**REMOTE METHOD INVOCATION – RMI**

*In questo capitolo sono illustrate le scelte effettuate per la realizzazione di una versione dell’applicazione basata su RMI.*

**Introduzione**

Per lo sviluppo dell’applicazione si ha la necessità di richiedere dei servizi presenti in oggetti che risiedono in processi diversi, eventualmente anche in un altro calcolatore. Per raggiungere tale scopo è stata usata la tecnologia Java RMI – Remote Method Invocation

**JAVA RMI**

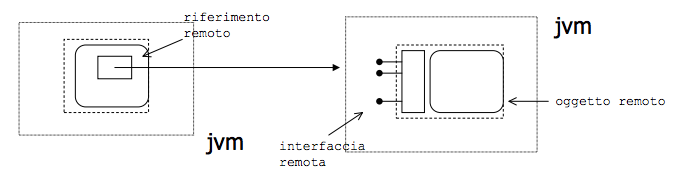
**Java Remote Method Invocation** (**RMI**) è l’implementazione Java del modello ad oggetti distribuiti che fornisce un’astrazione di programmazione distribuita orientata agli oggetti. In prima istanza, secondo questo modello, esiste una collezione di oggetti distribuiti che possono risiedere in processi diversi. Ogni oggetto può offrire servizi pubblici, mediante un’interfaccia, ed incapsulare uno stato privato. Gli oggetti cooperano mediante l’invocazione di metodi. In questo modello è possibile distinguere tra due tipi di oggetto: locali o remoti. Gli oggetti locali sono visibili localmente a un processo mentre quelli remoti possono essere distribuiti su più calcolatori o processi. Ciascun oggetto remoto implementa un’interfacci remota che specifica quali metodi possono essere invoca remotamente.

In Java RMI la sintassi per le invocazioni remote è esattamente la stessa di quella delle invocazioni locali:

* Possono essere passati argomenti, calcolati nel contesto della macchina chiamante;
* Possono essere restituiti valori, calcolati nel contesto della macchina remota,
* Gli elementi chiave sono serializzazione e dynamic class loading.

Come già accennato, in Java RMI tre sono i concetti fondamentali:

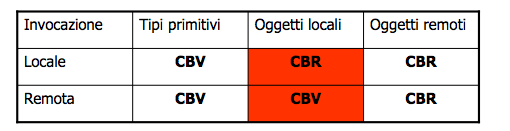
* **Interfaccia Remota**: interfaccia java che specifica i metodi che un oggetto rende visibile ad altri oggetti come metodi remoti;
* **Oggetto Remoto**: oggetto java che implementa un’interfaccia remota;
* **Riferimento Remoto**: riferimento ad un oggetto remoto.



La semantica di una invocazione remota non è uguale però a quella di una invocazione locale. I tipi di dati primitivi (int, float, …) sono sempre passati per valore (semantica Call By Value, CBV) ed il chiamato lavora su una copia del tipo. Gli oggetti remoti sono sempre passati per riferimento (Call By Reference, CBR). Il chiamato, in questo caso, riceve un riferimento all’oggetto.

[Se però il parametro è un oggetto locale, tale oggetto è passato CBV se parametro di un metodo remoto e CBR se parametro di un metodo locale.]

La tabella seguente riassume tutti i casi che si possono verificare indicando per ogni tipo di dato il tipo di chiamata.



Un aspetto importante nelle applicazioni distribuite è la separazione fra interfaccia resa visibile all’esterno e l’implementazione dell’interfaccia stessa. L’interfaccia viene definita mediante un linguaggio detto **Interface Definition Language**, IDL.

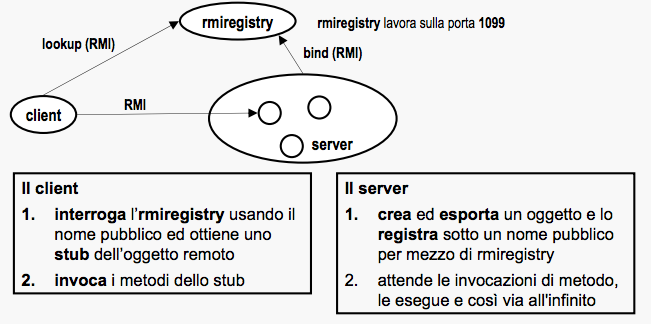
Il modo di operare tipico in Java RMI è il seguente:

* Si progetta l’interfaccia remota;
* Si implementa l’interfaccia remota in oggetti che assumeranno il ruolo di Serventi;
* Si implementa il Server;
* Si implementa il Client;
* Si avvia l’RMI compiler (rmic) per produrre lo Stub e lo Scheleton;
* Si avvia l’RMI Registry;
* Si registra il servizio nell’RMI Registry.

Il compito del Server è quello di istanziare l’oggetto servente, esportarlo come oggetto remoto registrandolo, attraverso un associazione con un nome simbolico, presso l’object registry (operazione **bind**).

L’RMI-Registry è un server RMI che gestisce le annotazioni nome simbolico – oggetto remoto.

Il client utilizzeranno il nome simbolico per interrogare il registry ed ottenere un riferimento remoto per l’oggetto (operazione **lookup**).

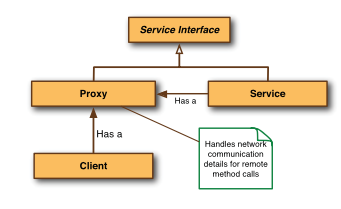


Il client quindi non colloquierà direttamente con l’oggetto remoto ma con un suo rappresentante chiamato **Stub**. Lato server esiste lo Skeleto che è il corrispondente dello Stub ed il cui compito è quello di accettare le comunicazioni dal client e tradurle in invocazioni.

**RMI E SPRING**

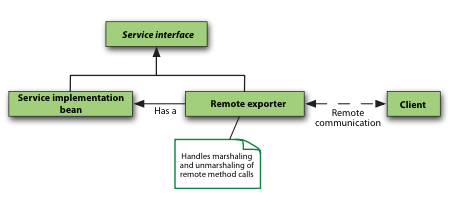
Dopo aver effettuato una veloce panoramica su Java RMI è necessario analizzare come Spring renda molto più semplice usufruire di un servizio esposto mediante RMI. Nell’ambito del lavoro di tesi sono state sfruttate proprio le facilitazioni messe a disposizione da Spring per integrare, all’interno di una applicazione web, dei servizi esposti mediante RMI.

Sviluppare ed accedere a un servizio RMI, come è stato spiegato poc’anzi, comprende un insieme di attività sia di programmazione che manuali. Molte azioni da compiere sono ripetitive e possono rappresentare delle fonti d’errore. Spring semplifica il modello di RMI fornendo un proxy factory bean che consente di collegare i servizi esposti in RMI all’interno dell’applicazione come se fossero Java Beans locali. La figura sottostante mostra come ciò sia possibile.

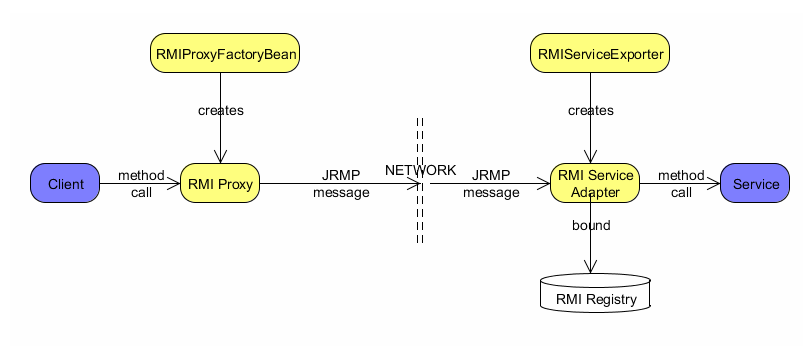


Dall’immagine si nota come il Client effettui delle chiamate al Proxy come fosse il Proxy stesso a fornire le funzionalità del servizio. Il Proxy quindi comunica con il servizio remoto per conto del client gestendo, inoltre, tutti i dettagli della comunicazione remota.

La figura sottostante mostra come il Remote Exporter esponga metodi di servizi remoti.



Il seguente diagramma d’interazione mostra, in modo più dettagliato, il lavoro svolto da Spring Framework. I riquadri in viola devono essere implementati dall’utente mentre gli elementi in giallo sono gestiti direttamente da Spring.

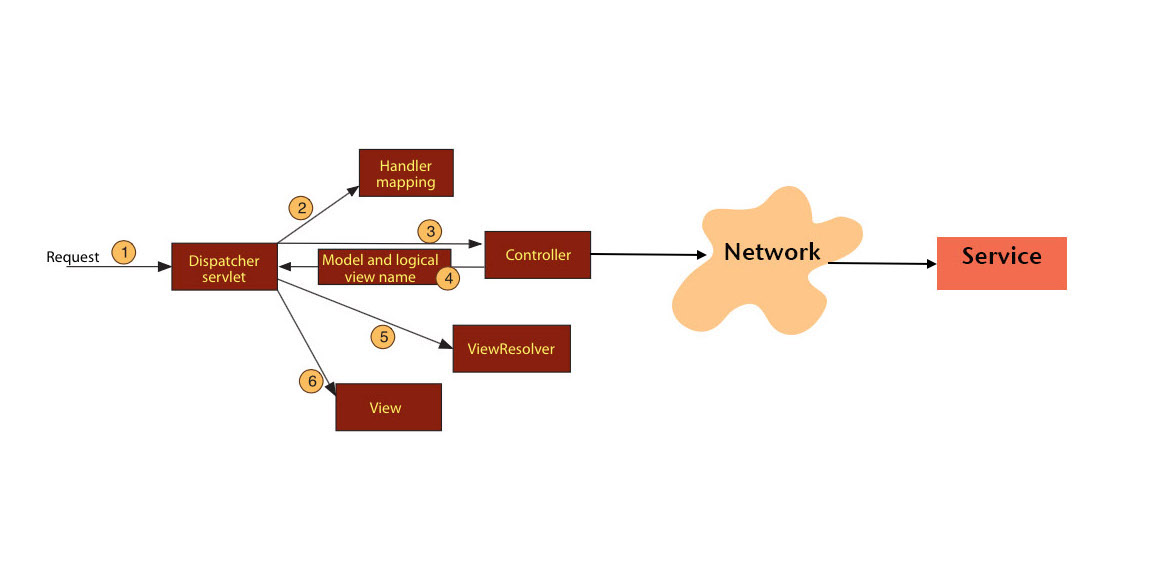


Dalla figura si evince che il Client, come detto, effettia la richiesta al Proxy creato dall’RMIProxyFactoryBean. Il Proxy converte la chiamata in una invocazione remota che rispetta il protocollo JRMP. La chiamata viene intercettata dall’RMI Service Adapter, creato dall’RMIServiceExporter, che la inoltra finalmente al servizio.

In Spring, se si sta sviluppando codice che utilizza servizi remoti, l’implementazione o l’uso di tali servizi è puramente una questione di configurazione. In Spring non sarà pertanto necessario scrivere alcuna riga di codice per supportare la comunicazione remota.

**Car2go-Rmi**

Per quanto riguarda il lavoro di tesi, è possibile identificare due macro-componenti: il client ed il server. Il ruolo del client è svolto dall’applicazione web. La richiesta inviata da un utente verrà gestita da Spring MVC ed in particolare giungerà ad un controller opportuno. All’interno del controller ci sarà l’invocazione del servizio remoto.



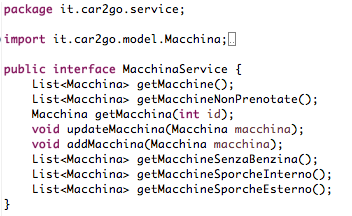
Per quanto riguarda il server sono state create le seguenti interfacce remote:

* MacchinaService;
* PrenotazioneService;
* RuoloService;
* UtenteService.

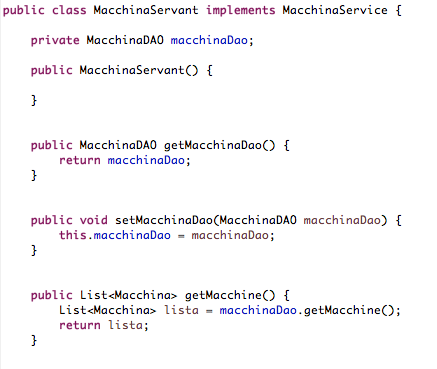
Tali interfacce sono implementate da elementi, che ricoprono il ruolo di serventi, cosi chiamati:

* MacchinaServant;
* PrenotazioneServant;
* RuoloServant;
* UtenteServant.

Il frammento di codice seguente riporta l’interfaccia MacchinaService



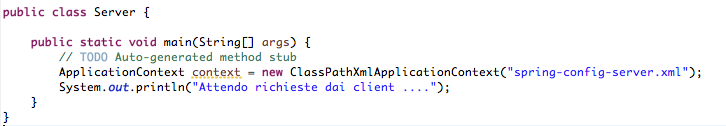
Tale interfaccia sarà implementata dall’elemento MacchinaService. Tale elemento, per completare gli scopi per i quali è stato creato, userà dei DAO per l’interazione con il database.



Analizzata la conformazione del progetto è necessario capire, a questo punto, come avviene l’esposizione remota di tali servizi mediante Spring.

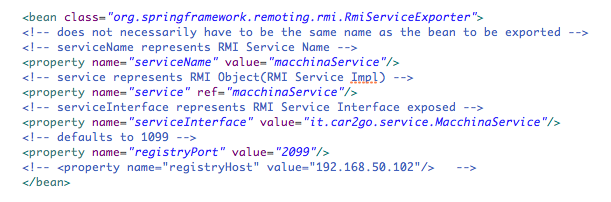
Come visto, è compito dell’elemento Server istanziare l’oggetto servente, esportarlo come oggetto remoto per poi registrarlo presso l’object registry.

In Spring il Server, per raggiungere tali scopi, deve soltanto riedere il caricamento dell’ApplicationContext di Spring.

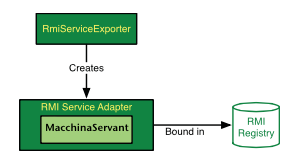


La definizione dell’ApplicationContext è contenuta all’interno del file spring-config-server.xml.

In tale file ci saranno specificate le regole per permettere l’iniezione delle dipende tra i serventi ed i DAO nonché le definizioni del servizio remoto.



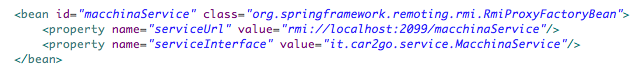
Le linee di codice appena mostrate sono quelle necessarie per esportare MacchinaService come servizio remoto.



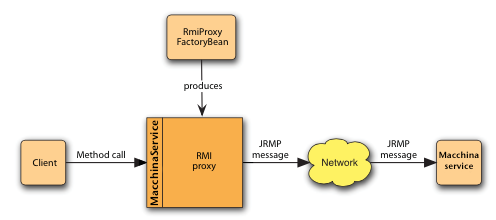
RmiServiceExpoter (classe specificata nella prima riga di configurazione) esporta qualsiasi bean gestito da Spring come un servizio RMI. Come illustrato nella figura soprastante, RmiServiceExpoter per poter svolgere la sua funzione incapsula il bean in una classe che svuole il ruolo di adattatore. La classe adattatore viene quindi registrata nell’RMI Registry e sempre tale classe rivolgerà le richieste che le arrivano al vero Servente (in questo caso MacchinaServant). Per quanto riguarda il registry RMI, la sua attivazione sarà svolta in automatico dall’RmiServiceExpoter.

Le righe successive della configurazione servono per indicare qual’è la classe che implementa il Servente, qual’è il nome del servizio remoto e la sua interfaccia. Può anche essere specificata una porta sulla quale esporre il servizio. Sarà compito di RmiServiceExpotert effettuare il mapping tra la porta indicata nel file di configurazione e quella effettiva dell’RMI Registry.

La realizzazione del **Client** è molto semplice.



Le linee di codice di configurazioni appena mostrate sono sufficienti per effettuare il collegamento con un oggetto remoto.



RmiProxyFactoryBean è una factory il cui scopo è quello di creare un proxy ad un servizio RMI. Per la creazione di tale proxy è necessario specificare l’interfaccia ed indirizzo del servizio. Il client può quindi inoltrare le richieste al proxy attraverso l’interfaccia del servizio senza sapere che in realtà il verso servente non è a lui locale.