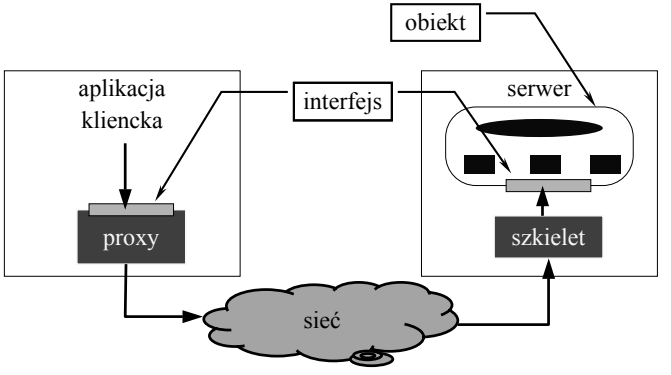
Wywoływanie metod zdalnych

Podejście obiektowe do budowy systemów rozproszonych

Wywoływanie metod zdalnych — model systemu



Istota podejścia obiektowego

- Obiekt jest jednostką integrującą w sobie dane (stan obiektu) oraz odpowiednio zdefiniowane operacje (metody)
- Jedyna forma dostępu do danych polega na wywoływaniu metod wyspecyfikowanych w publicznym interfejsie obiektu
- Obiekt może implementować wiele interfejsów
- Ten sam interfejs może być implementowany przez wiele obiektów

Podejście obiektowe do budowy systemów rozproszonych

- Kluczowe dla mechanizmu wywoływania metod zdalnych jest oddzielenie definicji interfejsu (specyfikacji) od jego implementacji w obiekcie
- W językach definicji interfejsu (IDL) interfejs traktowany jest jak typ danych
- W języku implementacji każdy obiekt implementujący dany interfejs jest instancją tego typu

Specyfikacja interfejsu

- Opis interfejsu w języku implementacji (np. Ada, Java RMI)
- Opis interfejsu w języku specjalnym, niezależnym od implementacji (CORBA — IDL, Sun RPC — rpcgen)

Przykład opisu interfejsu w podejściu obiektowym (CORBA IDL)

```
interface Konto {
     float stan();
     float wplac(in float value);
     float pobierz(in float value);
};

interface Bank {
     Konto otworzKonto(in int numer);
     float zamknijKonto(in Konto k);
};
```

Klasyfikacja obiektów ze względu na sposób implementacji

- Obiekt zdalny (ang. remote object) — stan obiektu utrzymywany jest przez jeden serwer (wszystkie dane obiektu znajdują się na jednym serwerze)
- Obiekt rozproszony (ang. distributed object) — stan obiektu przechowywany jest przez więcej niż jeden serwer (poszczególne serwery przechowują odpowiednie dane, stanowiące fragmenty tego samego obiektu)
- Z punktu widzenie klienta zarówno obiekt zdalny jak i obiekt rozproszony stanowi pewną całość odpowiednio identyfikowaną

Klasyfikacja ze względu na dostępność informacji o obiektach

- Obiekt dostępny w czasie kompilacji (ang. compile-time object) — informacja o obiekcie (jego typ) znana jest i dostępna w odpowiedniej formie w programie klienta na etapie tworzenia oprogramowania
- Obiekt dostępny w czasie wykonania (ang. runtime object) — informacja o obiekcie dostępna jest dopiero w czasie działania aplikacji

Klasyfikacja obiektu ze względu na trwałość

- Obiekt trwały (ang. persistent object) — obiekt istnieje nawet wówczas, gdy nie ma serwera, w przestrzeni adresowej którego mógłby być przechowywany stan tego obiektu
- Obiekt przejściowy (ang. transient object) — obiekt istnieje tak długo, jak długo działa serwer utrzymujący jego stan

Referencja do obiektu

- Referencja jest identyfikatorem, wykorzystywanym do wskazania obiektu, na którym ma zostać wykonana operacja
- W celu wywołania metody obiektu rezydującego na innej maszynie potrzebna jest zdalna referencja
- Referencja może być przekazywana jako parametr wywołania metody lub zwrócona jako wynik wykonania metody

Implementacja zdalnej referencji

- Referencja jest wartością, której struktura wewnętrzna nie jest w żaden sposób interpretowana przez aplikację
- Zdalna referencja musi zawierać wystarczająco dużo informacji, żeby dowiązać obiekt w programie klienta
- Dowiązanie obiektu oznacza uzyskanie informacji, niezbędnej do wywołania metody obiektu (uzyskanie proxy)

Dowiązanie obiektu

- Dowiązanie jawne (ang. explicit binding) — dowiązanie wymaga wywołania odpowiedniej funkcji (np. bind), po którym może dopiero nastąpić wywołanie metody
- Dowiązanie domyślne (ang. implicit binding) — wywołanie metody za pośrednictwem referencji skutkuje ustanowieniem dowiązanie do obiektu w procesie klienta

Wywoływanie metod

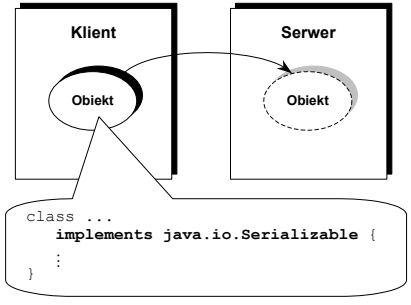
- Statyczne wywoływanie metod — wywołanie metody za pośrednictwem proxy, wygenerowanego na podstawie definicji interfejsu
- Dynamiczne wywoływanie metod — „skomponowanie" informacji niezbędnych do wywołania w czasie działania systemu, np.:

```
invoke(ref_obj, id_metody,
       param_wej, param_wyj);
```

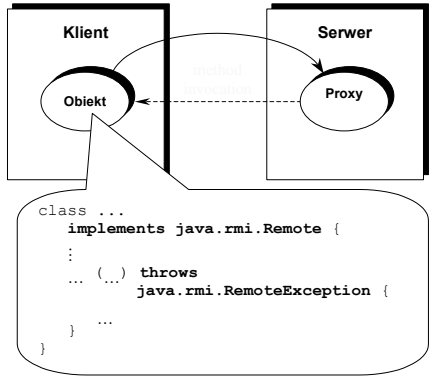
Przekazywanie obiektów jako parametrów

- Przekazywanie przez wartość — do zdalnej metody przekazywana jest kopia obiektu, który jest parametrem rzeczywistym wywołania.
- Przekazywanie przez referencję (zmienną) — do zdalnej metody przekazywana jest referencja (zdalna) do obiektu, który jest parametrem rzeczywistym.
- Przekazywanie przez kopiowanie i odtwarzanie — do zdalnej metody przekazywana jest kopia obiektu, a po zakończeniu zdalnej metody następuje aktualizacja obiektu po stronie wywołania, stosownie do zmian dokonanych w ramach wykonania metody.

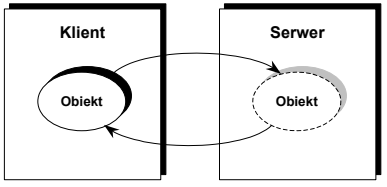
Przekazywanie przez wartość — przykład w Java RMI



Przekazywanie przez referencję — przykład w Java RMI



Przekazywanie przez kopiowanie i odtwarzanie



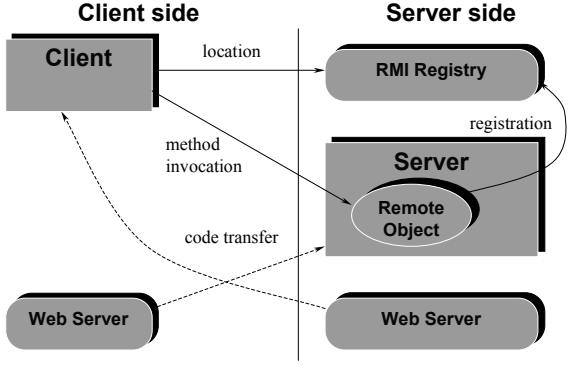
Tworzenia aplikacji rozproszonej w środowisku Java RMI (1)

1. Zdefiniowanie i implementacja odpowiednich klas (w szczególności klas dla obiektów dostępnych zdalnie)
   - zdefiniowanie interfejsu pochodnego od `Remote`
   - zdefiniowanie klasy wywiedzionej z klasy `java.rmi.server.UnicastRemoteObject`, implementującej interfejs pochodny od `Remote`
2. Kompilacja źródeł (`javac`, `rmic`)
   - `javac xxx.java` ⇨ `xxx.class`
   - `rmic xxx` ⇨ `xxx_Stub.class` `xxx_Skel.class`

Tworzenia aplikacji rozproszonej w środowisku Java RMI (1)

3. Udostępnienie wygenerowanego kodu klas
   - wspólny system plików
   - kopia kodów klas w różnych systemach plików
   - udostępnianie kodu przez serwer www
4. Uruchomienie aplikacji
   - uruchomienie `rmiregistry` (name server)
   - uruchomienie serwera: utworzenie zdalnych obiektów i ich rejestracja w `rmiregistry`
   - uruchomienie klienta: zlokalizowanie zdalnych obiektów (odwołanie do `rmiregistry`) i wywoływanie zdalnych metod

Komunikacja między klientem a serwerem



Interfejs `java.rmi.Remote`

Komunikacja pomiędzy serwerem (obiektem) a klientem jest określona przez definicję interfejsu pochodnego od interfejsu `Remote`. Klasa zdalnego obiektu musi implementować ten interfejs

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

public interface Compute extends Remote {
   Object executeTask(Task t) throws RemoteException;
}
```

Zdalne udostępnianie obiektu (serwer)

Definicja klasy obiektu:

```
import java.rmi.*;
import java.rmi.server.*;
import compute.*;

public class ComputeEngine extends UnicastRemoteObject
                           implements Compute {
      public ComputeEngine() throws RemoteException {
         super();
      }
      public Object executeTask(Task t) {
         return t.execute();
      }
```

Utworzenie i rejestracja zdalnego obiektu

```
public static void main(String[] args) {
 if (System.getSecurityManager() == null) {
   System.setSecurityManager(new RMISecurityManager());
 }
 String name = "//host:1099/Compute";
 try {
    Compute engine = new ComputeEngine();
    Naming.rebind(name, engine);
    System.out.println("ComputeEngine ok");
  } catch (Exception e) {
  System.err.println("ComputeEngine excep."
                       + e.getMessage());
      e.printStackTrace();
   } }}
```

Zdalne wywoływanie metod

```
import java.rmi.*;
public class launchComp {
  public static void main(String args[]) {
    if (System.getSecurityManager() == null){
       System.setSecurityManager(
                 new RMISecurityManager());
    }
    try {
      String name = "//"+args[0]+"/Compute";
      Compute comp = (Compute) Naming.lookup(name);
      CompArea task = new CompArea(...);
      BigDecimal arae =
              (BigDecimal) (comp.executeTask(task));
         System.out.println(area);
      } catch (Exception e) {
         ...
      } } }
```

Przekazywanie parametrów

- Jeżeli przekazujemy obiekt przez wartość, to klasa tego obiektu musi implementować interfejs `Serializable`.
- Jeśli przekazujemy obiekt przez referencja, to klasa tego obiektu musi implementować interfejs `Remote`.

```
public interface Task extends
                        java.io.Serializable {
     Object execute();
   }
```

```
public class CompArea implements Task {
   ⋮
}
```

Lokalizowanie zdalnych obiektów

- Rejestr — wiąże z nazwami i udostępnia zdalne obiekty
- Tworzenie rejestru — rmiregistry [<port number>]
- Klasa LocateRegistry — umożliwia tworzenie rejestru (obiekty klasy Registry)
- Klasa Registry — klasa obiektu-rejestru
- Klasa Naming — ułatwia korzystanie z rejestru, umożliwiając jego specyfikację w adresie URL

Klasa `LocateRegistry`

```
static
Registry createRegistry(int port)
                 throws RemoteException
static
Registry createRegistry(String host, int port)
                 throws RemoteException
static
Registry getRegistry(int port)
                 throws RemoteException
static
Registry getRegistry(String host, int port)
                 throws RemoteException
```

Klasa `Registry`

```
void bind(String name, Remote obj)
       throws RemoteException,
         AlreadyBoundException, AccessException
void rebind(String name, Remote obj)
       throws RemoteException, AccessException
void unbind(String name)
       throws RemoteException,
         NotBoundException, AccessException
String[] list()
       throws RemoteException, AccessException
Remote lookup(String name)
       throws RemoteException,
         NotBoundException, AccessException
```

"Compute"

Klasa `Naming` (1)

```
static
void bind(String urlName, Remote obj)
  throws RemoteException,
    AlreadyBoundException, AccessException,
    MalformedURLException, UnknownHostException

static
void rebind(String urlName, Remote obj)
  throws RemoteException, AccessException,
   MalformedURLException, UnknownHostException
```

Klasa `Naming` (2)

```
static
void unbind(String urlName)
  throws RemoteException, NotBoundException,
        AccessException, MalformedURLException,
        UnknownHostException
static
String[] list()
  throws RemoteException, AccessException
    MalformedURLException, UnknownHostException
static
Remote lookup(String urlName)
  throws RemoteException, NotBoundException,
        AccessException, MalformedURLException
```

"//host:1099/Compute"