AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Java Remote Method Invocation

http://galaxy.agh.edu.pl/~fmal/rmi

Filip Malawski

fmal@galaxy.agh.edu.pl

RMI

- Realizacja idei RPC (*Remote Procedure Call*)
- Mechanizm RMI został włączony do JVM począwszy od wersji JDK 1.1
- Główną zaletą RMI jest umożliwienie tworzenia programów rozproszonych w bardzo podobny sposób, jak w przypadku „pojedynczych” aplikacji.

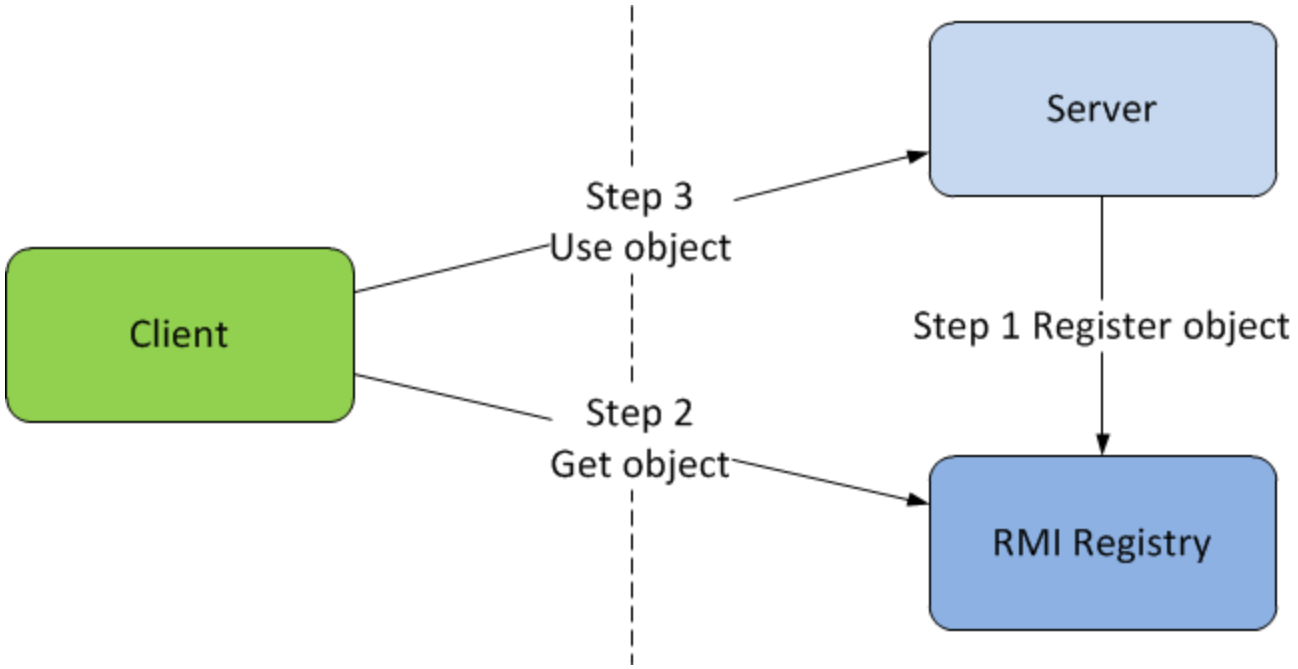
Architektura

- Aplikacje RMI mają strukturę klient-serwer. RMI dostarcza mechanizmów komunikacji pomiędzy klientem i serwerem.
- Wyróżniamy podstawowe elementy:
  - Klient
  - Serwer
  - Obiekty zdalne
  - Rejestr

Architektura

- Typowy serwer:
  - tworzy jeden lub więcej obiektów zdalnych
  - udostępnia referencje do nich (przez rejestr)
  - czeka na wywołania metod przez klienta
- Typowy klient:
  - uzyskuje referencje do obiektów zdalnych (przez rejestr)
  - wywołuje na nich metody

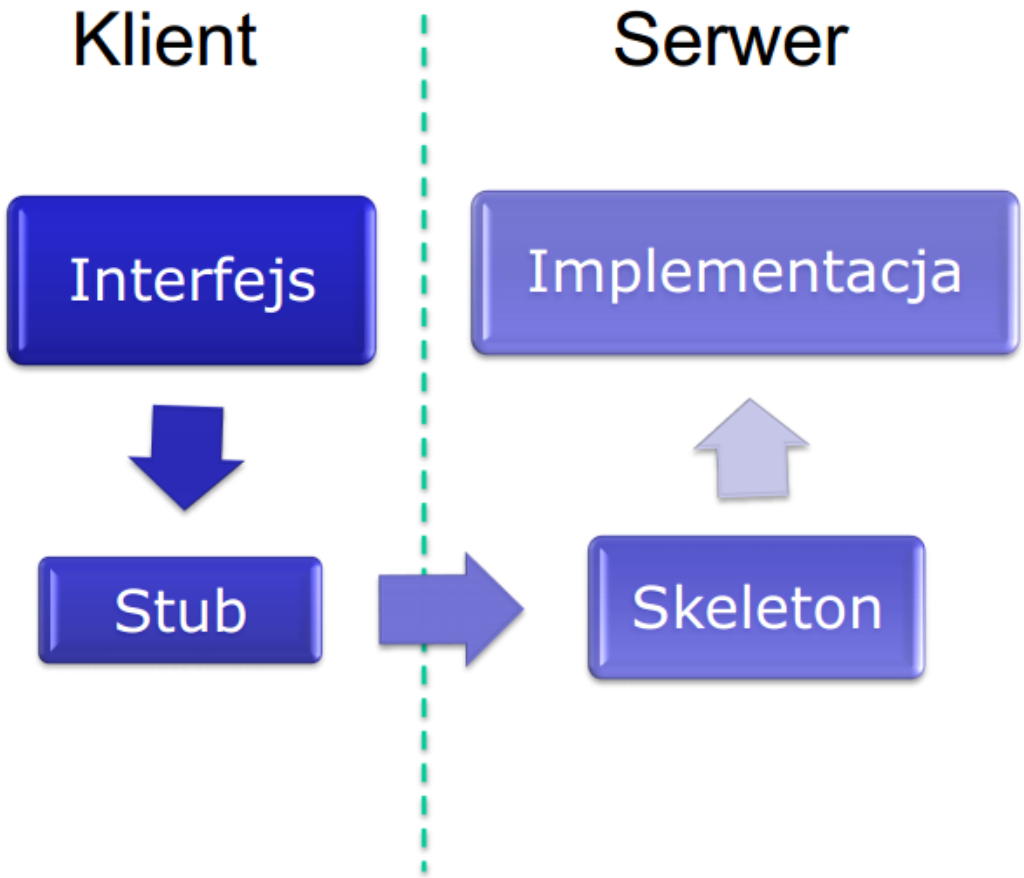
Architektura



Wywołanie operacji

- RMI nie tworzy kopii obiektu zdalnego (serwera) w JVM klienta
- Zamiast tego tworzony jest “stub” – rodzaj proxy, który jest lokalną reprezentacją obiektu zdalnego
- Stub implementuje interfejs obiektu zdalnego (serwera)
- Szczegóły komunikacji ukryte są przez RMI

Wywołanie operacji



Przesyłanie obiektów

- Przez referencję

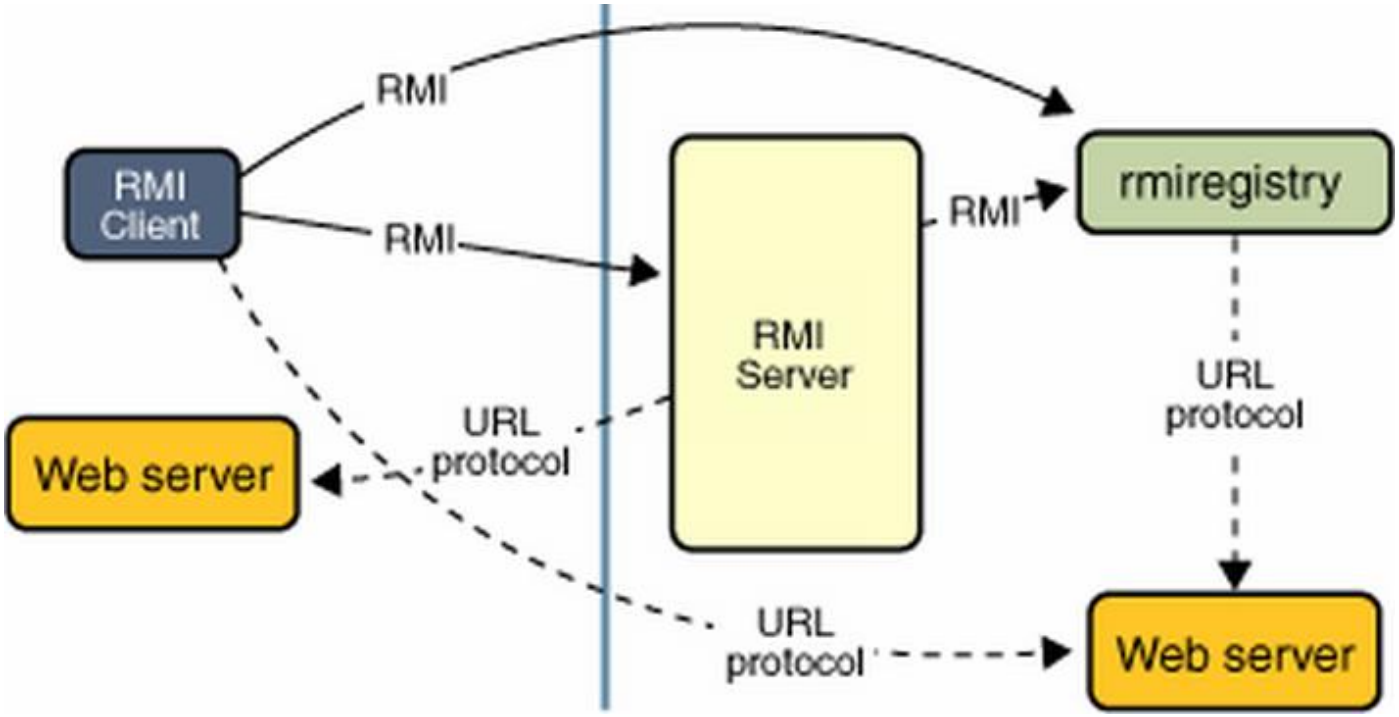
  obiekty zdalne – dziedziczą z Remote
- Przez wartość

  obiekty serializowane – implementują interfejs Serializable

Dynamiczne ładowanie klas

- Ładowanie klas dla obiektów przekazanych w wywołaniu zdalnym (Dynamic Class Loading)
- RMI umożliwia ściągnięcie definicji klasy, jeżeli nie jest ona dostępna w JVM klienta
- Dzięki temu zachowanie obiektu nie ulega zmianie podczas wysyłania go do JVM klienta
- DCL umożliwia dynamiczne rozszerzanie zachowania istniejącej aplikacji

Dynamiczne ładowanie klas



Jak napisać aplikację

- Interfejs zdalny
  - dziedziczy z java.rmi.Remote
  - każda metoda rzuca java.rmi.RemoteException
- Implementacja interfejsu
- Serwer
  - tworzy obiekt z implementacją interfejsu
  - eksport obiektu przez UnicastRemoteObject
  - rejestracja obiektu w RMI registry
- Klient
  - wyszukanie w RMI registry obiektu zdalnego
  - wywołanie odpowiedniej funkcjonalności

Jak uruchomić aplikację

- Kompilacja programów serwera i obiektu zdalnego
- Utworzenie stubów i skeletona (komenda rmic): od java 1.5 niepotrzebne
- Uruchomienie RMI registry
  - rmiregistry nr_portu (domyślnie 1099)
- Uruchomienie serwera
- Kompilacja klienta
- Uruchomienie klienta

Zadanie 1

- „Wspólna tablica”
- Serwer przechowuje tablicę
- Klienci mogą do niej dopisywać
- http://galaxy.agh.edu.pl/~fmal/rmi/rmi.zip
- Struktura:
  - INoteBoard - interfejs
  - NoteBoardImpl - implementacja
  - NoteBoardServer - serwer
  - NoteBoardClient - klient

Zadanie 1 - interfejs

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface INoteBoard extends Remote {
    public String getText() throws RemoteException;

    public void appendText(String newNote) throws RemoteException;
}
```

Zadanie 1 - implementacja interfejsu

```
import java.rmi.RemoteException;

public class NoteBoardImpl implements INoteBoard {
    private StringBuffer buf;

    public NoteBoardImpl() {
        buf = new StringBuffer();
    }

    public String getText() throws RemoteException {
        return buf.toString();
    }

    public void appendText(String newNote) throws RemoteException {
        buf.append("\n" + newNote);
    }
}
```

Naming

- Klasa java.rmi.Naming służy do przypisywania zdalnie dostępnym obiektom nazw. Każda z jej metod jako jeden z parametrów przyjmuje nazwę w formacie: „rmi://host:port/name”
- Domyślnie: host 127.0.0.1, port 1099

Zadanie 1 – Serwer

```
import java.rmi.Naming;
import java.rmi.server.UnicastRemoteObject;

public class NoteBoardServer {
    static INoteBoard nbi;

    public static void main(String[] args) {
        try {
            nbi = new NoteBoardImpl();
            UnicastRemoteObject.exportObject(nbi, 0);
            System.out.println("Export OK");
            Naming.rebind("rmi://127.0.0.1:1099/note", nbi);
            System.out.println("Rebind OK");
            System.out.println("Server active");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Zadanie 1 – Klient

```
import java.rmi.Naming;

public class NoteBoardClient {

    public static void main(String[] args) {
        try {
            INoteBoard nb = (INoteBoard)
            Naming.lookup("rmi://127.0.0.1:1099/note");
            System.out.println("Lookup OK");
            System.out.println("Dodajemy tekst");
            nb.appendText("ala ");
            nb.appendText("ma ");
            nb.appendText("kota ");
            System.out.println("Pobieramy tekst: ");
            System.out.println(nb.getText());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Zadanie 1 - uruchomienie

- Ścieżka do kompilatora
  - np. *set Path=%Path%;C:\Program Files\Java\jdk1.5.0_22\bin*
- Kompilacja
  - np. *javac *.java* w katalogu rmi
- rmiregistry
  - np. *rmiregistry*
  - uwaga: **nie** w katalogu z klasami
  - brak outputu
- uruchomienie serwera
  - classpath: interfejs, implementacja, kod serwera
  - codebase: interfejs
  - np. :

  java -cp rmi -Djava.rmi.server.codebase=file:/c:/rmi/INoteBoard.jar NoteBoardServer
  java -cp rmi -Djava.rmi.server.codebase=file:/c:/rmi**/** NoteBoardServer
  java -cp c:\rmi -Djava.rmi.server.codebase=file:/c:\rmi**** NoteBoardServer
  java -cp c:\rmi -Djava.rmi.server.codebase=file:/c:\Doc%20And@20Sett\rmi**** NoteBoardServer
- uruchomienie klienta
  - classpath: interfejs, kod klienta
  - np. :
  - java -cp rmi NoteBoardClient
  - java NoteBoardClient

Zadanie 1 – zdalne ładowanie klas

- Codebase definiuje lokację z której mogą być pobrane klasy (interfejsy) używane przez obiekty zdalne

- Zadanie
  - Zresetować rmiregistry
  - Zmienić codebase przy starcie serwera na:

-Djava.rmi.server.codebase=http://galaxy.agh.edu.pl/~fmal/classes/

Zadanie 2

- Klient przy rejestracji podaje obiekt User
  - User **nie** jest obiektem zdalnie dostępnym (serializacja)
- Serwer sprawdza jedynie, czy pseudonim użytkownika nie jest wykorzystywany (jeśli tak klient kończy działanie)
- Wywołanie metody appendText() powoduje wywołanie na wszystkich zarejestrowanych klientach metody onNewText()
  - Klient musi zarejestrować swój obiekt zdalny

Zadanie 2

- INoteBoard

  public boolean register(User user, INoteBoardListener nbl) throws RemoteException;

- INoteBoardListener

  public void onNewText(String text) throws RemoteException;

- Klient
  - Nazwa usera z linii poleceń
  - Pętla do wpisywania tekstu
  - Konieczne podanie codebase przy uruchomieniu

Zadanie domowe

- Implementacja gry kółko i krzyżyk
- Serwer
  - Umożliwia dołączenie się do gry
  - Gra z komputerem lub z drugim klientem
  - Wielowątkowość
- Klient
  - GUI
  - *Identyfikuje przez nick oraz avatar*
  - *Podczas gry wyświetla nick i avatar swój oraz przeciwnika*
- Współpraca klienta i serwera na różnych hostach

Zadanie domowe

- Spakowany katalog RMI_2012_Nazwisko_Imie
- Mail z tematem tj. nazwa katalogu na adres **fmal@galaxy.agh.edu.pl**

- Tutorial do RMI:

  http://docs.oracle.com/javase/tutorial/rmi/

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Dziękuję