RMI & JAVA RMI

Μοντέλο κατανεμημένου αντικειμένου

- Το μοντέλο κατανεμημένου αντικειμένου είναι μια επέκταση του μοντέλου αντικειμένου

- Αντικείμενα που μπορούν να δεχθούν απομακρυσμένες αιτήσεις για την παροχή υπηρεσιών, ονομάζονται *απομακρυσμένα αντικείμενα*

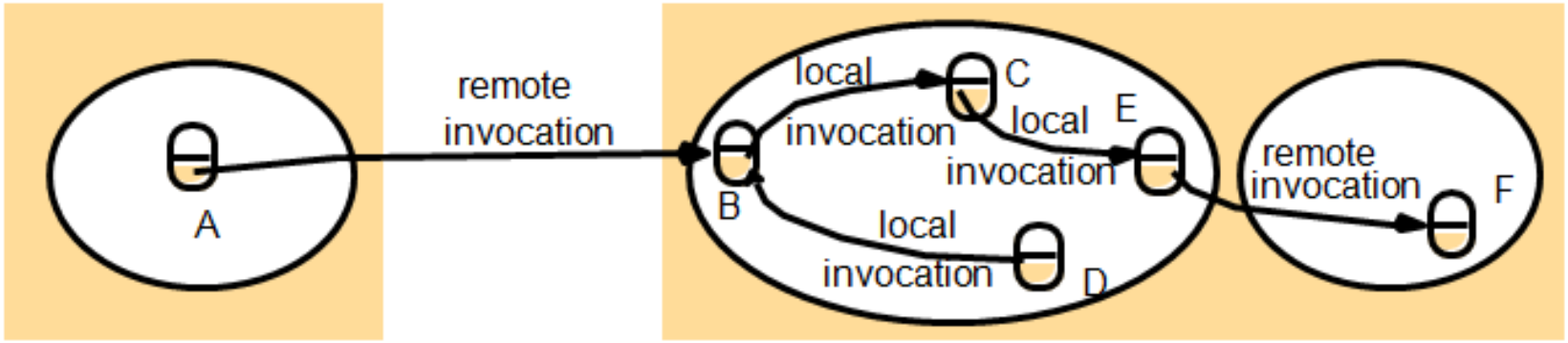
Μοντέλο RMI

- Τα απομακρυσμένα αντικείμενα θα πρέπει να διαθέτουν έναν τρόπο προσπέλασής τους δια μέσου μιας αναφοράς, που ονομάζεται *αναφορά απομακρυσμένου αντικειμένου*

- Για την κλήση μιας μεθόδου, η υπογραφή της μεθόδου και οι παράμετροί της θα πρέπει να έχουν οριστεί σε μια *απομακρυσμένη διασύνδεση (remote interface)*

- Οι προηγούμενες έννοιες συνθέτουν το μοντέλο της *κλήσης απομακρυσμένης μεθόδου* (*remote method invocation* ή *RMI*)

Μοντέλο RMI

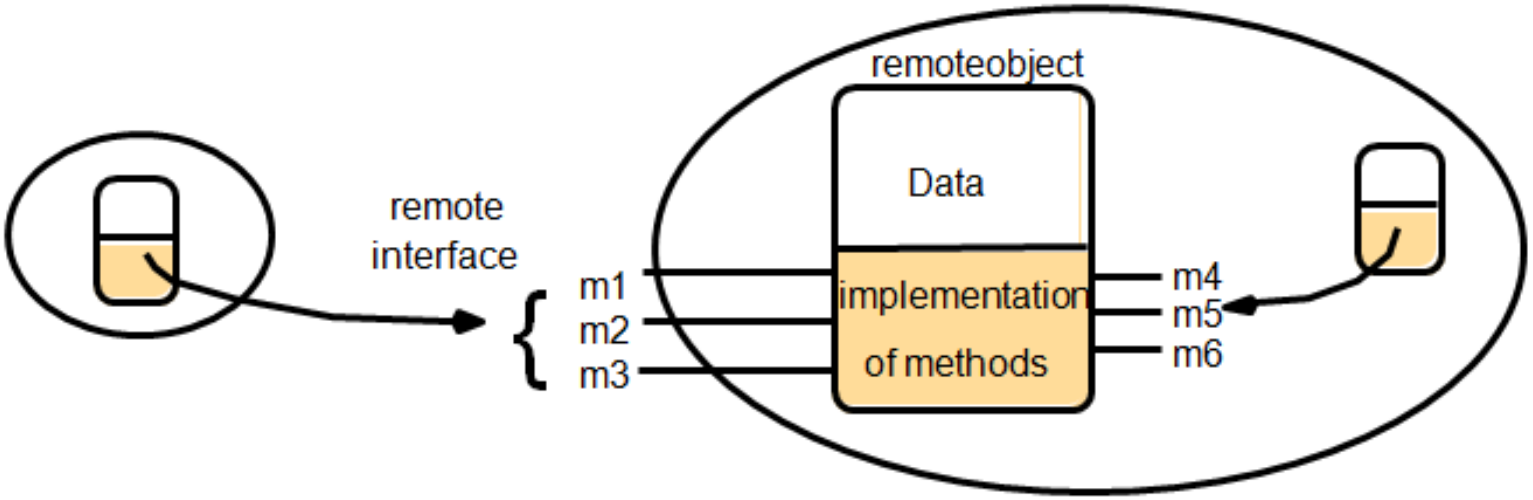


- Κάθε διεργασία (οβάλ σχήμα) περιλαμβάνει ένα σύνολο από αντικείμενα (A, B,…, F), μερικά από τα οποία μπορούν να χειρισθούν τόσο τοπικές όσο και απομακρυσμένες κλήσεις μεθόδων
  - κάποια άλλα αντικείμενα μπορούν να καταλάβουν μόνο τοπικές κλήσεις των μεθόδων τους.

- Η κλήση μεθόδων μεταξύ αντικειμένων διαφορετικών διεργασιών, είτε οι διεργασίες είναι στον ίδιο υπολογιστή (σχήμα πλαισίου), είτε όχι, είναι κλήσεις απομακρυσμένων μεθόδων.

- Οι κλήσεις μεθόδων μεταξύ αντικειμένων της ίδιας διεργασίας είναι τοπικές κλήσεις.

Μοντέλο RMI



- Αντικείμενα σε άλλες διεργασίες μπορούν να καλέσουν τις μεθόδους που περιέχει η απομακρυσμένη διασύνδεση (m1, m2, m3).

- Τοπικά αντικείμενα μπορούν να καλέσουν τόσο τις μεθόδους της απομακρυσμένης διασύνδεσης όσο και άλλες μεθόδους που υλοποιεί ένα απομακρυσμένο αντικείμενο.

Αξιοπιστία στο μοντέλο RMI

- Το μοντέλο απομακρυσμένου αντικειμένου χρειάζεται έναν τρόπο για να ανιχνεύει και να χειρίζεται αστοχίες
- Αστοχίες μπορούν να συμβούν στο καλών αντικείμενο (πελάτης), στο καλούμενο αντικείμενο (εξυπηρετητής) ή στο δίκτυο, και έτσι η διαχείρισή τους είναι πιο πολύπλοκη από την αντίστοιχη του μοντέλου αντικειμένου με τοπικές κλήσεις μεθόδων
- Διάφορα μέτρα προστασίας μπορούν να εφαρμοστούν για την παροχή αξιοπιστίας πάνω από τα πρωτόκολλα αιτήσεων-αποκρίσεων, με κυριότερες επιλογές τις εξής:

Αξιοπιστία στο μοντέλο RMI

- Επανάληψη Μηνύματος Αίτησης (Retry Request Message): προσδιορίζει εάν ο πελάτης θα πρέπει να επαναμεταδίδει την αίτηση μέχρι, είτε να ληφθεί μια απόκριση, είτε να θεωρηθεί ότι ο εξυπηρετητής έχει αποτύχει
- Φιλτράρισμα Διπλότυπων (Duplicate Filtering): προσδιορίζει εάν η λήψη του ίδιου αιτήματος πολλές φορές από τον εξυπηρετητή, θα πρέπει να φιλτραριστεί σε μια μόνο αίτηση, ή εάν όλες οι αιτήσεις θα πρέπει να εκτελούνται ανεξάρτητα κάθε φορά
- Επαναμετάδοση Αποτελεσμάτων (Retransmission of Results): προσδιορίζει εάν τα αποτελέσματα από την εκτέλεση των αιτημάτων θα πρέπει να αποθηκεύονται σε μια δομή history και να επαναμεταδίδονται, σε περίπτωση απώλειας, αντί να εκτελείται εκ νέου στον εξυπηρετητή η λειτουργία του αιτήματος

Invocation Semantics

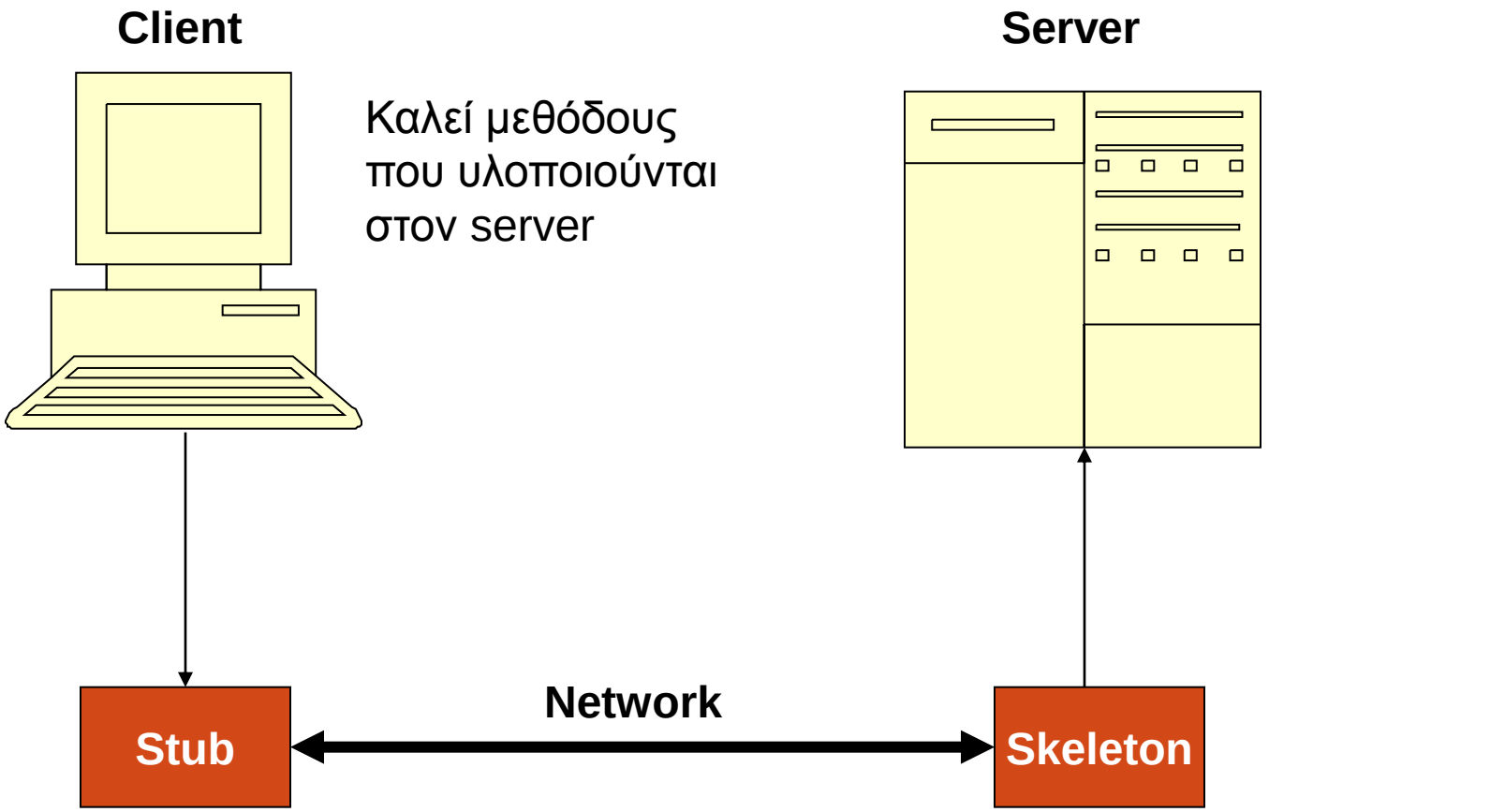
- Με βάση του ποιοι από τους προηγούμενους μηχανισμούς εφαρμόζονται προκύπτουν τρεις τύποι κλήσης απομακρυσμένης μεθόδου.
  - ***Maybe invocation semantics***: Το αίτημα για απομακρυσμένη εκτέλεση μεθόδου, είτε θα διεκπεραιωθεί μια φορά, είτε καθόλου.
    - Εφαρμόζεται όταν δε γίνεται επαναμετάδοση των αιτήσεων μετά από κάποια αστοχία.
    - Ο πελάτης δεν μπορεί να διακρίνει εάν ο εξυπηρετητής διεκπεραίωσε το αίτημα ή όχι, μετά την εμφάνιση της αστοχίας.
  - ***At-Least-Once Invocation Semantics***: Ο πελάτης λαμβάνει, είτε ένα αποτέλεσμα, είτε μια εξαίρεση.
    - Ο εξυπηρετητής θα εκτελέσει την λειτουργία του αιτήματος πολλές φορές, μια για κάθε επανάληψη του μηνύματος αίτησης που λαμβάνει.
    - Δεν είναι κατάλληλη επιλογή όταν η εκτέλεση της ίδιας μεθόδου πολλές φορές αλλάζει το επιθυμητό αποτέλεσμα.
  - ***At-Most-Once Invocation Semantics***: Ο πελάτης λαμβάνει, είτε ένα αποτέλεσμα, είτε μια εξαίρεση.
    - Εάν ο πελάτης λάβει το αποτέλεσμα, σημαίνει ότι ο εξυπηρετητής διεκπεραίωσε την λειτουργία του αιτήματος ακριβώς μια φορά.

Remote Method Invocation (RMI)

- Το RMI σχεδιάστηκε ώστε η αλληλεπίδραση μεταξύ δυο προγραμμάτων που ακολουθούν το αντικειμενοστρεφές μοντέλο και τρέχουν σε διαφορετικές μηχανές να μοιάζει όσον το δυνατόν περισσότερο με την κλήση μεθόδων ενός προγράμματος που εκτελείται τοπικά σε μια μηχανή

- Βασίζεται κυρίως σε δύο αντικείμενα που παράγονται αυτόματα από τον compiler
  - *Stub ή Proxy*
  - *Skeleton*

Stubs και Skeletons



Πέρασμα παραμέτρων (by value vs. by reference)

- Το RMI διαχωρίζει τα αντικείμενα σε:
  - Απομακρυσμένα αντικείμενα (remote objects)
    - λαμβάνουν απομακρυσμένες κλήσεις των μεθόδων τους
    - έχουν συγκεκριμένη θέση
    - ως ορίσματα περνούν by reference
  - Σειριοποιήσιμα αντικείμενα (serializable objects)
    - η θέση τους δε σχετίζεται με την κατάστασή τους
    - κελυφοποιούν δεδομένα
    - μετακινούνται μεταξύ JVMs
    - ως ορίσματα περνούν by value
- Πρωτογενείς τύποι δεδομένων περνούν by value
- Αντικείμενα τα οποία δεν είναι ούτε Remote ούτε Serializable προκαλούν την έγερση εξαιρέσεων

Υλοποίηση Java RMI εφαρμογών – Βήμα 1

- Αρχικά πρέπει να γράψουμε την απομακρυσμένη διασύνδεση και την υλοποίησή της, δηλαδή την υλοποίηση του απομακρυσμένου αντικειμένου

- Η απομακρυσμένη διασύνδεση πρέπει να επεκτείνει τη διασύνδεση java.rmi.Remote και όλες οι μέθοδοί της να εγείρουν την εξαίρεση java.rmi.RemoteException

- Η κλάση υλοποίησης συνήθως επεκτείνει την κλάση java.rmi.UnicastRemoteObject για ένα παροδικό αντικείμενο

Υλοποίηση Java RMI εφαρμογών – Βήμα 2

- Στη συνέχεια πρέπει να γράψουμε το πρόγραμμα του εξυπηρετητή και το πρόγραμμα του πελάτη
- Το πρόγραμμα του εξυπηρετητή θα δημιουργεί τα απομακρυσμένα αντικείμενα, θα εγγράφει τις αναφορές τους στο μητρώο του συστήματος εξυπηρετητή και θα περιμένει κλήσεις μεθόδων των απομακρυσμένων αντικειμένων από πελάτες
- Το πρόγραμμα του πελάτη θα αναζητά και θα αποκτά απομακρυσμένες αναφορές προς ένα ή περισσότερα απομακρυσμένα αντικείμενα για να μπορεί να καλεί τις απομακρυσμένες μεθόδους τους

Παράδειγμα διασύνδεσης σε Java RMI

```
import java.rmi.*;

public interface AddServerIntf extends Remote {
        double add(double d1, double d2) throws RemoteException;
}
```

Υλοποίηση του remote object

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

public class AddServerImpl extends UnicastRemoteObject implements
AddServerIntf {
        public AddServerImpl() throws RemoteException
        {
        }
        public double add(double d1, double d2) throws RemoteException
        {
                return d1 + d2;
        }
}
```

Java RMIregistry

- Παρέχει μια υπηρεσία ονόματος (naming service) για τον εντοπισμό των απομακρυσμένων αντικειμένων
- Ένα στιγμιότυπο της RMIregistry πρέπει να τρέχει σε κάθε μηχανή που φιλοξενεί remote αντικείμενα
- Οι μέθοδοί της ορίζονται στην κλάση java.rmi.Naming
- Τα απομακρυσμένα αντικείμενα περιγράφονται στο registry με URL-style ονόματα της μορφής:
  `//computerName:portNumber/human-readable-objectName`
  - **`computerName`**: το όνομα της μηχανής που τρέχει το RMIregistry
  - **`portNumber`**: η θύρα για την πρόσβαση στο RMIregistry
  - **`human-readable-objectName`**: το όνομα που αντιπροσωπεύει το απομακρυσμένο αντικείμενο

RMIregistry μέθοδοι (Naming class)

void rebind (String name, Remote obj)

This method is used by a server to register the identifier of a remote object by name,

void bind (String name, Remote obj)

This method can alternatively be used by a server to register a remote object by name, but if the name is already bound to a remote object reference an exception is thrown.

void unbind (String name, Remote obj)

This method removes a binding.

Remote lookup(String name)

This method is used by clients to look up a remote object by name,

String [] list()

This method returns an array of Strings containing the names bound in the registry.

Παράδειγμα Java RMI server

```
import java.net.*;
import java.rmi.*;

public class AddServer {
        public static void main(String args[]) {
        try {
                AddServerImpl addServerImpl = new AddServerImpl();
                Naming.rebind("AddServer", addServerImpl);
            }
        catch(Exception e) { System.out.println("Exception: " + e); }
                                                    }
                        }
```

Παράδειγμα Java RMI client

```
import java.rmi.*;
public class AddClient {
  public static void main(String args[]) {
    try {
      String addServerURL = "rmi://" + args[0] + "/AddServer";
      AddServerIntf addServerIntf =
                  (AddServerIntf)Naming.lookup(addServerURL);
      System.out.println("The first number is: " + args[1]);
      double d1 = Double.valueOf(args[1]).doubleValue();
      System.out.println("The second number is: " + args[2]);

      double d2 = Double.valueOf(args[2]).doubleValue();
      System.out.println("The sum is: " + addServerIntf.add(d1, d2));
    }
    catch(Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Υπόλοιπα βήματα

- Compile all aforementioned Java source files and keep all generated .class files in the same directory.
- Before you can use the client and the server, you must generate the necessary stub.
- You may also need to generate a skeleton, if a server runs on a machine that does not have Java 2, but has an earlier version of JVM.

1. Generate stubs and skeletons (deprecated!)
   - To generate stubs and skeletons, you use the RMI compiler:
     - rmic AddServerImpl
       - This command generates two new files: AddServerImpl_Skel.class (skeleton) and AddServerImpl_Stub.class (stub).

Υπόλοιπα βήματα

2. Start the RMI registry
   - Go to the directory on the server machine where you keep files mentioned above.
   - If your server is on another Windows machine: start rmiregistry
   - If your server is on a UNIX machine: rmiregistry &

3. Start the server
   - In the same directory:
     - java AddServer

4. Run the client
   - java AddClient xxx.xxx.xxx.xxx 567 999

Policy αρχεία

- Περιέχουν περιγραφές των πολιτικών ασφαλείας για να περιορίσουμε τις ενέργειες που εκτελούνται από μεταφερόμενο κώδικα
- Π.χ. μπορούμε να καθορίσουμε διαφορετικούς τύπους πρόσβασης σε κάθε είδος τοπικής πληροφορίας
  - Το είδος της πληροφορίας μπορεί να περιλαμβάνει πρόσβαση σε αρχεία, σε ιδιότητες του συστήματος, σε επικοινωνία μέσω sockets κ.λπ.
  - Ο τύπος πρόσβασης μπορεί να περιλαμβάνει, για παράδειγμα, δυνατότητα διαβάσματος, εγγραφής, εκτέλεσης (για αρχεία)
- Αν δεν έχουμε ορίσει το δικό μας policy αρχείο ο SecurityManager κοιτάει το default java.home/lib/security/java.policy
- Μπορούμε να χρησιμοποιήσουμε την κλάση java.net.SocketPermission για να ορίσουμε ένα νέο policy δυναμικά

Συστατικά εφαρμογών RMI vs. Sockets

- Επιχειρησιακή λογική
- Διεπαφή χρήστη στον πελάτη
- Marshalling/Unmarshalling
- Ξεκίνημα και διαμόρφωση της εφαρμογής
- Αποδοτικότητα (εξισορρόπηση φορτίου, νήματα)

Αυτόματη τακτοποίηση από το RMI