Introduzione alla tecnologia JMX[1]

Viene analizzata l'architettura sottostante le Java Managment Extensions (JMX) mostrandone un utilizzo applicativo e analizzando altri possibili scenari d'uso

di Ivan Venuti (ivanvenuti@yahoo.it)

Le *Java Management Extensions* (d'ora in poi JMX) sono, come idea generale, la specifica di un repository di oggetti Java, con delle ben definite proprietà al fine di garantire una infrastruttura (un insieme di API e dei servizi) per l'accesso sia locale che remoto a tale repository.
Gli oggetti Java nel repository rappresentano le caratteristiche delle risorse da gestire (*resource management*); tali risorse possono essere software (di solito le applicazioni che vengono sviluppate) ma anche hardware.
In questo articolo viene spiegata l'architettura sottostante la tecnologia JMX e viene mostrato il suo uso in un'applicazione d'esempio, cercando anche di capire come e quando una tecnologia del genere può essere utile.

Le esigenze applicative

Monitore e configurare: cosa si intende di preciso con questi termini?
Per monitorare si può intendere il verificare, in maniera continua o a intervalli regolari, lo stato di un'applicazione durante il suo utilizzo. In particolare si potrebbe pensare di verificare se l'applicazione, o alcune sue parti, risponde correttamente; un esempio di monitoraggio è connettersi ad una porta http se l'applicazione fornisce un servizio Web, oppure tentare di accedere ad un database dal quale l'applicazione reperisce i dati o accedere ad un file system condiviso dove vengono memorizzate alcune informazioni. Tutti questi esempi si riferiscono o a risorse esterne o a funzionalità verificabili dall'esterno. Eppure ci possono essere casi in cui è l'applicazione stessa a fornire dei dati che permettono di monitorarne lo stato (quanta memoria è disponibile nella Virtual Machine su cui è in esecuzione, quante sessioni sono attive e così via) o le prestazioni (come tempo medio di risposta).
JMX si può usare anche per configurare le applicazioni, dove per configurazione si intende qualunque azione che modifica l'esecuzione del programma o ne fissa alcune caratteristiche (livello di log, massima dimensione di pool di oggetti e così via); in questi casi è necessario porsi alcune domande; in particolare, impostando i valori dei parametri:

- L'applicazione cambia subito comportamento, utilizzando i nuovi valori, o è necessario aspettare un reload esplicito della stessa?
- I nuovi valori sono resi persistenti? Se sì, attraverso comandi espliciti?
- I parametri sono solo di lettura, solo di scrittura o entrambi?

Quando si sa chiaramente come si deve comportare l'applicazione si può analizzare una possibile soluzione applicativa.

1 Questo articolo è stato pubblicato su *Linux Journal Edizione Italiana* N. 25, Aprile 2002 edito dalla Duke Italia (http://www.duke.it); quest'ultima detiene i diritti di pubblicazione e utilizzazione economica di quest'articolo; l'autore è stato autorizzato a renderlo pubblico sul proprio sito Web personale.

Il problema della configurazione

Quando si scrive un'applicazione, la sua configurazione può avvenire:

- in fase di installazione (attraverso vari file di properties, variabili d'ambiente e così via) e, di solito, la configurazione viene letta durante l'avvio dell'applicazione stessa
- attraverso un meccanismo di configurazione creato ad hoc (console grafica, tool di configurazione accessibili dal Web, …); molte volte esiste un meccanismo per il "reload" dei nuovi valori dei parametri o essi vengono reperiti dinamicamente

è chiaro che entrambe le strade hanno grossi limiti nel caso in cui si voglia creare un modo per gestire e modificare la configurazione di più applicazioni. In questi casi è difficile integrare i diversi approcci in quanto chiunque crea un meccanismo di configurazione lo fa in maniera diversa dall'altro e senza seguire uno standard.

Management delle risorse

Con il concetto di "management delle risorse" si intende qualcosa di più della semplice configurazione: si vorrebbero reperire informazioni anche sullo stato dell'applicazione stessa (dove per stato si può intendere sia se l'applicazione risponde correttamente sia se si sono verificati errori o se sono stati incontrati problemi) e altre informazioni o statistiche che riguardino il suo funzionamento (performance dei servizi offerti, numero di richieste, accessi negati per mancanza di autorizzazione e molti altri aspetti).
È chiaro che per poter fornire questo tipo di informazioni sono necessari meccanismi di comunicazione dei risultati come log su file system o su database, console di monitoraggio e così via. Anche in questo caso, come per il problema della configurazione, ogni applicazione si costruisce ad hoc propri meccanismi e modalità di erogazione di queste informazioni rendendo impossibile (o per lo meno molto difficile) l'integrazione tra le diverse applicazioni.

L'architettura JMX

Per rispondere ai problemi di configurazione, gestione e, in senso più ampio, "management delle risorse" nasce una proposta che va sotto il nome di Java Management Extensions.
L'uso della tecnologia JMX non è particolarmente invasivo nelle applicazioni, anche se è bene prevederne l'uso il prima possibile in maniera da orientarne lo sviluppo. Gran parte della complessità è "mascherata" dall'uso di componenti infrastrutturali standard e dalla presenza di semplici regole con cui esporre certe funzionalità, come la configurazione e il monitoring, dell'applicazione.
JMX ha un'architettura a tre livelli. Il livello più basso, chiamato *Instrumentation Level*, è composto dagli oggetti che l'applicazione mette a disposizione per essere gestiti. Questi oggetti, chiamati *MBean* (contrazione per "*Managed Bean*") sono dei JavaBean contenenti metodi di accesso agli attributi (pertanto espone i metodi *Setter* e *Getter* per ogni attributo che è, rispettivamente, in scrittura e lettura) e, in più rispetto ai JavaBean "tradizionali", devono implementare un'apposita interfaccia.
Oltre ad eventuali MBean, l'applicazione può predisporre un *Notification Model* e delle classi chiamate *MBean Metadata*. Analizzeremo in seguito degli esempi dettagliati.

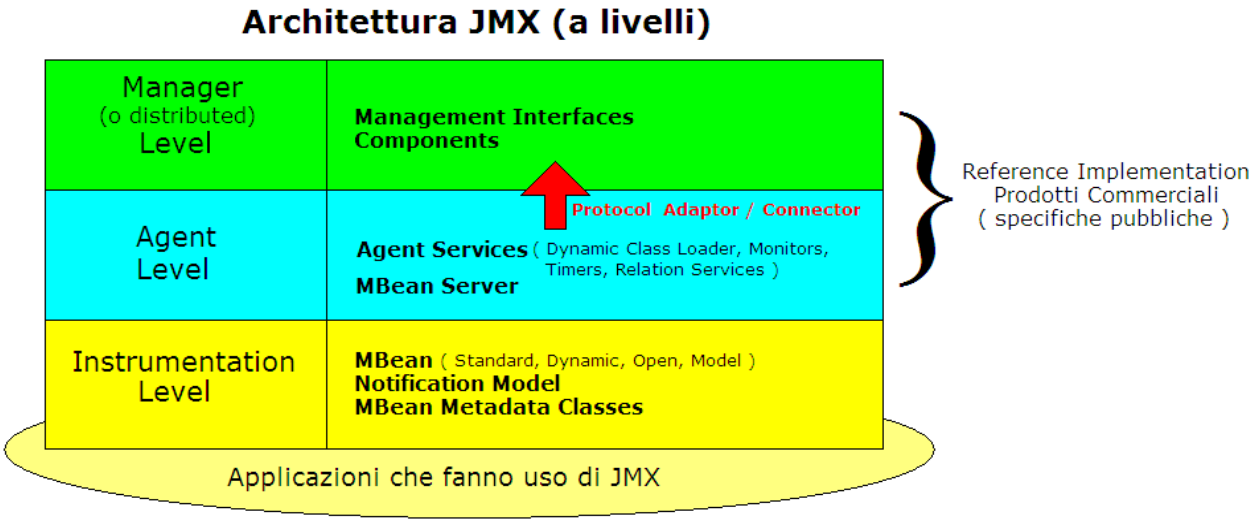

Figura 1: L'architettura a livelli di JMX

I livelli superiori, rispettivamente *Agent Level* e *Manager Level*, sono infrastrutturali e, come tali, non di diretta competenza dell'applicazione ma forniti da appositi framework. Tra questi framework di sicuro interesse è la reference implementation proposta da Sun (integrata nel JSE5.0 o reperibile all'url http://java.sun.com/products/JavaManagement/download.html) ma possono essere utilizzati prodotti commerciali oppure Open Source (tra i progetti più stabili si segnala MX4J, scaricabile dalla pagina http://mx4j.sourceforge.net); ogni framework è "intercambiabile", purché soddisfi le specifiche rilasciate da Sun (che sono pubbliche).
Come vedremo realizzando un esempio, il fatto che le funzionalità infrastrutturali vengano fornite da framework esterni semplificherà l'adozione della tecnologia, permettendo, in breve tempo, di realizzare applicazioni JMX complete, che utilizzino tutte le componenti architetturali descritte.
Analizziamo nel dettaglio le componenti dei diversi livelli dell'architettura JMX, realizzandoun MBean a cui si accederà da una console HTML utilizzata da un browser Web (**figura 2**).

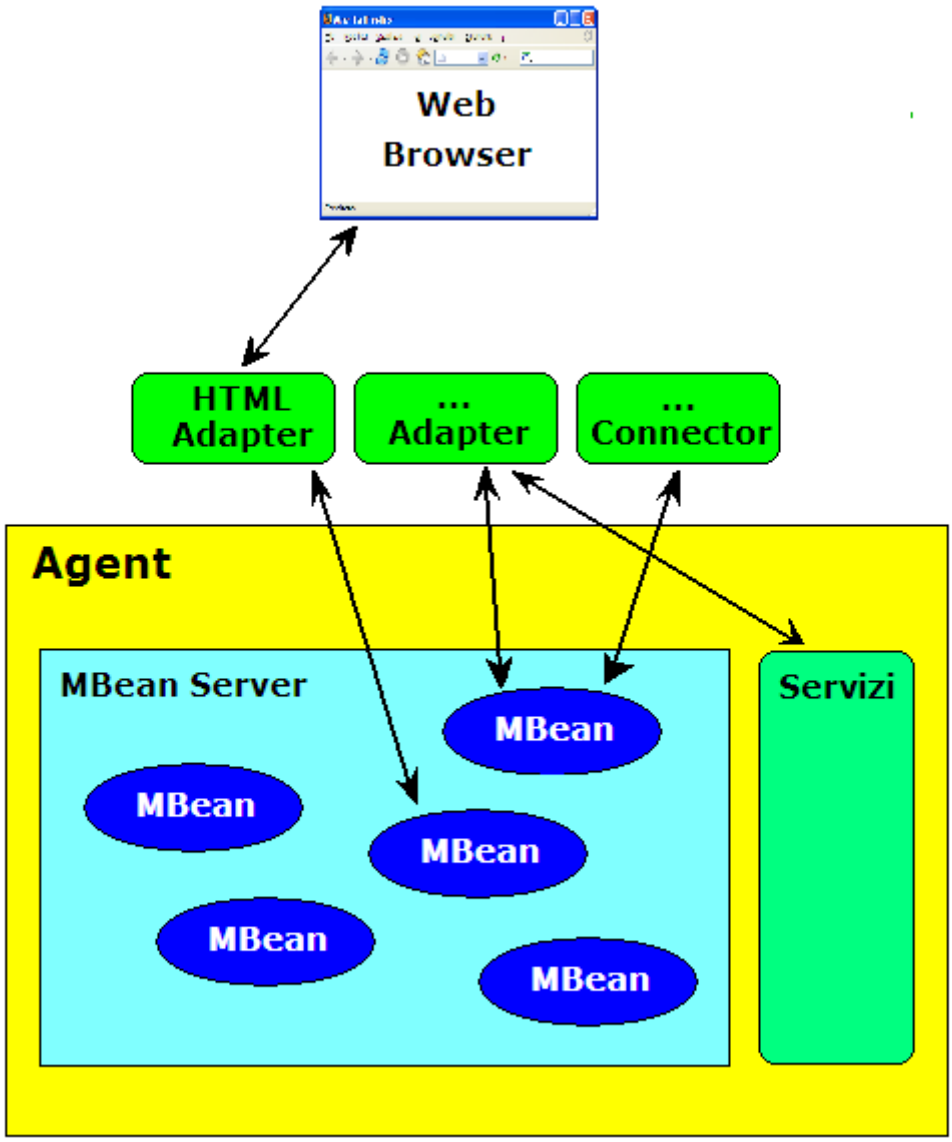

Figura 2: schema dell'esempio realizzato

MBean

Un MBean può essere composto da un'interfaccia e da una classe che la implementa. L'interfaccia espone attributi e operazioni pubblici; la classe che la implementa definisce i costruttori (pubblici) e tutti i suoi attributi e operazioni (pubblici e non).
In particolare:

- Costruttori: permettono di instanziare l'MBean
- Attributi: sono nella forma dei *setter* e *getter*: `setAttributo(valore)`, `getAttributo()`
- Operazioni: altri tipi di metodi che non fanno parte dei precedenti

L'invocazione di un metodo avviene:

```
mbeanServerConnection.method(mbeanID,"nomeDelMetodo",...)
```

Un Mbean può essere di due tipi: *Standard* o *Dynamic*. Gli MBean del primo tipo sono quelli più semplici; vengono riconosciuti attraverso un meccanismo di reflection sui nomi dei metodi, gli altri indicano come pubblicare i loro metodi e, cosa non possibile per Mbean Standard, possono pubblicare anche una descrizione per ogni metodo esposto.
Gli MBean di tipo Dynamic possono essere, a loro volta, di diverse tipologie: gli *Open MBean* hanno delle convenzioni che ne limitano la potenza ma li rendono estremamente portabili; in *Model MBean*, invece, si usa una factory per istanziare un nuovo MBean da utilizzare. A questo punto si "costruisce" (dinamicamente) le caratteristiche volute dell'MBean: in pratica esso non possiede un proprio file sorgente java, ma viene sviluppato al run-time.
In ogni modo, per esigenze non troppo particolari, gli MBean di tipo *Standard* sono quelli che soddisfano a tutte le necessità. Per questo motivo viene descritto solo questo tipo di oggetti.

Un esempio di MBean Standard

Supponiamo di voler rendere configurabili le connessioni ad un database da parte di un'applicazione. È presumibile che l'applicazione debba memorizzare le informazioni "atomiche" relative a tale connessione come:

- Nome del driver JDBC
- Url di connessione

Creiamo un MBean che offre due attributi, uno per ogni informazione da trattare; inoltre potremmo pensare ad un attributo (di sola lettura) che indica se è possibile reperire correttamente una connesione (attributo correct). Infine si possono realizzare un certo numero di operazioni da utilizzarsi sul db. A titolo esemplificativo creiamo un'operazione che ritorna il numero di righe restituite da una select, dove il comando per la select è passato come parametro al metodo.
L'interfaccia che definisce l'MBean sarà:

```
package it.linuxjournal.jmx;

public interface esempioMBean {
      public void setDriver(String name);
      public String getDriver();

      public void setUrl(String name);
      public String getUrl();

      public boolean isCorrect();

      public int numRowsSelect(String command)  throws Exception;
}
```

Non resta che implementare i diversi metodi; per prima cosa vanno inseriti gli attributi:

```
public class esempio implements esempioMBean{
      private String driver;
      private String url;
      private boolean correct = false;

      // . . .
}
```

I setter e i getter sono facili da realizzare:

```
public void setDriver(String name) {
      driver = name;
}

public String getDriver() {
      return driver;
}

public void setUrl(String name) {
      url = name;
}

public String getUrl() {
      return url;
}
```

ecco invece come realizzare le operazioni e il getter di “correct”:

```
public boolean isCorrect() {
      try{
            Class.forName(driver).newInstance();
            Connection con = DriverManager.getConnection(url);
            con.close();
            correct = true;
      }catch(Exception e){
            correct = false;
      }
      return correct;
}

public int numRowsSelect(String command) throws Exception {

      Connection con = null;
      Statement stmt = null;
      try {
            con = DriverManager.getConnection(url);
            stmt = con.createStatement();
            if (stmt.execute(command)) {
                  ResultSet rs = stmt.getResultSet();
                  rs.last();
                  int size = rs.getRow();
                  rs.close();
                  return size;
            }
            return 0;
      } finally {
      // chiude lo statement e la connessione . . .
      }
}
```

MBean Server

Una volta realizzati gli MBean voluti, è necessario aggiungerli ad un apposito repository, chiamato *MBean Server*. Ogni Mbean deve avere associato un nome, univoco, nella forma "`NomeDominio:key1=val1,key2=val2`".
Di solito un server MBean viene creato a partire da una Factory.

```
server = MBeanServerFactory.createMBeanServer();
```

Una volta creato si possono registrare gli Mbean:

```
ObjectName objName = new ObjectName("dominnio:chiave=valore");
server.registerMBean( qualeClasse.newInstance(), objName);
```

Agenti JMX

Un agente JMX è un oggetto che ha diverse funzioni; in particolare:

- È un container per un MBean Server;
- Fornisce funzionalità per creare relazioni tra MBean;
- Carica dinamicamente le opportune classi
- Fornisce servizi per il monitor e dei timer
- Dispone di un insieme di *protocol adapters* (adattatori) e *connectors* (connettori) che permettono a client remoti di accedere all'agente

Di seguito viene mostrato come realizzare l'agent JMX (che contiene un MBean Server e un adattatore per il protocollo HTML) utilizzando i package forniti dalla Sun:

```
package it.linuxjournal.jmx;

import javax.management.*;
import com.sun.jdmk.comm.HtmlAdaptorServer;

public class LinuxJournalAgent {

  public static void main(String[] args) {
    MBeanServer server = null;
    try {
      // Crea il Server Mbean
      server = MBeanServerFactory.createMBeanServer();
      // registra gli MBean
      try {
       Class quale = Class.forName("it.linuxjournal.jmx.esempio");
       ObjectName name = new
         ObjectName("dominio:classe=it.linuxjournal.jmx.esempio");
       server.registerMBean(quale.newInstance(), name);
       // . . .
      } catch (Exception e) {
         e.printStackTrace();
      }
      // registra l'html adaptor
```

```
        HtmlAdaptorServer adaptor = new HtmlAdaptorServer();
        ObjectName name = new ObjectName("adaptor:protocol=HTTP");
        server.registerMBean(adaptor, name);
        adaptor.start();
      } catch (Throwable th) {
        th.printStackTrace();
      }
    }
}
```

L'Agent può essere eseguito come applicazione stand-alone.
Ovviamente per poter funzionare è necessario che sul CLASSPATH del programma in esecuzione sia specificato il driver impostato sull'attributo dell'MBean (siccome il driver viene caricato al run-time, la compilazione va sempre bene, mentre può fallire l'esecuzione se tale driver non viene trovato).

Monitor via Web

Mandando in esecuzione l'agent appena descritto, esso fornirà l'accesso via http sulla macchina ove viene eseguito alla porta 8082 (che è la porta di default, ma è possibile personalizzarla). In **figura 3** si può osservare l'interfaccia presentata all'url http://localhost:8082 (punto 1); dalla figura si può notare come l'implementazione dell'adaptor permetta di specificare un filtro sugli oggetti da mostrare a video (punto 2); nell'esempio tale filtro non esclude alcun oggetto. Seguendo i link contrassegnati con 3 e 4 si può accedere, rispettivamente, all'adaptor e all'unico MBean registrato.

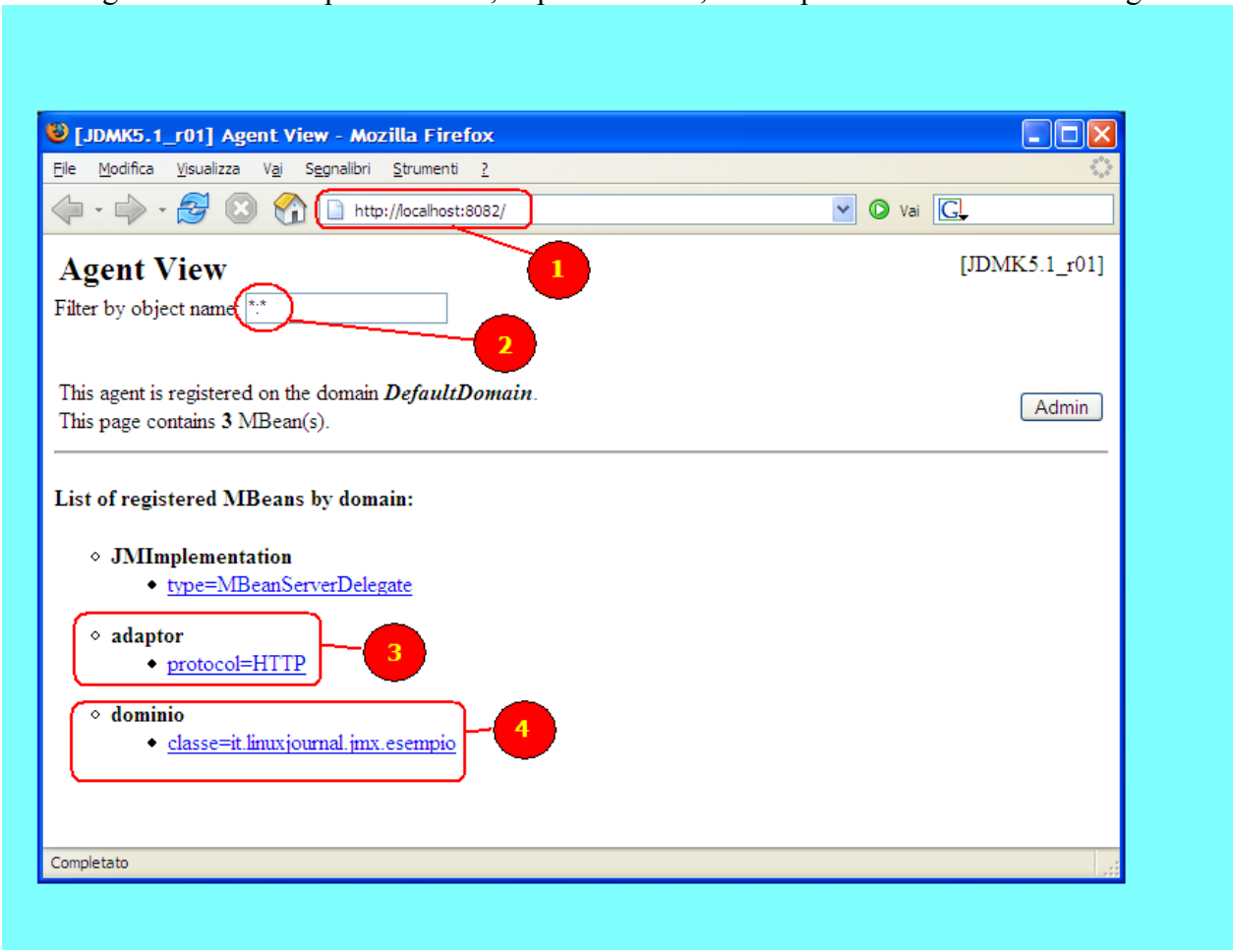

Figura 3: L'agent di esempio fornisce l'accesso all'MBean Server attraverso un connettore http

Accedendo all'MBean è possibile verificare il nome con cui è stato censito (1) e la classe caricata (2). Da questa pagina è possibile verificare quali attributi (3) e quali operazioni (6) sono esposti dall'MBean. Gli attributi in sola lettura presentano solo un campo di testo non modificabile (3); gli attributi che sono sia in lettura che in scrittura permettono di impostare nuovi valori (4). Per assegnare effettivamente un nuovo valore non basta scriverlo nel campo corrispondente, ma bisogna applicare l'assegnazione agendo sul tasto "Apply" (5).

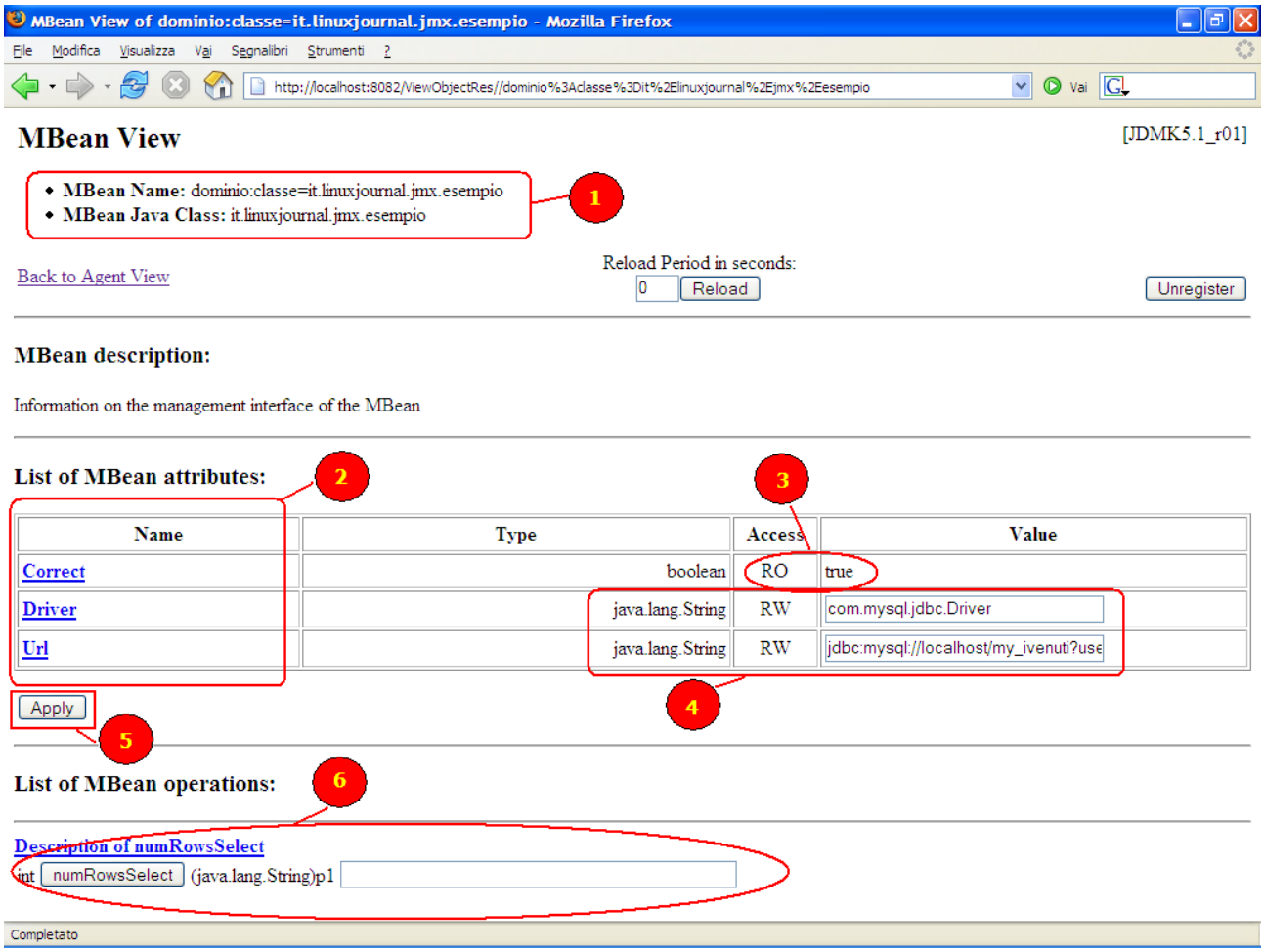

Figura 4: attributi e operazioni dell'MBean di esempio.

Estendere l'agent

È probabile che l'agente JMX creato poc'anzi sia poco flessibile: è necessario intervenire sul codice per ogni nuovo MBean inserito.
Comunque l'operazione di generalizzazione è piuttosto semplice. La prima strada consiste nel far leggere all'agente JMX gli MBean come parametri passati alla linea di comando dell'applicazione. In questo modo basterà far ripartire l'agent aggiungendo le nuove classi MBean.
Ancora più flessibile è la soluzione di dotare tale agent di una console che legge i comandi e li interpreta per registrare/de-registare nuovi MBean senza dover fermare e far ripartire l'agent stesso.

Conclusioni

Si è visto come è semplice e immediato realizzare dei componenti MBean che facciano uso dell'architettura JMX. Tale architettura merita di essere considerata per qualsiasi nuova applicazione Java, in quanto rende flessibile l'intero processo di configurazione e di monitoraggio, offrendo la possibilità di creare console unificate per più applicazioni, create anche da diversi partner commerciali.

Bibliografia

[1] "*Java Managment Extensions*", http://java.sun.com/jmx
[2] "*JMX Accelerated Howto*", B. Simpson, http://admc.com/blaine/howtos/jmx