OOLibrary

venerdì 19 novembre 2010
11:49

Obbiettivo

Avere una sola libreria che consenta, quando si utilizza la versione di tuProlog per .NET, di accedere sia agli oggetti Java che a quelli .NET.
In questo modo chi sviluppa la versione .NET non dovrebbe sempre rincorrere la versione Java, ma si dovrebbe occupare soltanto delle convenzioni.
Questo porterebbe anche al grande vantaggio di utilizzare, contemporaneamente tramite Prolog, sia librerie scritte in Java (file .class/.jar) che scritte in .NET (file .dll/.exe).
Questo è di fatto già possibile compilando la JavaLibrary con IKVM ma vi è il problema della convenzioni (vedi IKVMJavaLibrary vs CLILibrary).

Comportamento desiderato

Deve essere possibile:

1. Caricare e scaricare convenzioni per i diversi linguaggi;
2. Associare un oggetto ad una convenzione;
3. Utilizzare metodi, property o campi in base alla convenzione del linguaggio in cui è stata scritta la classe senza preoccuparsi delle modifiche fatte dal compilatore;
    a. nel caso in cui non vi sia associata nessuna convenzione all'oggetto si cercherà il nome del membro così come viene specificato (questo permette di utilizzare comunque oggetti anche se non si dispone della convenzione opportuna, a patto però di conoscere esattamente i nomi da utilizzare)
    (esempio: per utilizzare oggetti Java non servirebbe nessuna convenzione).

Idea

L'idea è quindi quella di ridefinire, in qualche modo (vedi sotto), il metodo java_call_3(...) per fare in modo che capisca se si vuole accedere ad un metodo , ad una property o ad un campo e modificare di conseguenza il nome.
Due possibili approcci:

1. Avere "qualcosa" in più già nella versione Java ma che di fatto viene utilizzata solo nella versione .NET:
   Nell'applicare questa soluzione si possono avere due approcci:
    a. **JavaLibrary come black box**: si lavora come se non si avesse a disposizione il codice della JavaLibrary e quindi si aggiungono i comportamenti particolari (un/loadConv, conversione nomi,…) mediante subclassing o proxy.
       PRO:
        i. Non si va a complicare ulteriormente il codice della JavaLibrary
       CONTRO:
        i. Scrivere le convenzioni per le property risulta un po' più difficile dato che non si possono ridefinire i metodi java_get() e java_set() che sono privati e quindi bisogna fare tutto ridefinendo il metodo java_call().
    b. **Modifica del codice della JavaLibrary:**
       PRO:
        i. Si possono ridefinire i metodi java_set() e java_get() e ci si semplifica la vita con le property (vantaggio insignificante)
       CONTRO:
        i. Si complica notevolmente il codice, già non banale, della JavaLibrary.
   Pensandoci bene questo tipo di soluzione non fa altro che complicare la versione Java di tuProlog perché aggiunge il concetto di Convenzione che in realtà "non centra niente" con il mondo java.
   Inoltre non è applicabile perché nel momento in cui l'utente cerca di accedere ad una property il sistema deve capire se la property esiste e modificare il nome di conseguenza, per fare questo controllo serve il sistema di Reflection del framework .NET.
2. Aggiungere "qualcosa" in più solo nella distribuzione della versione .NET:
    a. **Proxy:**
       In questa soluzione si crea una classe .NET (OOLibrary) che definisce la stessa interfaccia della JavaLibrary e che possiede un riferimento alla JavaLibrary.
       Quindi la OOLibrary è una sorta di proxy per la JavaLibrary.
       La OOLibrary aggiunge i predicati/metodi per il caricamento e scaricamento delle convenzioni.
       I metodi della OOLibrary utilizzano le convenzioni per determinare il nome del membro da utilizzare e poi invocano i corrispondenti metodi della JavaLibrary.
       PRO:
        i. La JavaLib non viene modificata (non aggiungo nella versione Java il concetto di Convenzione che non avrebbe molto senso);
        ii. Eventuali modifiche al codice della JavaLibrary vengono utilizzate in automatico
       CONTRO:
        i. La modifica dell'interfaccia della JavaLibrary non viene utilizzata in automatico (occorre scrivere i nuovi metodi sulla OOLIbrary)
        ii. Gli utenti della versione .NET dovranno caricare una nuova libreria per utilizzare le convenzioni.
    b. **Subclassing:**
       In questa soluzione la classe OOLibrary deriva da JavaLibrary, aggiunge i predicati/metodi per caricare e scaricare le convenzioni e ridefinisce i metodi come java_call per utilizzare le convenzioni (comunque poi richiama i metodi della sopraclasse).
       PRO:
        i. La JavaLib non viene modificata (non aggiungo nella versione Java il concetto di Convenzione che non avrebbe molto senso);
        ii. Eventuali modifiche al codice della JavaLibrary vengono utilizzate in automatico
        iii. La modifica dell'interfaccia della JavaLibrary viene utilizzata in automatico ma senza l'utilizzo delle convenzioni
        (esempio: se nella JavaLibrary viene aggiunto un nuovo predicato/metodo la OOLibrary ovviamente ottiene in automatico questo metodo ma non utilizza le convenzioni, andrà sempre ridefinito)
       CONTRO:
        i. Gli utenti della versione .NET dovranno caricare una nuova libreria per utilizzare le convenzioni.

Problema nomi predicati

La JavaLibrary è stata scritta in Java per il modo Java quindi i predicati hanno tutti i prefisso "java".
La OOLibrary invece è scritta in C# ma permette di utilizzare oggetti scritti in qualunque linguaggio .NET e anche in Java. Quindi sarebbe opportuno modificare i prefissi dei predicati, magari in "oo".
Indipendentemente dal prefisso scelto il problema del cambio di prefisso mi sembra una questione non da poco, almeno filosoficamente, dato che non è bellissimo utilizzare predicati come java_object, java_array,… quando in realtà si utilizzano oggetti scritti in tutt'altro linguaggio.
Inoltre se si vuole cambiare il prefisso si perde la possibilità di sfruttare il subclassing, perché non si potrebbe cambiare il nome dei metodi.
Si potrebbero cambiare i nomi dei predicati nella teoria restituita dalla libreria e lasciare i nomi dei metodi con "java" davanti ma questo non credo che cambi le cose dato che i nomi dei predicati che si possono utilizzare dipendono solo dai nomi dei metodi, **GIUSTO?**

Curiosità su questo punto: perché non avete utilizzato le annotazioni. Cioè ogni metodo che deve essere invocato quando si cerca di risolvere un predicato ha un'annotazione in cui vi è il nome del predicato. In questo modo si ha forse un po' più di libertà: il predicato e il relativo metodo non devono avere lo stesso nome.

Problema property

Come accennato, utilizzando la IKVMJavaLibrary, per accedere alle property bisognerebbe utilizzare la stessa sintassi che si utilizza quando si invoca un metodo.
Esempio: *Obj <- 'Prop' returns Nome* / *Obj <- 'Prop'("NewValue")*
La OOLibrary si occuperebbe solo di modificare il nome in base alla convenzione.

In realtà ridefinendo in modo opportuno il metodo java_call() è anche possibile supportare la sintassi tipica della CLILibrary (Oggetto.Property).
È un po' più complicato ma possibile ed è probabilmente la scelta migliore.

Considerazioni

L'unica soluzione applicabile è la **2**.
La scelta tra **a** o **b** dipende da come si vuole trattare la questione sui nomi dei predicati.