MobileRMI

MobileRMI

- MobileRMI is a mobility layer embedded into the Java Remote Method Invocation system:
  - Integration of logical mobility with a well-known 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 transparency vs location awareness**
- Remote method invocation provides location transparency: 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 transparency 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.

```
public static void main(String[] args) {
    Server remoteHost = <The host name where ComputeEngine class has to be downloaded>; //Server Remote Reference
    String codebase = <The URL from which the ComputeEngine class has to be downloaded>; //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).
    Compute comp;  //The handle for a ComputeEngine remote object
    try {
        comp = (Compute)MURO.create("ComputeEngine", remoteHost, codebase);
        resul = comp.executeTask(taskB);  //Makes the ComputeEngine execute taskB
        //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).
    } catch (RemoteCreationException r) {...}
    catch (RemoteException r) {...}
    catch (RemoteException r) {...}
}
```

```
Client move(<B>)

A
Server
B
Client
```

Programming with MobileRMI

```
public class ComputeTask {
    Compute comp;  //The handle for a ComputeEngine remote object
    public ComputeTask(Compute comp) { ... }
    public static void main(String[] args) {
        try {
            comp = (Compute)MURO.create("ComputeEngine", remoteHost, codebase);
            resul = comp.executeTask(taskA);
        } catch (RemoteCreationException r) {...}
    }
    public void executeTask(String task) {
        System.out.println("task is "+task);
    }
}
```

```
Move(<B>)

Server
Compute
Server
```

Implementation

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

```
<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stub</td>
<td>Server Remote Reference</td>
</tr>
<tr>
<td>Remote Reference</td>
<td>Skeleton (opt)</td>
</tr>
</tbody>
</table>
```

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 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 shortens 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.
Logical over Physical Mobility

- In the following we illustrate a migration pattern for applications running in a MANET, and validate experimentally its effectiveness.
- We discuss how logical mobility can increase the performance of applications, by reducing:
  - the consumed bandwidth
  - communication delay
- The engineering approach acts upon the idea that the communication scheme at the application level should be organized to match, as far as possible, the underlying communication network.

MANETs

- In Mobile Ad-hoc Networks (MANETs) multi-hop connectivity allows each host to reach hosts beyond the radio range through the use of routing provided by neighbors.
Assumptions

- Application Made of 1 server and n clients. Clients need to communicate with the server and vice-versa. Many real applications fit into this model (network games, distributed whiteboard, messaging applications,...)
- Middleware Programming model is distributed object based. It supports logical mobility through primitives to move code and objects from an host to another.
- Network Each node is able to host a client or/and the server. Nodes move freely in a given bidimensional area.

Application scenario

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

The Migration Pattern

- The server is the only mobile component.
- It moves to reduce the overall communication cost.
- Communication cost between two nodes is defined in terms of number of hops.
- The server determines the position of all clients and migrates to the node that is closer to the physical baricenter of the network.
- Expected benefits:
  - Minimize the mean delay;
  - Reduce bandwidth consumption;
  - Maximize the system fairness;
  - Increase server availability (since central nodes are usually less subject to be partitioned from the network).
An Example

Communication cost reduced by moving the server to node 4, the closest to the center of gravity.

The Experiment

We compared the performance of a 4-node test application when mobile and static.

- Communication range is about 250m;
- Clients and server communicate by using Remote Method Invocation;
- 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

Results

- The mobile application
  - Communication latency: 36.9 ms (8% better)
  - Generated traffic: 7.2e+06 Bytes (24% better)

- The static application
  - Communication latency: 40.4 ms
  - Generated traffic: 9.4e+06 Bytes (24% better)
Conclusions

- 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.