# Middleware for Grid Computing on Mobile Phones

Muthoni Masinde and Antoine Bagula
*University of Cape Town, South Africa*

Victor Ndegwa
*University of Nairobi, Kenya*

## 1 Need for Mobile Phone Applications in Africa

The low Internet penetration and lack of electricity in the rural areas of the developing countries of Africa make the use of computer-based solutions a big challenge. Yet there is dire need of such applications in these areas. Luckily, most of these countries have reported impressive adoption levels of mobile phones [3], a phenomenon that is now creating a paradigm shift; computing is slowly moving from the traditional PC to the phone. Coincidentally, advancements in the smartphone technology have produced such powerful gadgets that can ably compete with PCs of the 21st century. Today, for less than US$ 400, one can acquire a smartphone equipped with; 1000MHz clock speed, 512MiB (ROM +RAM), access to several types of data networks (CSD, HSCSD, GPRS, EDGE), and Wireless local-area network (WLAN) among other features [6]. With this kind of computing power, computer analysts/programmers can now develop both scientific and commercial applications to address numerous challenging facing poor people in the developing countries of Africa.

## 1.1 Using Mobile Phone as a Computing Device –The Challenges

A few challenges related to the use of mobile phones as computing devices need to be addressed before we see a wide deployment of mobile phone applications. Some of these are:

(I) Mobility – phones are highly mobile and their computation power may not be available where/when it is needed;

(II) Heavily reliance on battery power that gets drained fast especially under heavy computational tasks;

(III) Different hardware and software leading to high levels of heterogeneity among mobile phones;

(IV) Most of the phones found in the developing countries of Africa are low-end and may not provide the required computing power; and

(V) Mobile phones are highly personalized; the permission to use them for computational tasks may not always be granted.

## 1.2 Grid Computing on Mobile Phones

Grid computing on mobile phones is one way of addressing some of the above challenges. Computer grids have successfully been utilized to develop gigantic computer solutions especially for scientific use ([2] and [5]). This success can be replicated on a mobile phone environment as long a generic grid middleware precedes this. Grid computing middleware (hardware and/or software) is very critical for the operation of the nodes, its main roles being to support Single System Image (SSI) and to ensure enhanced system availability. Failure of a node for instance should not affect the operation of the system in anyway. To achieve this, the middleware enables the nodes to take advantage of services available in the grid transparently hence freeing the end user from having to know where an application will run. It also makes it possible to view all system resources and activities from any node as well as supports check pointing [7]. The latter occurs when

process state and intermediate results are saved periodically to ease process migration when and if needed. MobiGrid [4] is an implementation of mobile phone grid middleware.

Like any conventional grid middleware, MobiGrid is a middleware prototype for mobile phones that provides an API on which distributed applications can be built. MobiGrid's uniqueness lies in the fact that the middleware is for mobile phones environment. MobiGrid was developed to bridge the technological gap that exists in the rural areas of the developing countries of Africa where the adoption of mobile phones technology is higher than that of other forms of ICTs. By using MobiGrid, computer-based applications that are currently a reserve for the developed countries and for resource-endowed cities of Africa can become a reality for all. Consequently, the much needed custom applications such as e-heath, e-education, e-farming, e-weather forecast and so on can then be implemented as mobile phone applications for use in the remote villages where they are needed the most.

## 1.3 MobiGrid Version 1 – The Limitations

Originally, MobiGrid was implemented using the Python programming language for S60 (pyS60) and tested on only two phone models: the Nokia E63 and Nokia N95. As described in the following sections, MobiGrid was designed based on a client/coordinator model where each phone can run either as a local/client server that manages the resources of the phone and responds to requests from the coordinator to provide a registered service to the mobile grid or optionally run a coordinator server that responds to requests from the client servers and monitors their health to ensures they are still running. In its earlier implementation, MobiGrid had several limitations; the two major ones being:

(I) MobiGrid could only recognize coordinators using the subnet mask 255.255.255.0. This was due to the lack of support for broadcast in PyS60, requiring use of a work-around. The implemented work-around was too slow for use with a wide range

of IP addresses (e.g. addresses using mask 255.255.0.0) taking up to 5 minutes to find the coordinator. As a result, the middleware could only be used by up to a maximum of 254 nodes on the same network;

(II) For MobiGrid to function properly, it required that the two main modules (Local Server and Coordinator Server) be installed on each of the phones participating in the grid. However, due to limitations of S60 (at the time of developing the application), a phone could not run more than one instance of the application. This made it impossible to come up with a custom application to test MobiGrid.

In this chapter, the rich tools provided by the Android Framework[1] have been employed to address the above limitations. In particular, the account manager and the synchronization manager (through a sync adapter) have been used extensively. Using Android Virtual Device (AVD) tool, it is now possible to simulate a grid with hundreds of virtual phones, analyse and visualize performance of the grid using tools provided by the rich android.graphics class. Finally, it is now easier to control and manipulate the grid by remotely (using telnet) executing various commands such as for checking battery, network and memory status of the phones (virtual and real) participating in MobiGrid.

# 2 Analysis, Design and Implementation of MobiGrid

Both Exploratory and Operational Prototyping were used in developing MobiGrid. Exploratory prototyping was used to build and test individual features and once a feature had been fully tested, it was added to the operational prototype and further evolved.

## 2.1 MobiGrid Functional Requirements

MobiGrid should be able to

(I) locate the coordinator and register with it;

(II) tell if the coordinator has failed;

(III) call for an election if no coordinator was available;

(IV) participate in an election and take over the role of coordinator if elected;

(V) register new services provided by applications running on the phone;

(VI) service request for services that were registered with the MobiGrid;

(VII) provide a messaging system for communication; and

(VIII) provide an interface through which applications could use the grid.

## 2.2 MobiGrid Design

MobiGrid was designed to provide a middleware service that is always started and runs in the background to allow distributed applications to operate. This middleware service abstracts the developer of the distributed application from having to deal with the technicality of locating distributed services and managing remote nodes. The typical operation of MobiGrid is as follows:

### 2.1.1 Initialization

(I) The user starts the MobiGrid service on his/her phone by selecting it from the menu. The assumption here is that MobiGrid is already installed on his/her phone.

(II) MobiGrid searches for other phones running MobiGrid.

(III) If a phone running MobiGrid and acting as the coordinator is found, connect to this coordinator as a client.

(IV) If no coordinator is found, call for an election. The winner of the election becomes the new coordinator. If the caller of the election is not challenged, the caller is elected unopposed.

(V) Once a coordinator has been found, MobiGrid can connect to the coordinator and register any services it may have available. These services are provided by applications that use MobiGrid running on the phone.

(VI) If an application requires a service from MobiGrid, it passes a message to MobiGrid, which then provides the service or requests the service from the coordinator.

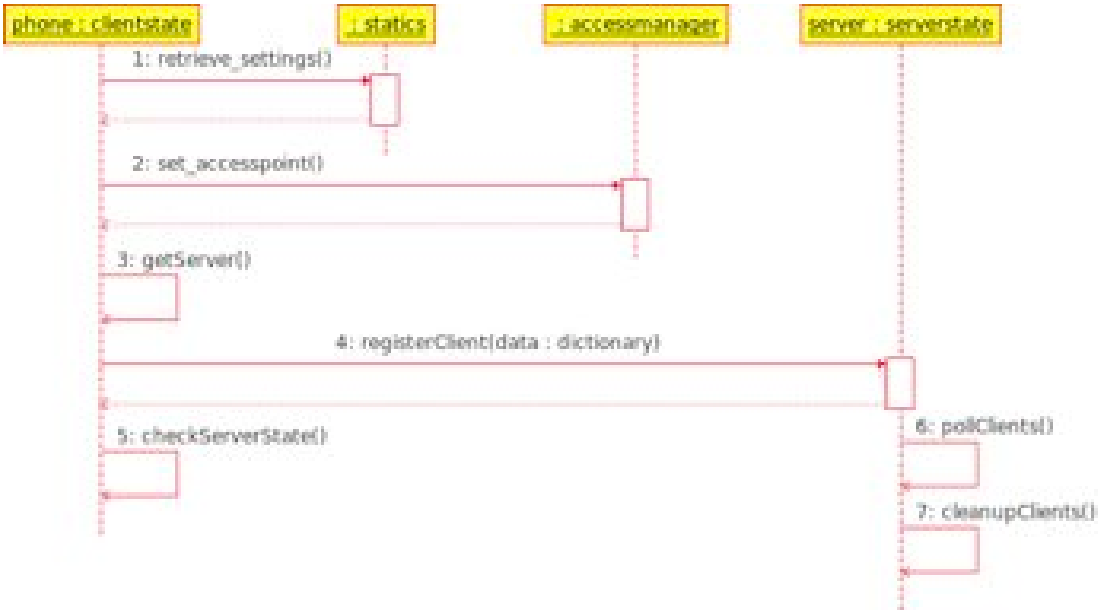


Figure 1: Initialization Sequence

Each phone runs a client/local server and may optionally run a coordinator server if it is the coordinator. The client server manages resources on the client phone and responds to requests from the coordinator (e.g. to provide a registered service). The coordinator server responds to requests from the client servers and monitors their health (i.e. ensures they are still running).

### 2.2.2 Finding the Coordinator

The coordinator is responsible for the following:

(I) determining which nodes are part of the grid;

(II) maintaining a dictionary of the services available on the grid and which nodes are offering them;

(III) checking which nodes are up and which have failed and

(IV) servicing request for services by finding the node offering the service requested and forwarding the request

The first node to join the grid automatically becomes the coordinator and other nodes register with it on joining. The steps involved in finding the coordinator are as follows:

(I) The client broadcasts to a given SERVER_PORT (default is port 2904) asking who the coordinator is

(II) The coordinator responds and registers the client. The client also registers the IP address of the coordinator

(III) If no response is received within a designated TIMEOUT period, the client calls for an election by broadcasting an !Election message to a designated CONTROL_PORT (default is port 7609). If a node with better election attributes receives a call for an election, it responds by calling its own election. If the caller of the election is not challenged, it automatically becomes the new coordinator

### 2.2.3 Registering/Requesting Services

When an application that uses the MobiGrid is started, it automatically registers services it offers with the coordinator by sending a !Service command. The coordinator then adds the service and a description of the service to a dictionary of services owned by that node. If/when the coordinator fail, all the services will have to be re-registered with the new coordinator.

A node can also request a service by sending a !Request command with the appropriate parameters (depending on the service). For example, PlotGragh(x,y,z) to request for a service to plot a graph given the graph coordinates (x,y,z).

The coordinator process receives and processes request from the local server running on the individual handset. It sends its response to the local server, which then forward the response to the application that made the request

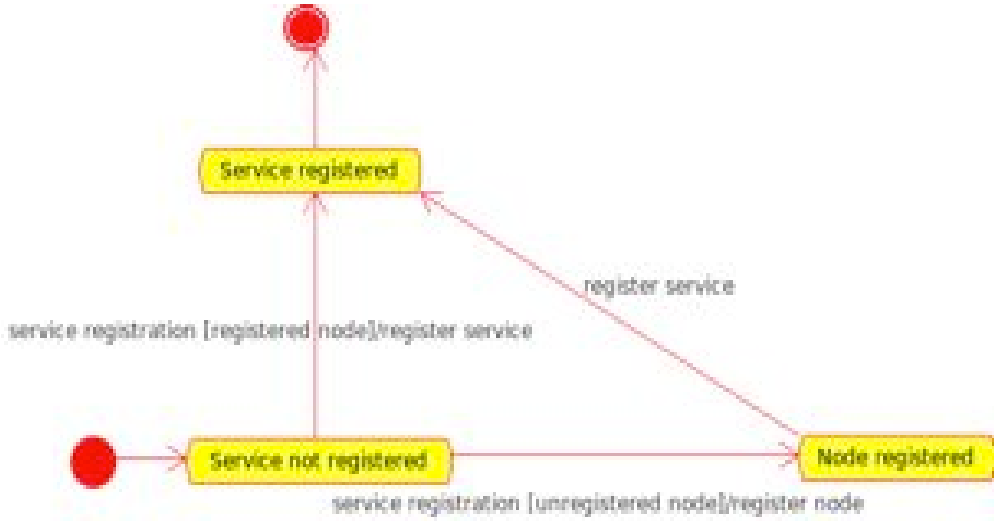


Figure 2: Add a Service - State Diagram

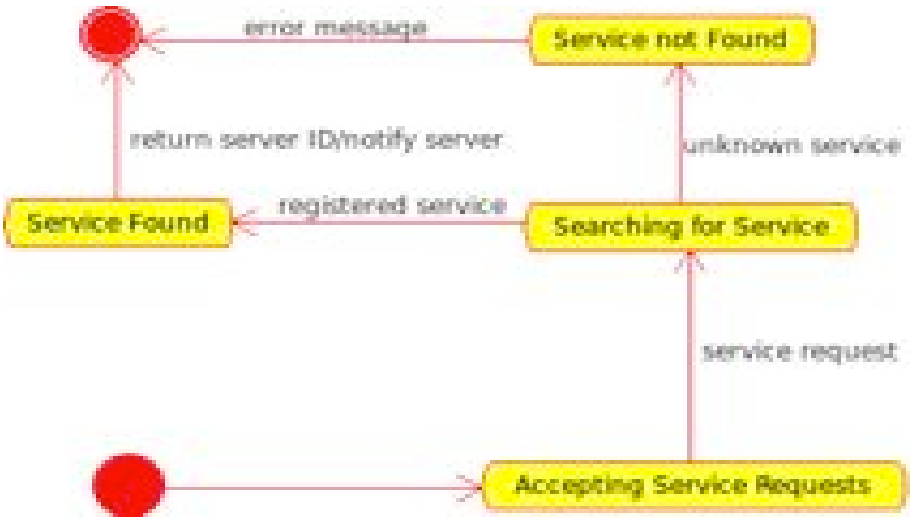


Figure 3: Request Service - State Diagram

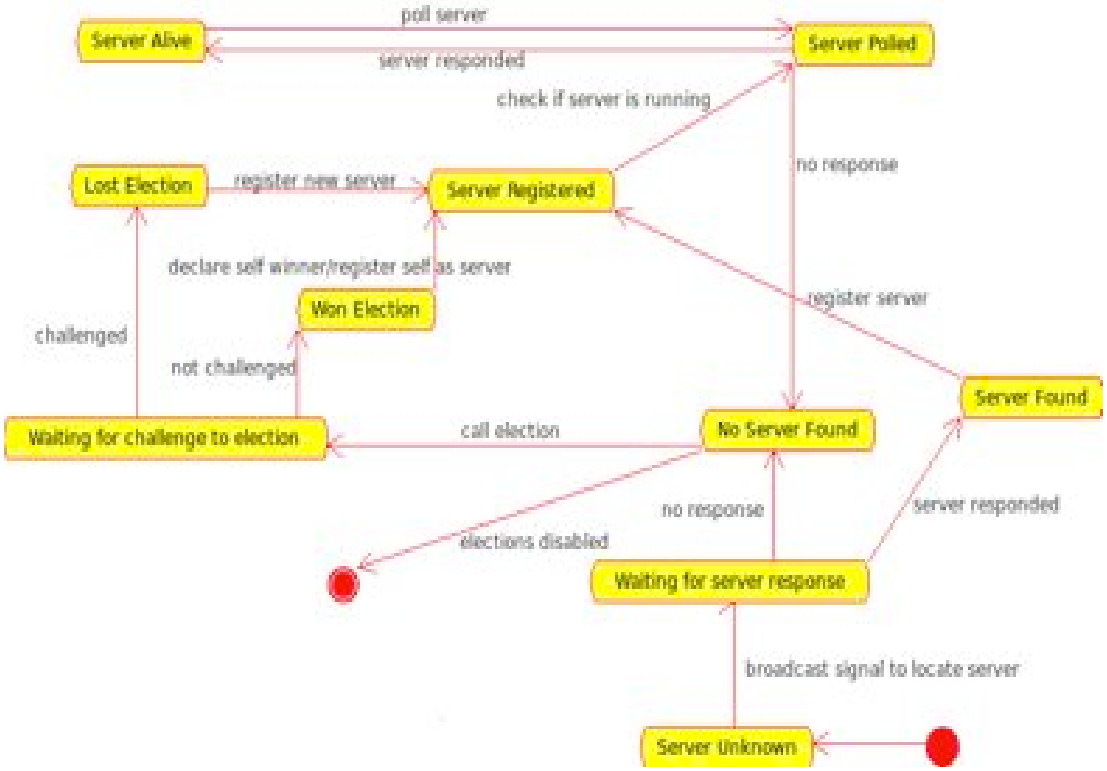


Figure 4: Elections - State Diagram

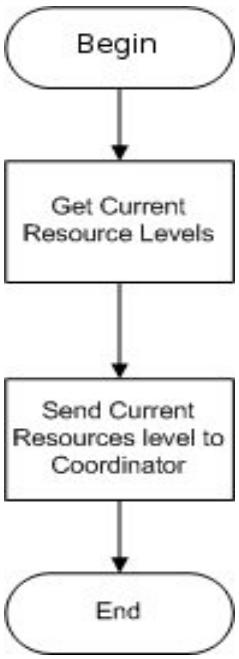


Figure 5: Responding to Polls

### 2.2.4 MobiGrid Coordination

The coordinator recognizes the following commands:

(I) **!Register** —register a new client by calling registerClient

(II) **!Poll** —client checking if server is still alive. It responds by calling notifyStillAlive

(III) **!PollResponse** —receive a response from a client node indicating that the client is still up and running. Update status of client by calling updateClientState

(IV) **!Request** —process a service request from the client by calling offerService

(V) **!Service** —register a new service from a client by calling addService

(VI) **!Quit** —shut down the coordinator by calling shutdown

### 2.2.5 Local Server Operation

Local server is a client of the coordinator but it is itself a server that is charge of any custom applications that may be installed on the handset

The local server recognizes the following commands:

(I) **!Election** —respond to an election by calling acknowledgeElections

(II) **!Winner** —declare self as winner of an election by calling signalRegisterServer

(III) **!Poll** —respond to a poll from the server to indicate that the client is still alive by calling respondToPoll

(IV) **!Alive —**a reply from the server indicating that the server is up and running. Processes using updateServerStatus

(V)**!Service** —register a service with the coordinator by calling registerService

(VI) **!Serve —**offer a service on request from the coordinator by calling doServe

(VII) **!Confirm** —a confirmation from the coordinator that the service offered by the client has been registered successfully. Processed by calling serviceConfirmed

(VIII) **!Request —**request a service from the coordinator by calling requestService

(IX) **!Found** —a notification from the server that a requested service was found. Processed by calling notifyClient

(X) **!Quit** —shut down the local server by calling shutdown

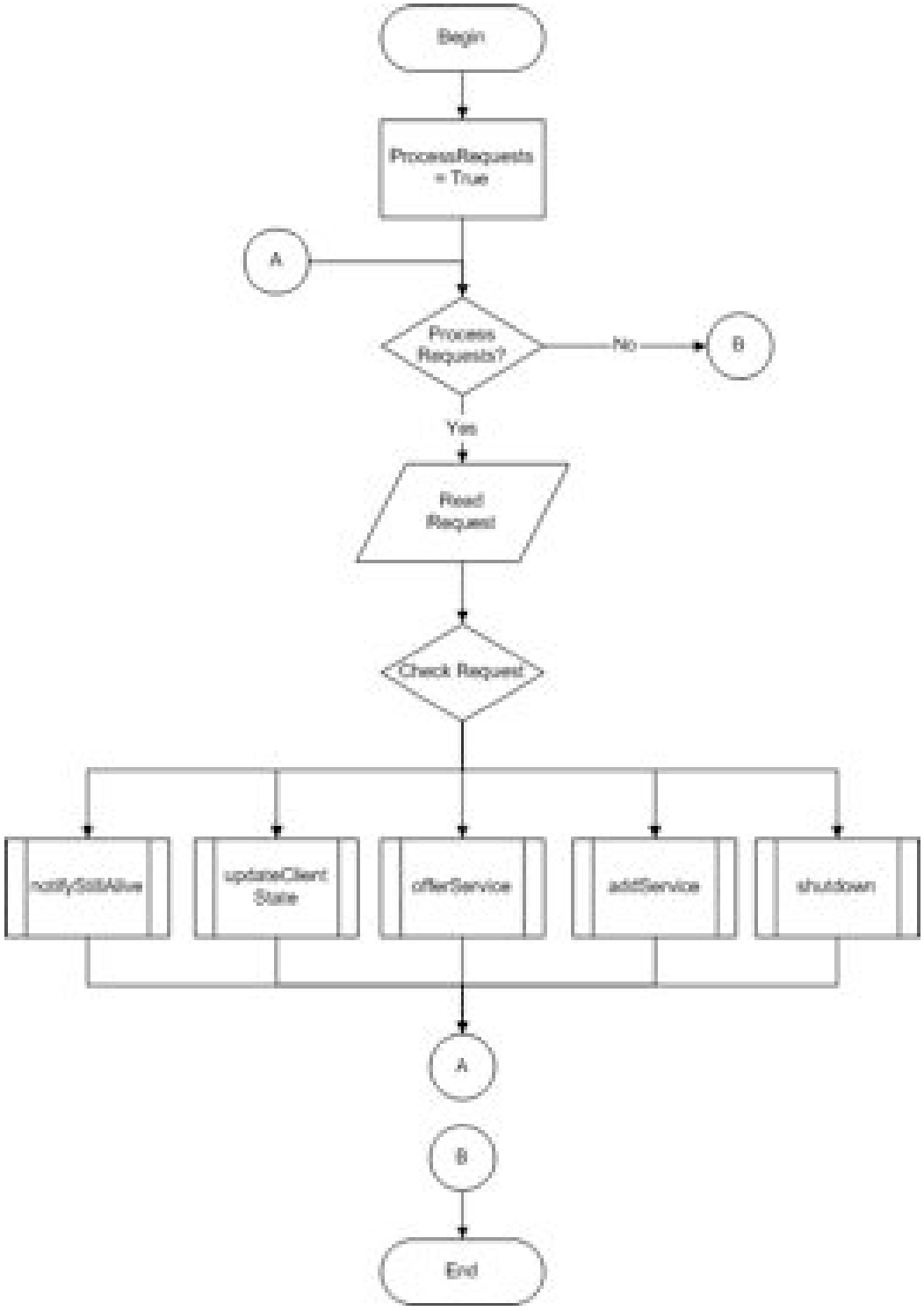


Figure 6: MobiGrid Coordination

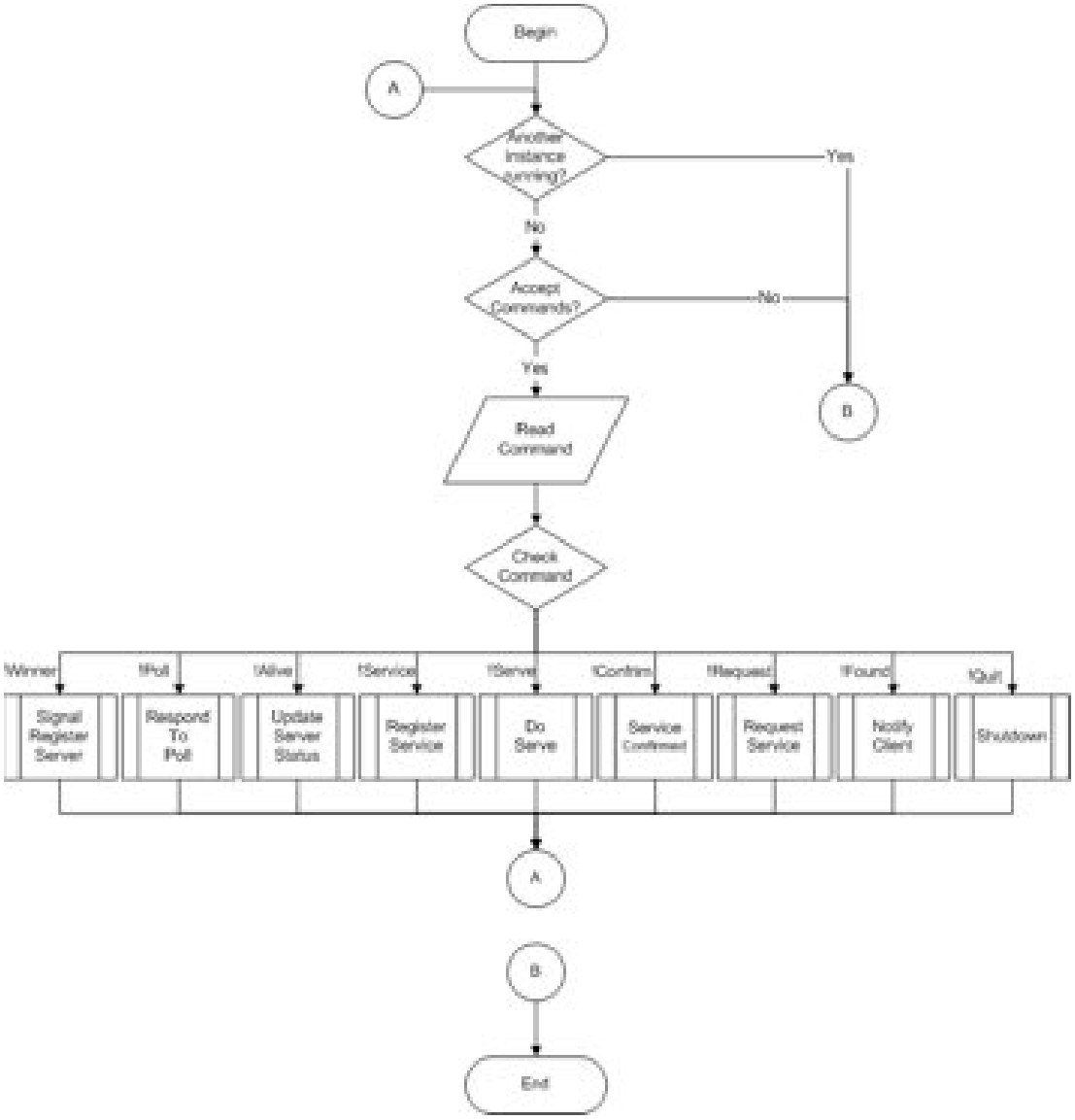


Figure 7: Local Server Operations

## 2.2.6 Keeping Track of Nodes

Like in any other grid, the coordinator has to keep track of which nodes are still up and which have failed. The nodes also need to continually check if the server is up and running. This is achieved by the nodes and the coordinator polling each other at regular

intervals and checking and keeping count of the number of times a node/coordinator has failed to respond.

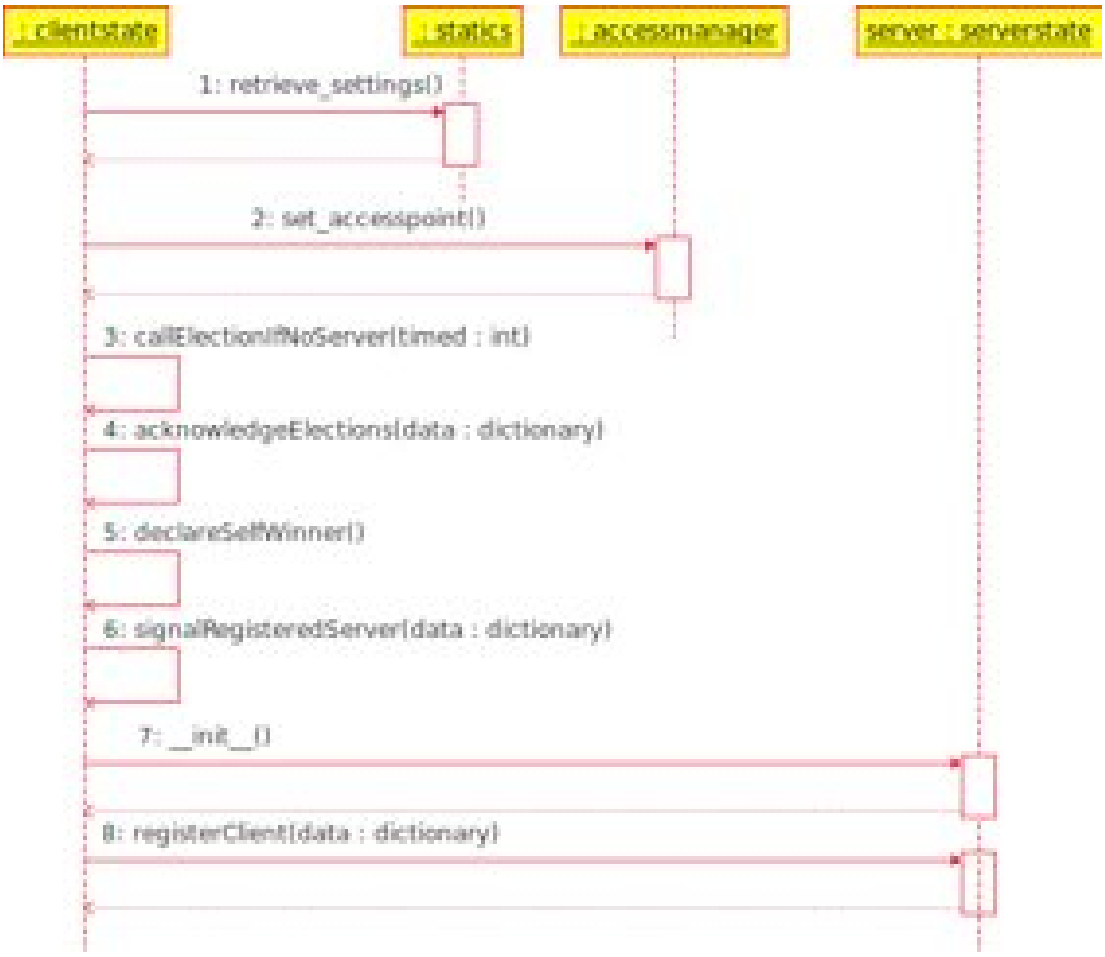


Figure 8: Keeping Track of Nodes

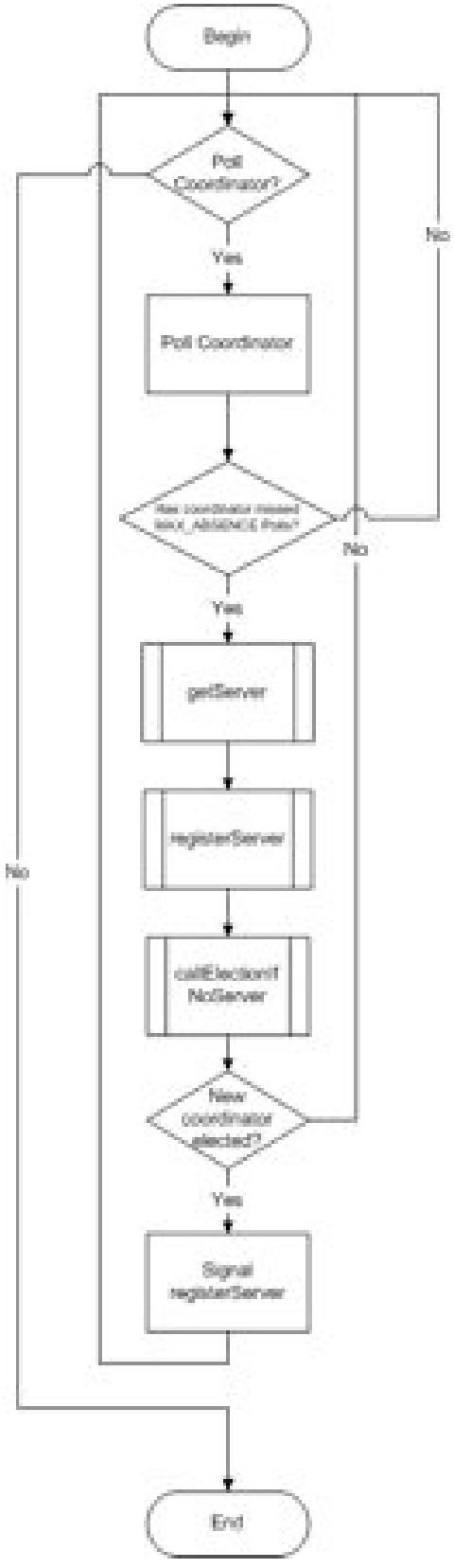


Figure 9: Checking state of the coordinator

## 2.3 MobiGrid Version 2 Implementation

MobiGrid Version 2 is being implemented using Android SDK 2.2 with the latest (0.97 at the time of implementing MobiGrid) version of Android Virtual Device (AVD) plug-in for Eclipse 3.6. The development (main) machine being used is MacBook Pro running Mac OS X 10.5.8 (9L31a). At the time of writing this, MobiGrid had not been tested on actual phones but was extensively tested on several AVDs.

To depict the design presented under MobiGrid Design as well as maintaining the structure of MobiGrid Version 1, two main modules have been implemented:

(I) **Coordinator Module** to run on the phone acting as the coordinator for the grid

(II) **Local Server Module** to run on all the other handsets and act as the server to manage the handsets' resources (hardware and software) needed/participating in the grid

The design also requires that all the handsets be installed with both modules in order to:

(I) empower all the phones with the ability of playing coordinator role when/if elected

(II) provide a means for servicing request to run applications running on the handset (and are part of the grid)

(III) provide statistics of resource (battery and memory in this case) utilization on the handset

**Note**: The Coordinator Module remains 'dormant' in all the handsets except in the one playing the coordinator role.

# 3 MobiGrid Version 2

Though the implementation of MobiGrid is still on going, the following are the new features that Android has added to MobiGrid Version 1:

## 3.1 Broadcast Communication

Android directly supports broadcast communication through android.content.BroadcastReceive class. This is a base class for code that will receive intents sent by sendBroadcast(). This drastically reduced the communication overhead that was experienced during the communication from the Coordinator instance to the Local Server instances.

## 3.2 Multiple and Graphical Testing

By initializing multiple AVDs, it is now possible to effectively test the various aspects of MobiGrid. This is contrary to the MobiGrid Version 2 that was command based (see screenshot below) and only run on two physical Nokia phone models. Acquiring tens of phones for the tests was an uphill task.

## 3.3 Testing MobiGrid Version 2 – Screen Shots

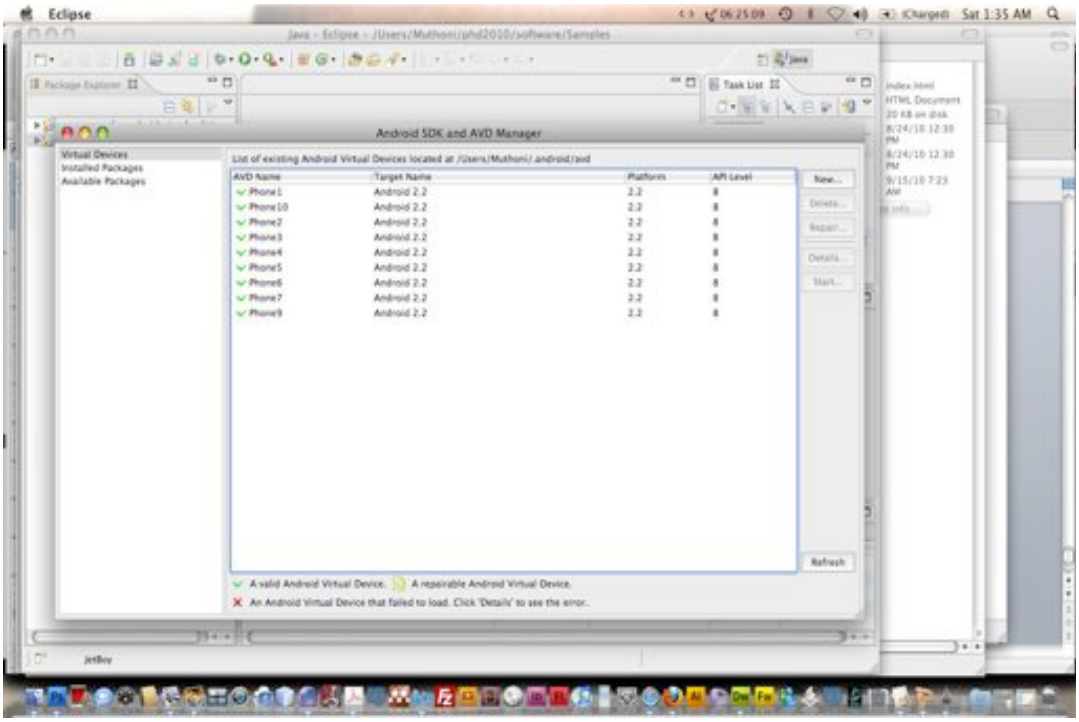


Figure 10: Creating Multiple AVDs

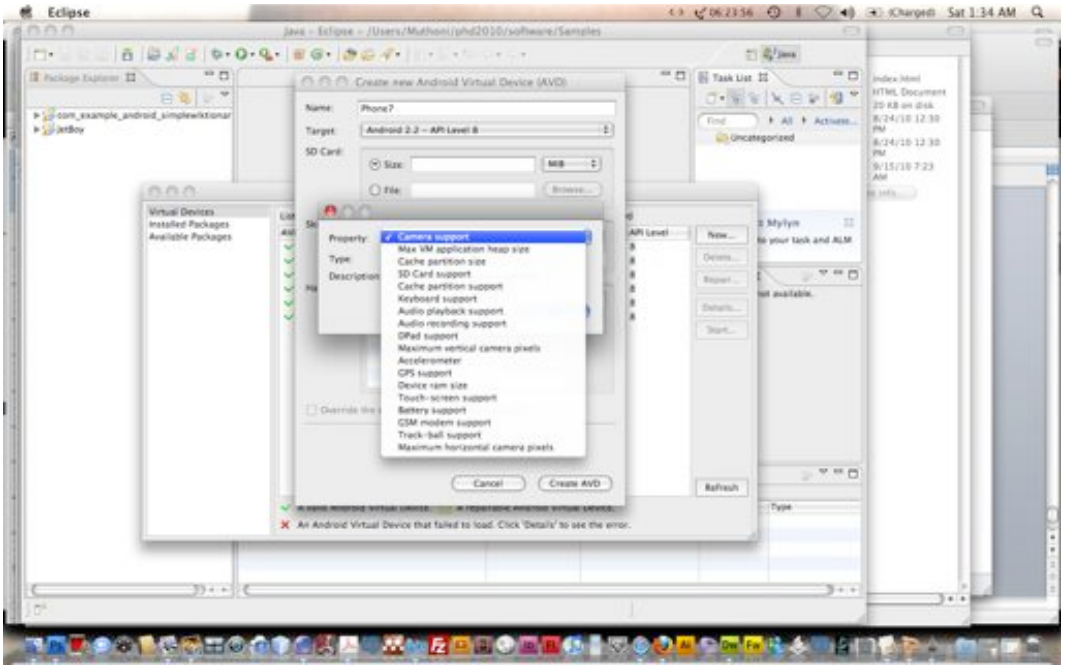


Figure 11: AVDs with Different Features


Figure 12: Running Multiple AVDs

## 3.4 Other Features

In the current implementation, the following features of Android are being utilized to further improve MobiGrid.

### 3.4.1 Location Services

This is to enable both the Coordinator and Local Server get the actual location of services/servers. android.location class is being used to achieve this

### 3.4.2 Improved Security

In MobiGrid Version 1, no efforts were made to impalement security. This is now being incorparated into MobiGrid Version 3 by use of the android.net.wifi subclasses such as WifiConfiguration.AuthAlgorithm and WifiConfiguration.Protocol

### 3.4.3 Centralized Management

Unlike in MobiGrid Version 1, management of all the nodes in the grid is now possible from a central point through telnet. Through this, nodes can be shutdown and battery/memory/network status checked

## References

[1] ANDROID. 2010. Android Developers Reference. 2010.

[2] EU-FP6 2010. EUChinaGrid.

[3] ITU. 2010. Global Telecom Indicators for the World Telecommunication Service Sector. ITU Global Telecom Indicators for the World Telecommunication Service Sector 2010, 3-3. http://www.itu.int/dms_pub/itu-t/oth/23/T23010000040001PDFE.pdf.

[4] MASINDE, M., ANTOINE, B. AND VICTOR, M. 2010. MobiGrid:A Middleware for Integrating Mobile Phone and Grid Computing. In The 6th International Conference on Network and Service Management (CNSM 2010), Niagara Falls, Canada, October 25 - 29, IEEE COMMUNICATIONS SOCIETY, Ed. IEEE Communications Society, IEEE Communications Society.

[5] PANDE LAB STANFORD UNIVERSITY. 2010. Folding@home - a distributed computing project 2010,.

[6] PDADB.NET. 2010. Lenovo LePhone. 2010.

[7] RAJKUMAR, B. 1999. High Performance Cluster Computing: Architectures and Systems. Prentice Hall, PTR.