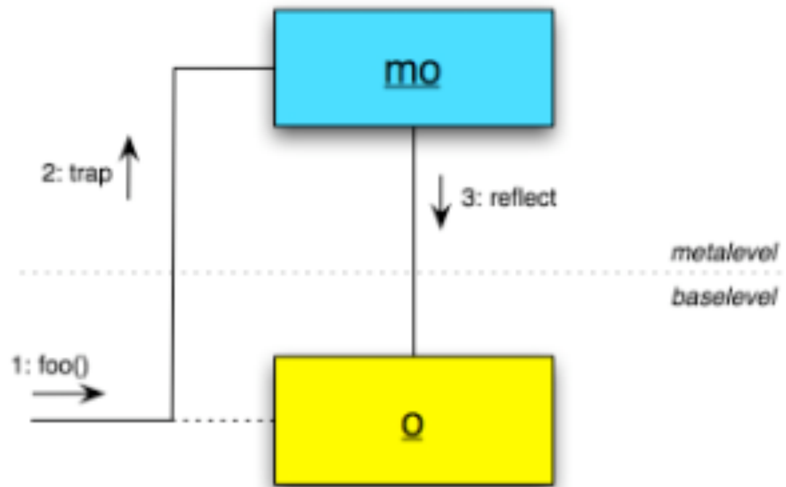
Modello **meta-classe**

- Ogni entità base (oggetto) è legata ad una meta entità che è la sua classe
- Una meta-meta entità è una meta classe
- E' implementabile direttamente solo nei linguaggi che gestiscono le classi come oggetti (Smalltalk, CLOS)

Modello **meta-oggetto** (è il modello di gran lunga più diffuso)

- Ogni entità base (classe, oggetto) è legata ad una o più meta entità (classe meta-oggetto)
- I meta-oggetti (nella figura mo) sono istanze di una classe speciale, chiamata `MetaObject`, o di una delle sue sottoclassi
- Una volta che un metaoggetto è associato ad un oggetto, tutte le operazioni sull'oggetto vengono intercettate dal metaoggetto
- Il metaoggetto può eseguire delle elaborazioni prima dell'esecuzione dell'operazione chiamata sull'oggetto e dopo dare il controllo all'oggetto. Analogamente alla fine dell'operazione dell'oggetto il controllo passa al metaoggetto.
- Il metaoggetto può essere pensato come un wrapper.

Modelli Riflessivi - 2



Per l'implementazione di sistemi che usano il modello meta-oggetto è necessario disporre di linguaggi che forniscono i meccanismi di introspezione e intercettazione.

Modello **Reificazione di messaggio**

- Le meta entità sono chiamate messaggi
- Una chiamata ad un metodo è reificata in un messaggio
- Un messaggio viene creato quando un metodo è chiamato e poi è distrutto

Modello **Reificazione di canale**

- Un canale è costituito dalla tripla (trasmettitore, ricevitore, tipo di canale) ed è stabilito quando un oggetto richiede un servizio
- Un canale è reificato in una meta entità chiamata canale
- La meta entità persiste dopo la computazione al meta livello

Riflessione in Java - 1

Java è un linguaggio che fornisce un supporto per la riflessione, le librerie `java.lang` e `java.lang.reflect` offrono la capacità di ispezionare classi ed oggetti.

Tramite le librerie di Java possiamo: scoprire campi metodi e costruttori presenti in una classe (che non conosciamo al momento della compilazione); istanziare una classe (il cui nome non è inserito all'interno del codice); invocare metodi, ispezionare e cambiare il contenuto di campi che scopriamo a run-time; ed avere superclasse ed interfacce di una classe.

La classe di partenza per usare la riflessione è `java.lang.Class.` Istanze di `Class` rappresentano classi e interfacce di una applicazione Java.

```
System.out.println("La classe di " + obj +
                   " e' " + obj.getClass().getName());
```

La classe `Object` fornisce il metodo `getClass()` che ritorna una istanza di `Class` che rappresenta la classe dell'oggetto.

Una istanza di tipo `Class` possiamo ottenerla attraverso:

```
Class c = Class.forName(“miaClasse”)
```

abbiamo usato il metodo statico `forName()` che restituisce il descrittore di tipo `Class` di una classe specificata tramite stringa.

Un oggetto istanza di `Class` permette di avere tutte le informazioni riflessive riguardo una classe, tipo: superclasse, interfacce implementate; ed anche costruttori, campi e metodi definiti dalla classe.

Riflessione in Java - 2

Alcuni metodi di `Class`

`getConstructors()`

restituisce un array con tutti i costruttori pubblici della classe rappresentata dall'oggetto su cui si invoca. Il valore restituito è una istanza di `java.lang.reflect.Constructor`

```
Constructor cons = cl.getConstructors()
```

`getFields()` (con la variante `getDeclaredFields()`)

in modo simile alla lista di costruttori, restituisce la lista dei campi pubblici dell'istanza di `Class`. Il valore di ritorno è una istanza di `java.lang.reflect.Field`

`getMethods()` (con la variante `getDeclaredMethods()`)

restituisce tutti i metodi pubblici di una classe. Il valore di ritorno è di tipo `java.lang.reflect.Method`

`getSuperclass()`

restituisce la `Class` che rappresenta la superclasse della `Class` su cui è invocata.

`newInstance()`

restituisce una nuova istanza della classe di cui viene fornito il nome come parametro stringa.

```
Object o = Class.forName("miaClasse").newInstance()
```

Permette di istanziare una classe non conosciuta al momento della compilazione.

Riflessione in Java - 3

Le classi della libreria `java.lang.reflect: Constructor, Field e Method` completano il supporto all'ispezione di Java.

La classe `Constructor` fornisce, tra gli altri, il metodo:

`newInstance()`

> prende come argomento un array di oggetti ed usa l'oggetto costruttore per creare ed inizializzare una nuova istanza della classe che contiene quel costruttore, con i parametri specificati.

La classe `Field` fornisce i metodi

`getType()`

> restituisce un `Class` che identifica il tipo del campo

`get(), getByte(), getInt(), getFloat(), ...`

> restituiscono il valore che quel campo contiene per l'oggetto che viene passato come argomento.

`set(), setByte(), setInt(), setFloat(), ...`

> ciascuno dei metodi assegna al campo su cui si invoca e per l'oggetto che viene passato come primo argomento il valore passato come secondo argomento.

La classe `Method` fornisce il metodo

`invoke()`

> serve ad invocare un metodo su una istanza. Prende in ingresso due parametri: l'istanza della classe e un array di parametri per la chiamata.

Riflessione in Java - 4

Vantaggi nell'uso della riflessione in Java:

- Fornisce un modo per collegare ad un programma nuove classi, non conosciute a compile time.
- Permette di manipolare oggetti di una qualsiasi classe senza inserire nel codice la classe, quindi rinviando il binding fino a run-time.

Svantaggi:

- Invocare metodi o accedere a campi con i meccanismi riflessivi è molto più lento che col codice diretto. Ma se l'uso della riflessione all'interno di un programma è limitato, l'overhead è sostenibile.

  In una prova effettuata, l'esecuzione di un ciclo che accede un campo direttamente impiega 170 ms. L'esecuzione dello stesso ciclo quando si accede al campo con la riflessione (con il metodo `getInt()` di `Field`) impiega 63500 ms.

  In pratica, la seconda esecuzione è 373 volte più lenta della prima.

- Capire cosa sta facendo un programma leggendo il codice diventa molto più difficile, poiché il codice è più complesso.

Alcuni Riferimenti

Computational Reflection

B. C. Smith. Reflection and Semantics in LISP. In *Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. Salt Lake City, Utah, United States. 1984

P. Maes. Concepts and Experiments in Computational Reflection. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'87),* volume 22 (12) of *Sigplan Notices*, pages 147–155, Orlando, FA, 1987.

J. Ferber. Computational Reflection in Class Based Object Oriented Languages. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'89),* volume 24 of *Sigplan Notices*, pages 317–326, New York, NY, 1989.

W. Cazzola. Evaluation of Object-Oriented Reflective Models. In *Proceedings of the ECOOP'98 Workshop on Reflective Object-Oriented Programming and Systems*. Brussels, Belgium. 1998.

Sistemi Software Riflessivi

R. J. Stroud and Z. Wu. Using Metaobject Protocols to Satisfy Non-Functional Requirements. In C. Zimmermann, editor, *Advances in Object-Oriented Metalevel Architectures and Reflection*. CRC Press, 1996.

E. Tramontana. Managing Evolution Using Cooperative Designs and a Reflective Architecture. In W. Cazzola, R. J. Stroud, and F. Tisato, editors, *Reflection and Software Engineering*, volume 1826 of *LNCS*. Springer-Verlag, June 2000.

A. Di Stefano, G. Pappalardo and E. Tramontana. Introducing Distribution into Applications: a Reflective Approach for Transparency and Dynamic Fine-Grained Object Allocation. In *Proceedings of the Seventh IEEE Symposium on Computers and Communications (ISCC'02)*. Taormina, Italy. 2002.