MobileRMI

Marco Avvenuti and Alessio Vecchio
Dip. di Ingegneria dell’Informazione
Università di Pisa
{m.avvenuti, a.vecchio}@ing.unipi.it

MobileRMI

- MobileRMI is a mobility layer embedded into the Java Remote Method Invocation system:
  - Integration of logical mobility with a well-kown middleware.

- MobileRMI:
  - Achieves explicit mobility of remote objects;
  - Maintains the capability of interacting with mobile objects through method invocation.
Java RMI

- Remote method invocation is the action of invoking a method on a remote object (server) from a client object living in a different JVM.
- Clients must hold a remote reference to the server.
- Key features:
  - Invocation of a remote method has the same syntax of local invocation;
  - The programmer is aware of semantic differences between local and remote method invocation.
- An effective paradigm for building distributed applications across all Java flavors (J2SE, J2ME, J2EE).

RMI and mobility

- RMI supports object mobility in an indirect way (by passing objects as arguments)
  - But explicit primitives for mobility do not exist.
- Such mobile objects have the limit that it is not possible to invoke remote methods on them (because they are not remote objects).
- RMI prevents a remote from being serialized and sent between JVMs as a parameter
  - For remote objects, passing is by reference.
- Achieving serialization of remote objects is not enough:
  - To preserve remote method invocation on mobile objects, references held by clients must follow the migrating server.
Location transparence vs location awareness

Remote method invocation provides location transparence: the remote reference abstraction hides the location of a remote object from client programs.

However, the design of mobile applications is location-aware: the location of components is under programmer's control.

In MobileRMI, creation and migration of a remote object are location-aware operations: the client program must indicate the target location explicitly, in the form of a parameter of operations.

As remote references are location-dependent names, preserving location transparence requires a mechanism for updating remote references to mobile objects.

Remote creation and remote clone

MobileRMI extends standard RMI with special methods

- To *create* a server in a remote address space;

- to *clone* a server in a remote address space;
Move

MobileRMI makes server object mobility possible by means of special methods
- to *move* a server from an address space to another.

- Provides class loading policies more suitable for object mobility.

![Diagram](image)

Programming with MobileRMI

```java
public class ComputeTask {
    Compute comp; //The handle for a ComputeEngine remote object
    Compute compClone;
    Task taskA = <Task for ComputeEngine>;
    Task taskB = <Task for ComputeEngine>;
    String codebase = <The URL from which the ComputeEngine class has to be downloaded>;
    String remoteHost = <The host name where ComputeEngine must be created>;

    public static void main(String[] args) {
        try {
            comp = (Compute) MURO.create("ComputeEngine", remoteHost, Mobile.DEFAULT_PORT, codebase);

            resul = comp.executeTask(taskA); //Makes ComputeEngine execute taskA
            compClone = (Compute) comp.remoteClone(<another host>); //Clone the object on another host
            comp.move(<a third host>); //Moves ComputeEngine to another host
            resul = comp.executeTask(taskB); //Makes the ComputeEngine execute taskB

            ...

            } catch (RemoteCreationException r) {...}
            catch (MigrationException m) {...}
            catch (RemoteException e) {...}
        }
    }
```

Client-side code: remotely creates an object and move it to another host (maintaining the capability of interacting with the object through remote method invocation).

The ComputeEngine only has to extend the MobileUnicastRemoteObject class and implements its own remote interface (the same way of standard RMI)
Implementation

Usually mobile object systems exploit Java RMI only at this level as transport facility.

MobileRMI modifies the RMI system at this level to obtain mobility enabled references.

MobileRMI
- has been built by modifying SUN’s RMI source code;
- is portable.

Reference updating

Reference updating takes place through a distributed logging facility (a dummy object replaces the migrating server on the departure site).
Reference Updating Through GC Messages

Reference updating at almost no cost by piggy-backing garbage collection messages exchanged among clients and servers:

- When a client renews the lease time associated with an out-of-date reference (dirty call (1)), the return message is piggy-backed (2) with the reference to the next element in the chain.
- The client has its reference updated (3) to the next element in the path.
- The garbage collector removes a dummy-object as soon as it is no more useful.

Contraction of the dummy-object chain

The dummy-object chain contract itself allowing clients to reach the migrating server in a lower number of steps:

Dummy-objects are themselves server objects and so reference updating by means of GC messages applies also to them.
Conclusion

• Instead of providing a set of mobility libraries on top of the Java programming environment, MobileRMI integrates the mobility primitives inside an existing middleware;

• It is possible to move components from a JVM to another by means of a straightforward syntax;

• As a side effect, MobileRMI shows how it is possible to turn an existing middleware into a mobile object system;

Application scenario

• MobileRMI is intended for slow application mobility.
• The server can relocate itself to
  – minimize the mean delay;
  – maximize the system fairness.
Our Testbed

- NS/VINT with CMU wireless extensions
- Up to 10 real machines to execute application nodes (middleware+application). Other mobile nodes can be simulated.
- The simulator acts like a router allowing real-world traffic to be passed through without being manipulated.
- The ns packet contain a pointer to the network packet. Network packets may be dropped, delayed, re-ordered or duplicated by the simulator.

An Example of Use (1)

- We are currently using the testbed to study *Logical Mobility over Physical Mobility*
- Through a case study, we try to give an answer to the questions
  - Where migrate ?
  - When migrate ?
  - Which are performances of logically mobile applications against static ones ?
- Application model under test:
  - 1 server and many clients;
  - Periodically each client sends a message to the server;
  - On receiving a message, the server delivers it to all other clients.
An Example of Use (2)

Migration strategy: the server migrates towards the physical barycenter of the network.

A Didactic Experiment (1)

- Uses an hand-made scenario with 4 nodes;
- Communication range is about 250m;
- Clients and server communicate by using Remote Method Invocation (RMI);
- Performance indexes: generated traffic and communication latency.
- Movement:
  - At time 200s Node1 starts moving towards Node0;
  - At time 400s Node0 starts moving towards the initial position of Node1;
  - ...

The mobile application
- Method invocation delay (mean value): 36.9.4 ms
- Generated traffic: 7.2e+06 Bytes

The static application
- Method invocation delay (mean value): 40.4 ms
- Generated traffic: 9.4e+06 Bytes