Abstract
Mark Weiser suggested that computing devices should be assisting human beings in their day to day routine life activities, staying invisible from their attention. In such type of computing scenario, called Ubiquitous Computing, a very large number of computers are required to identify, connect and communicate with each other dynamically, without any break in service to their users.

In this paper, a design of middleware layer has been suggested. It has been implemented using JAVA Remote Method Invocation (RMI). This middleware design facilitates communication among heterogeneous devices. Due to inherent characteristics of platform independence and portability, Java RMI is an appropriate technology to implement middleware layer.

Keywords: Middleware, Remote Method Invocation, Heterogeneous Systems

1. Introduction
Basic concept of Ubiquitous Computing [1] is to provide break free continual service to the users, without restricting them to one single place. Infrastructure, required to meet this purpose, shall contain large number of service providing distributed computers to support mobility of users, covering the complete path of movement of user with wireless connectivity.

As there shall be very large number of service seeking users in ubiquitous computing environment, each carrying many wireless mobile devices, so there has to be sufficient number of service providers so that all the service seekers may stay connected without any break in service. Computing devices expected to be present in any Ubiquitous Computing Environment may be broadly categorized as below

(i) Fixed, portable or mobile computing devices -- categorized on the basis of their mobility and portability,
(ii) Battery operated or non battery operated devices -- categorized on the basis of source of their power.
(iii) High computational power or low computational power devices -- categorized on the basis of speed of processor, size of RAM etc.
(iv) Homogeneous or Heterogeneous -- categorized on the basis of whether similar and different type of Operating Systems are used.

Ubiquity of service applications requires interoperability and co-ordination in between large number of computers constituting the basic infrastructure for ubiquitous computing. Transparency is the primary requirement of ubiquitous computing. It specifies that ubiquitous computing system is inherently a distributed computing system. User should not be concerned about physical location of the service and node to node migration of service. This requires handling the issues of service discovery and activation, mobility of...
service users’ devices, frequent connectivity and disconnection, and resumption of service.

These facilities have to be incorporated through context sensitive administrative processes, in any layer below application layer. Heterogeneous platform of large number of computers, in the service providing infrastructure presents another problem of compatibility of computing devices in service infrastructure. As a result of that it is complex to implement this feature at the level of operating system.

The platform independent technology used in Java programming language presents a good solution to the problem of incompatible system. Using Java RMI, a middleware layer is developed for seamless service availability throughout heterogeneous system. RMI is used to create methods/processes/services which may be accessed from user applications on remote machines connected to the computers in the infrastructure.

This paper is organized into four sections. Section 2 introduces to architecture of RMI for developing distributed applications. Section 3 introduces design of middleware for ubiquitous computing and section 4 provides implementation details of ubiquitous computing using Java RMI. Section 6 concludes this paper.

2. Distributed Applications Using Remote Method Invocation

Java Remote Method Invocation fulfills the one of the basic requirement of distributed applications i.e. using Java RMI a method available on remote machine can be invoked by an object running on local machine. This feature allows the service on any machine to be accessed by any another process/service located on same or any other machine which is connected to the owner node of the service. However Java RMI does not support transparency inherently. It has to be built over the basic services provided by Java RMI system. Java RMI may be divided in three layers (Fig 1)
1. Application layer
2. Client and Server Stub
3. RMI Registry

RMI registry keeps a directory of various services, which may be accessed by any process on other device connected. It handles transmissions of call messages on the network between client process and service owner node.

![Fig 1. Remote Method Invocation Mechanism in Java](image)

Client stub act as interface between client process and client machine’s RMI runtime. On receipt of call from client process client stub packs it and forwards to RMI runtime, which in turn forwards it to Server Stub. On receipt of reply from server side RMI runtime, client stub unpacks it and
delivers to the client process. Server stub performs similar task on server side.

3. Proposed design of middleware for ubiquitous computing

On the basis of utility of ubiquitous computing environment to its users, it may be divided in two sections as mentioned below-

(i) service infrastructure and
(ii) mobile users

3.1 Service Infrastructure

Ubiquitous infrastructure contains fixed or portable computing devices which forms the part of service rendering system. The complete area of user mobility is to be covered by these devices. So this area has to be divided into different sections called active areas. All the devices in any particular active area are considered to be connected in wired or wireless LAN forming a distributed system of services with the help of middleware layer, to be implemented using Java RMI, as shown in Fig 2. With the help of RMI any remote method can be accessed same way as local method. Middleware layer present in infrastructure devices makes services on these devices platform independent and transparent, which are essential attributes of distributed applications.

3.2 Mobile User

Users in ubiquitous computing environment may move freely from one place to another. Sufficient number of ubiquitous system devices along path of movement provides service without any break. Unique identification technique like RFID [2][3] may help to automatically detect user device and process without any explicit network setting.

4. Implementation of Ubiquitous Computing Environment using Java Remote Method Invocation

Java provide a separate ‘java.rmi’ package [4][5][6] to support remote method invocation. Following code

```
public interface ServiceInterface extends Remote {
    ret_type remoteMethod( list of Args..) throws RemoteException
}
```

create ‘ServiceInterface’ for client to access service remotely. Service need to implement this interface and define ‘remoteMethod’ to be remotely available

```
public class RemoteService extends UnicastRemoteObject implements ServiceInterface
```

Now this service need to be bind with appropriate proxy name in RMI Registry using as below

```
RemoteService remoteServiceObj=new RemoteService();
Naming.rebind("RemoteServer", remoteServiceObj);
```

Client process use ‘lookup’ for ‘RemoteServer’ to access the ‘RemoteService’ method.

```
string remoteServerURL= “rmi://+ /ip_add of remoteMachine+/RemoteServer”;
```

ServiceInterface object is created to refer to RemoteServer
ServiceInterface serviceInterface=(ServiceInterface)
Naming.lookup(remoteServerURL);

‘serviceInterface.remoteMethod()’ may be used to access remote method. Thus using Java RMI, a method on remote machine may be accessed with local semantics. However, this does not truly create a distributed service as user needs to specify network path of the remote service. This model of service access does not satisfy features access and location transparency of ubiquitous applications.

With the help directory servers keeping record of available service, a middleware layer is created for ubiquitous access of distributed application

4.1 Directory server for ubiquitous services

Whenever a new service is created in the ubiquitous system, its interface transmits its location to designated server or set of server. Directory server/ servers for services may be organized following three ways

(i) Centralized Directory Server
One single server per active area containing information of all service in the system

(ii) Distributed Directory Servers
Set of designated distributed servers in every active area contains information of sublist of service in the system as shown in Fig 3

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Service Binding</th>
<th>URL of Owner Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains Entry for a subset of services in the system</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(iii) Dynamically Distributed Directory Servers
Every node in active area contains complete information of all services in the system as shown in Fig 4

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Service Binding</th>
<th>URL of Owner Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains an entry for each service in the system</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig 4. Every node in active area maintain entry of all services in the system

When all service entries are maintained at all nodes then the updation process increases network traffic and memory space requirement (Fig4).

<table>
<thead>
<tr>
<th>Remote Method</th>
<th>Method Attribute</th>
<th>Utility of Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig 5. Table Contain List of Methods for each service (One table for each service)

The standard user interface on user device contains information about service directory in lookup server. There are two alternate ways of updating information about service coming in and going out. First, the user interface updates its list from directory of lookup servers. Alternately, Directory Servers transmit the updated information to the users in the active area. Second option is viable only with limited number in users in active area and small number of changes in service directory.
5. Conclusion

Due to inherent characteristic of platform independence, service/process developed in Java may work in heterogeneous environment. Middleware layer works over Operating System environment and protocol stack. So, this property makes it an ideal language to develop middleware layer for ubiquitous computing.

Universal Plug and Play (UPnP) protocol provide service for online multimedia streaming and and Service Location Protocol (SLP) provide location of any service available on internet. But these technologies does not provide facility to directly access any method based service. These technologies works on the basis of client-server model and are not truly distributed in nature. Java RMI, does not support access and location transparency of applications as well. With the help of service directory, property of transparency is implemented. Thus middleware service for service access in ubiquitous environment is created.

References:


