Agent-based Ad-hoc On-demand routing over Ad-Hoc Wireless Networks

Graham Sinclair

Submitted in partial fulfilment of the requirements of Napier University for the Degree of BEng with Honours in Software Engineering

Supervised by Dr. Bill Buchanan

School of Computing
2003
I, Graham Sinclair confirm that this dissertation and the work presented in it are my own achievement.

1. Where I have consulted the published work of others this is always clearly attributed;
2. Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;
3. I have acknowledged all main sources of help;
4. If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;
5. I have read and understand the penalties associated with plagiarism.

Signed: Date:

Matriculation no:
Acknowledgement

I would like to thank my supervisor, Dr. Bill Buchanan, for the help, support and encouragement throughout the project, and also for introducing me to the topic, which has been very interesting and has given me a great insight to the future of mobile computing. I would also like to thank Nikos Migas who provided invaluable support throughout the development and testing stages of the project.

Finally I would like to thank Paolo Bellavista from Università degli Studi di Bologna Viale Risorgimento for kindly providing me with code extracts from his Monitoring Application Programming Interface (MAPI) project.
Abstract

This report describes the investigation process used in the study of the use of an agent-based system to dynamically route network traffic over ad-hoc wireless networks. The traditional Network and Systems Management approach relies on a centralised, Client/Server approach where network performance data is stored in a central location. This centralised paradigm has severe scalability problems as it involves increasing amounts of data transfers of management data as the size of the network increases (Gavalas, D., et al 2001). There are many research projects investigating the use of mobile agents to decentralise the transfer of network management data (Gavalas, D., et al 2001, Puliafito, A., and Tomarchio, O., 2000, Lee, J.O. 2000, Papavassiliou, S., et al 2001); the main role of the mobile agents is to migrate throughout the network collecting data as they hop from host to host. This approach eases the problem of increasing traffic congestion but still poses the problem of continued use of central depositories to hold the data collected. This approach highlights the benefits of the mobile agent, which has the ability to migrate from host to host, gathering data as it travels; unfortunately it fails to combat the problem of access to local resources for performance measurements information which could be used to calculate the best routing path through the network.

The research is being carried out to investigate the static agent development as described by Nikos Migas from Napier University, in the papers, (Migas, N., et al 2003², 2003³). This research proposes to develop an agent-based framework “MARIAN” which includes a routing protocol based on mobile and static agents. The main objectives of the project were to investigate the mobile and static agent models for ad-hoc wireless networks and to develop a static agent that would perform the required automated local tests to obtain performance metrics that could be used to route traffic. These static agents will allow for a predefined set of tests to be run with the results stored to a local file or database. The results will then be used to determine a set of metrics for the device. These metrics will be made available to the mobile agents, informing them of the abilities of the device, thus allowing for automatic network configuration, which includes control over the types of network traffic directed to a device.

The report concludes with suggestions for further development of an agent-based framework using the information gathered during the project. The gathering of performance metrics from the Personal Digital Assistant (PDA) proved to be the greatest challenge during the project. This has been due to the lack of current research into the use of the Windows CE Application Programming Interface (API). Further development could include the use of the Java Native Interface (JNI) to obtain further performance metrics from (PDA’s). This would allow for the production of a more accurate routing metric.

Keywords: mobile agents, static agents, wireless networks, ad-hoc wireless networks, agent-based, routing protocols, automatic network configuration.
# Table of Contents

## 1 Introduction .................................................................10
  1.1 Introduction ...................................................................10
  1.2 Background ...................................................................12
  1.3 Scope, Aims and Objectives of the project ...................13

## 2 Theory ..............................................................................16
  1.2 Introduction ...................................................................16
  1.3 Wireless Networks .....................................................16
      2.2.1 Infrastructure .......................................................16
      2.2.2 Infrastructure-less .............................................17
      2.2.3 Wireless versus Wired .......................................19
      2.2.4 Interoperability with fixed networks .................20
      2.2.5 Problems with wireless networks ..................20
  2.3 Software Agents .........................................................21
      2.3.1 Mobile and Static Agents .................................22
      2.3.2 Evolution of Mobile Agents ............................23
      2.3.3 Advantages of Mobile Agents .........................24
      2.3.4 Disadvantages of the mobile agent paradigm ....24
  2.4 Agent-Based Development platforms ......................27
      2.4.1 Implementing the Mobile Agent Paradigm using Java .27
  2.5 Examples of current and future Agent-Based technology ..28
      2.5.1 Current Applications .........................................29
      2.5.2 Future Applications ..........................................29
  2.6 Routing Protocols .......................................................31
  2.7 Benchmarking software ..............................................35
  2.8 Access to system resources .........................................35
      2.8.1 Java Native Interface (JNI) ..................................36
      2.8.2 Java Open Database Connectivity (JDBC) ..........37
  2.9 Conclusion .................................................................37

## 3 Methodology .....................................................................38
  3.1 Introduction ...............................................................38
  3.2 Requirements ............................................................38
      3.2.1 Software Requirements ...................................39
      3.2.2 Hardware Requirements ..................................39
      3.2.3 System Requirements ......................................39
  3.3 Analysis ..........................................................................40
      3.3.1 Software Development ....................................40
      3.3.2 Performance tests ..........................................40
      3.3.3 System-level resources ...................................41
      3.3.4 Permanent Data storage ..................................41
  3.4 Design ...........................................................................42
      3.4.1 Design of tests ................................................42
      3.4.2 Design of data storage and retrieval systems ....44
      3.4.3 Overall design of system .................................45
  3.5 Proposed system design as part of the MARIAN framework 46
# 4 Implementation .................................
4.1 Introduction ........................................
4.2 Implementation of tests ........................
   4.2.1 Client-Server test ............................
   4.2.2 4D bubble sort ................................
   4.2.3 Remote file download test ................
   4.2.4 Total system utilisation test ..............
   4.2.5 Battery Strength test ......................
4.3 Running the tests ..............................
4.4 Data storage and retrieval ...................
4.5 Conclusions ....................................

# 5 Testing and Analysis ..........................
5.1 Introduction ......................................
5.2 Hardware device specifications ..........
5.3 Kernel Level tests ............................
   5.3.1 Bubble sort tests ..........................
   5.3.2 File creation tests ........................
   5.3.3 Client-Server on local device ...........
5.4 Network level tests ...........................
   5.4.1 Remote Client-Server tests ..............
   5.4.2 Remote file download test ..............
   5.4.3 Network traffic monitor test ...........
5.5 Application level tests ....................
   5.5.1 System utilisation test .................
   5.5.2 Java Thread Test ...........................
5.6 Group level test ..............................
5.7 Conclusions ....................................
   5.7.1 Comparison of how each test depreciates battery power ........
   5.7.2 Overall system metrics .................

# 6 Conclusions ....................................
6.1 Evaluation of Achievement ................
6.2 Suggestions for future work ..........
6.3 Gantt Chart of project timescales .......

# 7 References ......................................

# 8 Appendix A ......................................
8.1 Details of all tests implemented ....
8.2 Client-Server test ............................
   8.2.1 Server source code ......................
   8.2.2 Client source code ......................
8.3 Four dimensional bubble sort source code ..
8.4 Remote file test source code ..........
8.5 System utilisation test, BSSTskManager source code ....
8.6 Visual Basic system monitor source code ....
8.7 Sample configuration file ...............
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.8</td>
<td>Sample test file</td>
<td>99</td>
</tr>
<tr>
<td>8.9</td>
<td>Sample Results file from a test-run</td>
<td>101</td>
</tr>
<tr>
<td>8.10</td>
<td>Email response on JDBC status by Sun Microsystems Inc.</td>
<td>102</td>
</tr>
<tr>
<td>8.11</td>
<td>Code extracts from scenario in section 4.4</td>
<td>102</td>
</tr>
<tr>
<td>8.12</td>
<td>Database tables</td>
<td>104</td>
</tr>
<tr>
<td>9.1</td>
<td>CD Contents</td>
<td>105</td>
</tr>
<tr>
<td>9.2</td>
<td>Installation of the Benchmark System Software (BSS)</td>
<td>105</td>
</tr>
<tr>
<td>9.2.1</td>
<td>PC Installation</td>
<td>106</td>
</tr>
<tr>
<td>9.2.2</td>
<td>PDA Installation</td>
<td>108</td>
</tr>
<tr>
<td>9.3</td>
<td>Configuring the software</td>
<td>108</td>
</tr>
<tr>
<td>9.3.1</td>
<td>Set run mode</td>
<td>108</td>
</tr>
<tr>
<td>9.3.2</td>
<td>Load and configure .tst file to automate test run</td>
<td>109</td>
</tr>
<tr>
<td>9.3.4</td>
<td>Using templates to automate test run</td>
<td>110</td>
</tr>
<tr>
<td>9.4</td>
<td>Running the BSS System</td>
<td>111</td>
</tr>
<tr>
<td>9.4.1</td>
<td>Running on PC Based system</td>
<td>111</td>
</tr>
<tr>
<td>9.4.2</td>
<td>Running on PDA</td>
<td>112</td>
</tr>
<tr>
<td>9.5</td>
<td>Example of an automated remote client server test</td>
<td>112</td>
</tr>
<tr>
<td>9.5.1</td>
<td>Server set-up</td>
<td>112</td>
</tr>
<tr>
<td>9.5.2</td>
<td>Client set-up</td>
<td>113</td>
</tr>
<tr>
<td>9.5.3</td>
<td>Running the test</td>
<td>114</td>
</tr>
</tbody>
</table>
List of tables

Table 1 – Kernel level tests 43
Table 2 – Network level tests 43
Table 3 – Application level tests 43
Table 4 – Group level tests 44
Table 5 – Temporary results from file creation test 55
Table 5 – Bubble sort metrics table 64
Table 6 – Metrics produced for varied file test 66
Table 7 – Metrics produced for fixed size file test 66
Table 8 – Metrics table for Client-Server on local device 67
Table 9 – Metrics table for Remote Client-Server test 69
Table 10 – Average Remote download times by device 71
Table 11 - Metrics for Remote file download test. 71
Table 12 – Sample data from network monitor test 72
Table 13 – Metrics table for network monitor test 72
Table 14 – Data captured during the system utilisation test. 73
Table 15 – Metric for overall utilisation test 73
Table 16 – Sample data captured from a java threads test. 74
Table 17 – History table of Java Thread totals 74
Table 18 - Sample data captured by the group level test 75
Table 19 – Average battery strength used during the continuous tests 76
Table 20 – Example of total system metrics 77
Table 21 - Time taken by tests to depreciate the battery by 1% 79
## List of Figures

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Estimated, fixed, mobile and Internet Subscribers globally</td>
<td>10</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Ad-hoc wireless network example</td>
<td>12</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Proposed MARIAN framework</td>
<td>13</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Infrastructure wireless network architecture</td>
<td>18</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Ad-Hoc wireless network architecture</td>
<td>18</td>
</tr>
<tr>
<td>Figure 6</td>
<td>An example of combination of ad-hoc wireless with fixed networks</td>
<td>20</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Agent Taxonomy</td>
<td>22</td>
</tr>
<tr>
<td>Figure 8</td>
<td>The nine building blocks of the Wireless World.</td>
<td>30</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Distribution of devices in heterogeneous networks</td>
<td>31</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Ad-hoc routing protocols</td>
<td>32</td>
</tr>
<tr>
<td>Figure 11</td>
<td>An example of AODV routing protocol.</td>
<td>33</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Creation of a route record in DSR</td>
<td>34</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Example of the Java Native Interface</td>
<td>36</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Connecting to a database using JDBC</td>
<td>37</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Stages of Methodology</td>
<td>38</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Database access using three-tier model</td>
<td>45</td>
</tr>
<tr>
<td>Figure 17</td>
<td>High-level design of the Benchmark Software System</td>
<td>46</td>
</tr>
<tr>
<td>Figure 18</td>
<td>BSS design as part of MARIAN framework</td>
<td>46</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Implementing tests using JNI and native code</td>
<td>51</td>
</tr>
<tr>
<td>Figure 20</td>
<td>VB Application running on emulator and PDA</td>
<td>53</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Network Test Menu on the user Interface</td>
<td>54</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Sequence diagram of 4D bubble sort test run</td>
<td>58</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Example of utilising the database</td>
<td>60</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Bubble sort comparisons, per device</td>
<td>62</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Comparisons between devices running Bubble sort</td>
<td>63</td>
</tr>
<tr>
<td>Figure 26</td>
<td>Bubble sort: time to discharge 1% of battery</td>
<td>63</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Comparison between devices creating files of varied size.</td>
<td>64</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Comparison between devices creating files of fixed size.</td>
<td>64</td>
</tr>
<tr>
<td>Figure 29</td>
<td>Varied file size test: time to discharge 1% of battery</td>
<td>65</td>
</tr>
<tr>
<td>Figure 30</td>
<td>Varied number of fixed size file test: time to discharge 1% of battery</td>
<td>65</td>
</tr>
<tr>
<td>Figure 31</td>
<td>Client server test on local device, comparison between devices</td>
<td>66</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Client server test on local device: time to discharge 1% of battery</td>
<td>67</td>
</tr>
<tr>
<td>Figure 33</td>
<td>Remote Client-Server test, Comparison between devices</td>
<td>68</td>
</tr>
<tr>
<td>Figure 34</td>
<td>Remote Client-Server test: time to discharge 1% of battery</td>
<td>69</td>
</tr>
<tr>
<td>Figure 35</td>
<td>Remote file download, connection times</td>
<td>70</td>
</tr>
<tr>
<td>Figure 36</td>
<td>Remote file download, download times</td>
<td>70</td>
</tr>
<tr>
<td>Figure 37</td>
<td>Remote file download, total time taken</td>
<td>70</td>
</tr>
<tr>
<td>Figure 38</td>
<td>List of Templates for automated tests</td>
<td>110</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Introduction

As mobile computing is becoming a more viable option for business as well as home users there is an increasing need to re-develop the traditional network infrastructure to empower mobile devices to actively take part in the creation and maintenance of this new distributed infrastructure. The wireless network and in particular the ad-hoc wireless network is proving to be the emerging force behind the drive for a new distributed network infrastructure. (Hännikäinen, M., et al 2002) suggests that by the end of 2004, the numbers of users connected to the Internet by Wireless and fixed connections will converge, with a large proportion of the subscribers being personal data communications. Figure 1 shows the forecasted numbers. This is supported by the similar findings of (Makki, S.A.M., et al 2003). However they state that although the current number of business subscribers falls short of the personal sector, the business market is expected to be very large and that companies are preparing for it.

![Figure 1 – Estimated, fixed, mobile and Internet Subscribers globally](image)

The growth of the number of wireless enabled devices has seen the emergence of the wireless ad-hoc network, this network, due to the lack of infrastructure has to operate effectively within rapidly changing circumstances with dramatically increasing quantities of available information. To do this effectively the role of the mobile device has to change from a simple network device to one that is capable of routing data between nodes and in some cases between networks. This view is also taken by (Remondo, D., et al 2003) who propose the introduction of dedicated mobile routers in ad-hoc networks due to the constant movement of the nodes.

According to (Gerla, M., et al 2002) wireless networking and in particular ad-hoc networking is an emerging research field where the networks are no longer viewed as stand-alone groups of wireless terminals but are expected to become fundamental in the future development of infrastructure networks. The ad-hoc network is designed to be self-configuring with no central management system with configuration responsibilities, the responsibility for routing and network topology discovery is therefore allocated some if not all of the nodes in the network. (Migas, N., et al 2003) refer to this type of network as an “Infrastructure-less” network and propose a scheme where mobile nodes are grouped in clusters according to geographical criteria thus
dividing the network into sub networks with gateway nodes, which are the most powerful nodes in terms of processing power or memory, taking the responsibility for routing packet across the sub networks.

It is my opinion that this is among the most efficient method as it ensures that only the most powerful nodes take the burden of routing thus conserving the energy resources of the nodes with low battery power. This scheme also takes into consideration the heterogeneous nature of the network with the changing of gateway nodes as the topology of the network changes. This theory is reiterated by (Gerla, M., et al 2002) in their “Selective Intermediate Node Scheme” where the nodes are arranged in clusters with gateway nodes, the main difference being that, as a priority, nodes with low energy are not utilised to route traffic as opposed to the most powerful nodes becoming gateways.

(Migas, N., et al 2003) suggest that the use of mobile and static agents are a promising solution for distributed computing over open and heterogeneous networks, this is due to the ability of the mobile agent to roam the network gathering information from the static agent residing on each node. This view is also taken by (Tripathi, A.R et al 2000), who explains the functionality and purpose of static and mobile agents in network management and Internet-wide collaborative systems.

According to a number of authors (Gavalas, D., et al 2001, Puliafito, A., and Tomarchio, O., 2000, Lee, J.O. 2000, Papavasiliou, S., et al 2001), the use of mobile and static agents for decentralised network management is the answer to the scalability limitations of centralised models and the flexibility problems of static hierarchical frameworks. Although these authors tend to agree with the use of software agents for network management, they fail to address the need for an efficient routing paradigm. (Phanse, K.S., and DaSilva, L.A., 2003) agree that the mobile agent paradigm has shown some promise in carrying out some network management tasks. However they claim that their application for Quality of Service, QoS is not yet well enough understood.

This report investigates the use of static and mobile agents to determine the best route through ad-hoc networks. The idea is based on current research by (Migas, N., et al 2003a, 2003b). The research suggests the implementation of a static agent running in the background on each mobile device, monitoring available resources such as connection availability, processing power, battery life, memory usage and memory capacity. The static agent will calculate a routing metric based on the information gathered, making this metric available to mobile agents, which will independently roam the network gathering this information. The mobile agents will then, according to the performance of the mobile devices, build a best path routing table.
1.2 Background

The increasing popularity of ad-hoc wireless networks, fuelled by reduced costs and increased efficiency and power of wireless devices has seen the paradigm of mobile agent-based computing cross the boundaries of academia into the industrial domain and more recently into home computing. This increased popularity has opened up new opportunities to exploit the technology to create a more open and dynamic system with no network boundaries. The heterogeneous nature of the ad-hoc wireless network also poses new technological challenges as the devices within the network may be of different operating systems and have no common interface.

The principal of an ad-hoc wireless network is based around mobile devices continually moving from one network to another and in the process, participating in the routing of information within the network. In some cases, the devices, due to their physical location within the network may route information from one network to another. This is illustrated in Figure 2, where Node A in network N1 wishes to send email traffic out onto the Internet; as it has no direct access to the Internet it has to discover a route that is suitable for the type of information it wishes to send. This is made possible by routing the email through Node B, (which is a PDA able to route simple email traffic) on network N2, which in turn routes the email through Node C on network N3 and finally, as this node has access to the internet the traffic is passed on towards its final destination.

![Figure 2 – Ad-hoc wireless network example](image)

The type of traffic used in this example is an email, which has a small payload. Discovering the type of traffic a device is capable of routing is fundamental to the development of a dynamic ad-hoc network, as the traffic should be routed via the most suitable route i.e. a PDA would not be used to route real-time video. Current on-demand routing protocols do not take the capabilities of the device into consideration when attempting to route traffic through the network. This indiscriminate routing is an expensive process in terms of drain on the battery power on a mobile device. (Migas, N., et al 2003) propose to develop a routing protocol that will take into consideration the performance capabilities of each mobile device, thus saving battery resources. This is part of their research into producing an agent-based framework for routing, topology and automatic network configuration in ad-hoc networks.
1.3 Scope, Aims and Objectives of the project

The subject area of the project encompasses both static and mobile agent technologies and the interconnections that exist between them; although they are similar in the sense that they are autonomous pieces of software that run independently of their environment, they are very different in the way that they perform their functions. This report analyses the functionality of both types of agents with a view to utilising them to create a more dynamic network infrastructure.

The report investigates the use of agent-based technologies to discover the routing ability of each device on the network and proposes to develop a prototype benchmark system to act as a static agent within the MARIAN framework proposed by (Migas, N., et al 2003), See Figure 3.

![Figure 3 – Proposed MARIAN framework](image)

Various research projects are currently being carried out investigating current and future agent-based routing protocols to determine the best path for routing traffic through the ad-hoc wireless network. According to a number of authors (Hassanein, H., and Zhou, A., 2002, Gerla, M., et al 2002, Jan., R-H., et al 2002), the use of traditional routing mechanisms in ad-hoc wireless networks are inappropriate due to issues such as low bandwidth, dynamic topology and limited survivability of the connection.

The research carried out for this report proposes that the determination of the best route should be based on a combination of complex metrics gathered from the mobile device; these metrics include memory capacity, network performance, processing capabilities, battery power, cost etc. Mobile Agents will then use these metrics to determine what the device is capable of routing. Due to the constant changes in the wireless ad-hoc topology the metrics will be frequently changing to reflect the surroundings as well as the current state of the device.

The heterogeneity of the ad-hoc network makes the Java language a suitable candidate for programming the agent-based system, this is due to its platform independence and strong networking capabilities. (Bellavista, P., et al 2001) suggest the use of java for its portability, dynamic class loading and easy integration with the Web. However the use of java also poses various challenges as it runs in a sandbox environment where it has no direct access to system resources. (Illmann, T., et al 2000) highlight the problems imposed by a java-based solution; one proposed method is the use of the Java Native Interface (JNI) to access local resources. (Binder, W., et al 2003) suggest
the development of a resource-aware java-based kernel, (J-SEAL2) to overcome the java “sandbox”. This is not a suitable approach for this project, as it limits the portability and use of the software to environments specifically designed for this new kernel thus defeating the purpose of developing a system for heterogeneous wireless networks. These challenges, as outlined below are investigated through the various stages of this report:

- **Research.** Investigate the current Java based Mobile Agent platforms and technologies available to help determine the most suitable language and development environment for the MARIAN agent-based framework for ad-hoc wireless networks. The main software development aspects of the study concern the creation of static agents that reside on all hosts within the network; these static agents will be responsible for gathering information from the local host such as battery life, CPU capabilities, determining network latency and connectivity and also gathering information such as hostname, IP address, operating system, Java Virtual Machine version etc. The details will then be recorded in a local file or database, which will be made available to authenticated Mobile Agents in the form of a system metric for use in building up network topology and routing information. The use of the Java Native Interface (JNI) as well as other methods will be investigated as a means gathering system resources information. The research investigates current systems and future proposals for Agent Based Platforms paying particular attention to routing, topology discovery and automatic network configuration techniques. The study will propose how the static agent-based benchmark system will function within the MARIAN framework and how after further development could be utilised in current and future software applications.

- **Design.** Develop and design a static agent that will run predefined benchmark tests on various network devices. These tests will have to be designed to run on Personal Digital Assistants (PDA) as well as Laptops, Workstations and Servers. The programming language used must cater for the heterogeneity of the network devices and operating systems. This will allow for the reuse of the same source code for all platforms. The tests will have to be carefully designed to ensure that the information captured is relevant to the capabilities of each device. Design a database and also a test file structure to allow for automated running of these tests, based on input criteria. The system must be designed in such a way that it promotes reusability, security, scalability, portability, efficiency and maintainability.

- **Implementation.** Implement a prototype system able to act as static agent that allows for automatic running of tests on all types of devices mentioned in the design stage. The results of these tests will be stored in a results file and where applicable to a database. Create a database to hold all test and system configuration data, which will in future allow for the extraction of the final metrics calculated from the test results. The metrics produced from the results are to be made available to the mobile agents via a secure interface.
• **Testing.** This will involve running all tests and collating the results to establish a set of metrics for each device. The tests will be run to ensure that each performance capability of the device is tested, for example a complex algorithmic test will determine the processing power of the CPU whereas a file creation test will determine the speed of the hard drive of a PC or Server and the memory on a PDA, (as the PDA has no hard drive but only RAM:ROM). The can be run using a menu-based interface, or automated via a test run file, which will set the test criteria at runtime. The test file to run as well as the system mode of operation will be specified in a configuration file, which is automatically loaded at the system start-up.

• **Evaluation.** Critically evaluate all work carried out to identify the areas where further research and development are required to fully implement the proposed system. Outline the problems encountered throughout the research and how these were overcome. This section will also include an evaluation of the results obtained from running the tests, highlighting any patterns that have been observed. The findings from the testing and analysis stage will be discussed to identify where the tests and the methods for calculating the system metrics could be improved.
1.2 Introduction

The purpose of this section is to review current research in the field of agent-based wireless ad-hoc networking, critically evaluating all evidence presented. According to (Migas, N., et al 2003b) there is currently no routing protocol that allows for the automatic network configuration in ad-hoc wireless networks, their proposed MARIAN framework addresses these issues using a combination of static and mobile agents. The research will investigate the current proposed solutions to determining the best path to route traffic through an ad-hoc wireless network, the aim being to determine the most suitable methodology for the development of a static agent to operate as part of the overall MARIAN framework.

This section investigates a number of key aspects and research questions aimed at agent-based wireless networks; these are outlined as follows:

**Wireless Networks:** The architecture and application of wireless networking

**Software Agents:** What are agents, why use them and for what purpose?

**The future:** Do agent-based technologies have a future?

**Agent platforms:** What agent development platforms are available and what standards are there for their development?

**Routing:** What current routing protocols are available for ad-hoc networks and investigate their shortcomings?

**Security:** What security issues arise from the use of ad-hoc networks and how can they be overcome?

**Local resources:** How can agent-based software access system resources?

1.3 Wireless Networks


2.2.1 Infrastructure

Wireless networks are often used to extend, rather than replace traditional wired networks and are referred to as infrastructure networks, (Agrawal, P., et al 2001). An example of an infrastructure network is shown in figure 4. The infrastructure network is based on a hierarchy of Wide are networks, WANs and local area networks, LANs; these networks are used as the backbone of the network and provide a connection to
the wireless network. The wireless network connects to this backbone via a wireless base station. The base stations are usually situated in fixed locations designed to facilitate maximum transmission rate for the mobile nodes; they are responsible for coordinating the access to the wired network for one or more mobile wireless devices. Many organisations have adopted this approach where the costs of implementing a complete wired network have been far greater than a combination of wired and wireless. Infrastructure wireless networks have various advantages over the infrastructure-less networks such as reliability, connectivity, security, predictability and increased bandwidth. These advantages are made possible due to the use of base stations to connect to existing wired or wireless networks. However the infrastructure-based wireless network is rigid and lacks the mobility of the infrastructure-less network as the nodes in the infrastructure network must communicate via a base station, which if installed in an incorrect manner could greatly reduce the area covered by the infrastructure wireless network.

2.2.2 Infrastructure-less

Infrastructure-less wireless networks are self-configuring, self-organising and self-administering (Gerla, M., et al 2002), they are also referred to as ad-hoc networks. According to (Chlmtac, I., et al 2003) mobile ad-hoc networks (MANETs) represent complex distributed systems that comprise wireless mobile nodes that can freely and dynamically self-organise into arbitrary and temporary, “ad-hoc” network topologies, allowing people and devices to seamlessly internetwork in areas with no pre-existing communications infrastructure.

An ad-hoc device is by definition a source of information, a drain for information and a router for information flows from other devices in the network. A wireless ad-hoc network allows unrestricted mobility of the mobile devices, as long as at least one device is within transmission range. Direct neighbours can be used to route information to nodes outside the transmission range of a wireless node. The transmission range of a node is restricted by the limited battery power of the device, thus the transmission range is relatively small in comparison to the potential overall size of the ad-hoc network. Mobile routers could also act as gateways to other wireless ad-hoc networks. Figure 5 shows a typical ad-hoc network structure.

One of the main advantages of the ad-hoc network is, that it does not require any kind of infrastructure, like a base station in infrastructure networks. Thus ad-hoc networks are best suited to an environment unable to provide any infrastructure. (Fitz, F., et al 2002) define an ad-hoc network as networks formed by users or devices wishing to communicate, without the necessity for the help or existence of any infrastructure or previously established relationship between the potential network members. This definition is supported by (Jones, C.E., et al 2001) who state that ad-hoc networks are characterised by dynamic, unpredictable, random, multi-hop topologies with typically no infrastructure support.

Nodes in the ad-hoc network can vary in terms of characteristics such as transmission power, processing capabilities, memory, size, cost etc.
Figure 4 – Infrastructure wireless network architecture

Figure 5 – Ad-Hoc wireless network architecture
Many writers on WLANs tend to argue for one networking technology over the other, that is, wireless versus wired, as though you need to choose one over the other (Wilfred, B., 2000). As discussed earlier, the wireless network has to connect, at some stage to a wired network to complete the communication with the target host, if that host resides on a network that is only reachable via a wired network. So in most cases a wired network is a necessity whereas a wireless network is a supplement to its wired counterpart, enabling the expansion of the wired network. (Wilfred, B., 2000) describes the key functional distinction between the two types as follows: wired networks are much faster than their wireless cousins. Most current wired networks operate at 100Mbs, using switching technology. The switch in a wired network allows for the concurrent connection of various nodes, rapidly switching the signal from port to port, thus in theory allowing multiple nodes to each have the full data rate of 100Mbs. The notes in an infrastructure wireless network on the other hand share the available bandwidth, which in most cases is 11Mbs. For this reason the wireless network is not suitable for applications that require high bandwidth such as real-time video capture and Computer Aided Design (CAD) where large files are transferred over the network.

One of the major advantages the wireless network has over the wired is the mobility and flexibility it provides to the computer user, especially in the case of the ad-hoc wireless network where self-configurability and independence of existing infrastructures are key issues. Wireless networks have the potential to provide wireless and mobile computing capability in situations where efficient, economical and rapid deployment of communication is required, and where the use of a wired network is either too expensive or impractical (Phanse, K.S., et al 2003).

(Makki, S.A.M., et al 2003) states that a wireless network exhibits the following advantages:

- Mobility,
- Ease of installation in difficult-to-wire areas,
- Reduced installation time,
- Increased reliability,
- Long-term cost savings.

These advantages, in many cases outweigh the disadvantages. However due to the nature of wireless networks, we should be aware of the following disadvantages of implementing a wireless network as described by (Makki, S.A.M., et al 2003).

- Radio signal interference,
- Power management,
- System inoperability,
- Network Security,
- Installation issues,
- Health risks.
2.2.4 Interoperability with fixed networks

The implementation of ad-hoc wireless networks with fixed networks is very attractive, because it allows usage of a wider range of services (Fitz, F., et al 2002). The connection between the networks can be easily achieved by connecting one node that is participating in the ad-hoc network directly to the fixed network, the node on the fixed network could also be a wireless node that is connected to the fixed network via a wireless base station. The node in the ad-hoc network would advertise the fact to the nodes in the ad-hoc wireless network that it is acting as a router, routing traffic to and from the ad-hoc network to the fixed network. The research being carried out by (Migas, N et al 2003) would identify this mobile node by use of the agent-based MARIAN routing protocol. This would ensure that the node selected for the routing would have sufficient performance capabilities to perform the routing efficiently.

There are still various issues relating to the use of ad-hoc networks for extending the fixed network, such as, authentication of mobile devices, limited capabilities and resources of mobile nodes, signal interference etc. An example of ad-hoc networks combined with fixed networks is given in figure 6.

Wireless nodes are distributed over a given area, some of which connect directly to a wired access point. Because of the missing infrastructure, not every access point can be connected hard-wired. Therefore virtual access points are introduced. Virtual access points are connected directly or over multi-hop with the wired access points. Wireless nodes can connect to any access point, for example the nodes between buildings b and c will use other nodes, (in an ad-hoc manner) as routers for multi-hopping based access to “access points”.

![Figure 6 – An example of combination of ad-hoc wireless with fixed networks](image-url)

2.2.5 Problems with wireless networks

Wireless networks involve self-configurability, interoperability and new networking concepts such as mobile routers. Routing becomes a major problem, due to these issues as well as the heterogeneous nature of the ad hoc wireless network. Factors
such as energy availability, terminal mobility, terminal processing capabilities increase the complexity of the routing. This creates new challenges in the networking layer (Remondo, D., and Niemegeers, I.G., 2003). The networking problems are also highlighted by Xu (2002), who points out that although the IEEE 802.11 medium access control (MAC) protocol is a standard used for wires LAN’s, and is used in wireless mobile multi-hop networks, it is not designed for multi-hop networks (Xu, S., and Saadawi, T., 2002).

Another major challenge as described by (Phanse, K.S., and DaSilva, L.A., 2003) is the level of Quality of Service (QoS), which is reduced in areas such as radio signal strength and availability. A number of various challenges are (Phanse, K.S., and DaSilva, L.A., 2003, Qi, H and Wang, F., 2001, Luck, M., et al 2003):

- **Low Bandwidth**
  - Wireless links are typically more bandwidth-constrained than in wired networks. The bandwidth can also be affected by fading, jamming, interference etc.

- **Dynamic Topology**
  - The dynamic nature of the wireless network causes limited link or node survivability, this coupled with new nodes being added adds to the complexity of the required routing protocol.

- **Limited resources**
  - Nodes in the ad hoc network tend to have limited resources such as battery life, storage and memory capacity and also limited processing power. These factors are detrimental to the routing capability of the nodes.

- **Limited Survivability**
  - Due to the deployment of ad hoc networks in diverse and often hostile environments, causes problems with node survivability due to high error transmission rates, this also increases the potential for security attacks.

This report, in coordination with the research being carried out by (Migas, N., et al 2003) recognises the problems with routing in wireless networks and as discussed earlier proposes the novel routing protocol MARIAN to help overcome them.

### 2.3 Software Agents

A Software agent is a piece of software that acts to accomplish tasks on behalf of the user (Jennings, N.R., et al 1998). There is much discussion about whether some particular system is an agent, an intelligent agent or merely a program. This is a general problem in AI of defining “intelligence” that had led to much fruitless discussion. This has resulted in as many definitions as there are research papers. However there are certain broad characteristics that most have some measure of general agreement. One of the key features is autonomy; this is the ability of the agent to formulate its own goals and to act in order to satisfy them. Nwana defines an agent in terms of three behavioural attributes, any two of which must be possessed by a software agent; these are: autonomy, co-operation and learning (Nwana, H.S., 1996). Nwana’s requirements for agenthood are shown as a Venn diagram. (Fig. 7)
According to Maamar a software agent exhibits a number of features that make it different from other traditional components (Maamar, Z., 2002). These features are: autonomous, goal-orientated, collaborative, flexible, self-starting, temporal continuity, character, communicative, adaptive and mobile. Certain features are explained below:

- **Autonomous**: an agent is able to take initiatives and exercise a non-trivial degree of control over its own actions.
- **Goal-orientated**: an agent accepts high level requests indicating what a human wants and is responsible for deciding how and where to satisfy the requests.
- **Collaborative**: an agent does not blindly obey commands, but has the ability to modify requests, ask for clarifications, or even refuse to satisfy certain requests.
- **Flexible**: the agent’s actions are not scripted; it is able to dynamically chose which actions to invoke and in what sequence.
- **Adaptive**: the agent automatically customises itself to the preferences of its user based on previous experiences.
- **Mobile**: an agent is able to transport itself from one machine to another across different computing platforms.

### 2.3.1 Mobile and Static Agents

Agent-based programming can be split into two distinctive groups, Static Agents and Mobile Agents. Mobile agents are software programs than run on behalf of a user in a network and are capable of migration from one node to another; they do this by suspending there execution on the current host, transferring to the destination host then continuing execution on the new host. (Ilmman, T., et al 2000) define the migration process as being Strong or Weak, strong migration being when the mobile agent can migrate at any point in its’ execution taking with it the code and complete state of execution and weak being any other migration that is not strong. The main problem with strong migration is that the Java language has limited access to intern
and native information required to capture the full execution state of the agent i.e. program counter, local stack frames and open resources of the current thread. (Illmann, T., et al 2000) get round this by extending the functionality of the Java Virtual Machine, (JVM) with a Java native Interface, (JNI) plug in. This JNI plug in allows for access to the intern and native information required for strong migration.

Static Agents are far less complicated in their structure and behaviour as they do not migrate from node to node. Each node in the proposed model will continuously run static agent which will be responsible for the monitoring and calculation of the node’s performance capabilities. As illustrated by (Migas, N, W. J., et al 2003), the static agent may act as an interface between routing agents and the local database holding the performance information, and provides security against malicious agents.

(Luck, M., et al 2003, p.9) define agent-based systems as:

...One of the most vibrant and important areas of research and development to have emerged in information technology in the 1990’s. Put at its simplest, an agent is a computer system that is capable of flexible autonomous action in a dynamic, unpredictable, typically multi-agent domain. Many observers believe that agents represent the most important new paradigm for software development since object orientation.

2.3.2 Evolution of Mobile Agents

Gray (1996) describes mobile agents as an extension of the remote procedure call and remote programming paradigms (Gray, R., et al 1996). Remote Procedure Call (RPC), allows a client to invoke a server operation using the standard procedure call mechanism. Remote programming allows a client to send a subprogram to a server. The subprogram executes on the server and sends its result back to the client. Mobile agents generalise the remote programming paradigm to allow arbitrary code movement.

Conventional distributed systems typically assume a static configuration of the environment where the distributed application executes. Communication among a set of hosts is enabled by physical links whose configuration is fixed and statically determined (Picco, G.P. 2001). This traditional fixed network is being challenged by technical developments that demand a degree of mobility in the distributed system. Gavalas (2001) proposes the use of Mobile agent technology for Network and Systems Management (N&SM) as an answer to the scalability limitations of centralised models and the flexibility problems of static hierarchical frameworks (Gavalas, D., et al 2001). Puliafito (2000) suggests the use of Mobile Agent systems to provide advanced network management functionalities, due to the possibility to easily implement a decentralised and active monitoring of the system (Puliafito, A., and Tomarchio, O. 2000). They argue that although the centralised paradigm adopted by the Simple Network Management Protocol (SMTP) is appropriate in several network management applications, it fails to provide the flexibility and scalability required by the growing complexity of computer networks.
Advantages of Mobile Agents

According to (Picco, G.P., 2001) by and large mobile agents, and more generally code mobility, are acknowledged to bring two main advantages over conventional, client-server based technologies; these are:

- **Enhanced flexibility**
  - Mobile agents can be used to update dynamically the interface on the client and/or server. For instance, this technique can be used by an application to change dynamically the interface to a Database Management System (DBMS), allowing a client to interact with it without requiring an Open Database Connectivity Driver (ODBC).

- **Reduced Bandwidth consumption**
  - The ability to migrate a client program to achieve co-location with resources it must access on the server reduces the need for remote communication and thus may enable, under some conditions, a more efficient use of the communication link.

The mobile agent paradigm may offer a several other advantages compared to the traditional client-server approach. (Picco, G.P., 2001) describes the following as being a major advantage:

- **Improved fault tolerance**
  - In conventional systems, a high-level interaction between client and server, e.g. a commercial transaction, unfolds as a series of low-level interactions. During these interactions, the state of overall computation is distributed. This fact heavily complicates the task of recovering from a fault, due to the distributed consensus problem. Instead, agents embedding the code describing the whole high-level interaction can migrate on the server. Thus, the state of the interaction remains entirely local, and faults can be dealt with easily.

- **Support for disconnected operations**
  - Mobile agents can carry out their tasks autonomously and independently of the application that dispatched them.

- **Protocol encapsulation**
  - In conventional systems, data is typically a passive element that gets processed by other active components in the system. Mobile agents and code mobility may change this view dramatically, by allowing a piece of data to travel within the system together with the application logic needed to interpret and manipulate it. Thus, a packet could flow in a network carrying along with it its own routing routines, as proposed in the MARIAN framework (Migas, N., et al 2003b).

Disadvantages of the mobile agent paradigm

According to Luck 2003, one of the most fundamental obstacles to the take up of agent technology is the lack of mature software development methodologies for
agent-based systems. Clearly, basic principles of software and knowledge engineering need to be applied to the development and deployment of multi-agent systems, but they also need to be augmented to suit the differing demands of this new paradigm (Luck et al 2003). The lack of standards being applied to the development of agent-based systems as well problems such as Security and Interoperability continue to be the main obstacles to the wide take up of this technology (Bellavista, P., 2000). This section of the report outlines these problems, suggesting what strategies the industry needs to adopt to develop the agent-based paradigm.

2.3.4.1 Interoperability

An important goal in mobile agent technology is interoperability between various manufacturers’ agent systems. Interoperability becomes more achievable if actions such as agent transfer, class transfer, and agent management are standardized. When the source and destination agent systems are similar, standardization of these actions can result in interoperability. However, when the two agent systems are dramatically different, only minimal interoperability can be achieved.

Interoperability is not about language interoperability. Mobile Agent System Interoperability Facilities (also called MAF, an acronym for the original proposal, Mobile Agent Facility) is about interoperability between agent systems written in the same language, but potentially by different vendors and systems that are expected to go through many revisions within the life-time of an agent. Language interoperability for active objects that carry “continuations” around is technically difficult to achieve. Furthermore, it is not needed, since the support for different languages can be replicated at each node. There are various bodies currently investigating the lack of implementation of these interoperability issues with an aim to standardise the agent-based software development. Examples of these standards and their aims are discussed below.

2.3.4.2 Standards

Much of the standardisation effort in the agent community has been taken up by the Foundation for Intelligent Physical Agents (FIPA) and also the Object Management Group (OMG). The main goal of FIPA is to facilitate the interoperation and internetworking between agents across multiple, heterogeneous agent systems. To achieve this goal, FIPA have developed a set of specifications that range from agent platform architectures to support communicating agents, semantic communication languages and context languages for expressing messages and interaction protocols that expand the scope from single messages to complete transactions.

The OMG is moving forward in establishing the Model Driven Architecture™ as the "Architecture of Choice for a Connected World"™ through its worldwide standard specifications including CORBA®, CORBA/IIOP™, the UML™, XMI™, MOF™, Object Services, Internet Facilities and Domain Interface specifications. The CORBA standard provides a distributed computing environment (DCE) where distributed objects can transparently interact according to the client-server model (Bellavista, P., et al 2001). CORBA allows the integration of already implemented software components by wrapping them in an Interface Definition Language (IDL) that describes their behaviour. This enables compliance between CORBA and mobile agents despite their differences.
The OMG has defined the Mobile Agent System Interoperability Facilities (MASIF) standards, also known as MAF. This standard, The Mobile Agent Facility (MAF) is built within the CORBA framework, MAF includes a collection of definitions and interfaces that provide an interoperable interface for mobile agent systems. MAF is as simple and generic as possible to allow for future advances in mobile agent systems.

The main benefits of MAF are:
- enables interoperability between agent platforms of different vendors
- considers demands & capabilities of existing agent products
- integrates RPC paradigm and MA technology
- benefits from already existing CORBA standards

2.3.4.3 Security of Ad-Hoc Wireless Networks

One of the major challenges in the design of ad-hoc networks is their vulnerability to security attacks. (Haas & Zhou 1999) sets out detailed challenges and opportunities posed by the emergence of ad-hoc networks. The main problem with the security of these networks is that by their very nature they are inherently vulnerable to security attacks.

The emergence of Java-based Mobile Agent Platforms has greatly enhanced the security features of ad-hoc networks; this is partly due to the security mechanisms and policies provided as basic facilities of the Java language as well as the additional security features implemented as part of the Mobile Agent Platforms. (Fischmeister et al., 2001) outlines the security flaws still inherent in these systems as in the case of the Aglets system where the Java Security Manager allows the security policy to be changed by an unauthorised agent resulting in a compromised security database, this is the same for the Jumping Beans system.

There are various other security flaws mentioned by (Fischmeister et al., 2001) ranging from the ability to shut down the agent based system and the JVM to corrupting the GUI interface. These results clearly show that there are serious security vulnerabilities not just in the Mobile Agent systems but also with the underlying JVM. The Agent Based systems as well as Java have improved greatly as emphasised by (Chatzipapadopoulos et al., 1999) with the introduction of X509 certificates, Secure Socket Layer communications, and symmetric key encryption plus various other security enhancements to the Grasshopper Mobile Agent system.

Assuming that the security shortfalls as mentioned by (Fischmeister et al., 2001) are as bad as they seem, I feel that the most important consideration is to get the security right first as this will help to build up trust in the system and subsequently lead to increased adoption of the technology by mainstream businesses. The use of a database as suggested for the project will introduce a further level of security as the static agent will act as an interface to this database which will require password authentication, effectively isolating the information from malicious agents.
2.4 Agent-Based Development platforms

All Agent platforms investigated were java-based, as all mobile agent systems have to be implemented using code that can execute on a variety of hardware (Chatzipapadopoulos et al., 1999). Furthermore, applications developed using Java-Based Mobile Agent Platforms have more expressive power and can deal more elaborately with a particular problem that the respective scripting language ones, (Chatzipapadopoulos et al., 1999).

Among the Mobile agent platforms investigated by (Puliafito et al., 1998) the \textit{MAP} platform, although it complies with some standards as defined in (Chatzipapadopoulos et al., 1999) it fails to implement efficient security measures. The Ajanta platform as discussed by (Ahmed et al., 2002) on the other hand implements a secure location independent naming system that creates a unique name for each Agent it creates to allow for detection of “unnamed” malicious Agents. However the Ajanta platform fails to comply with many standards as defined in (Chatzipapadopoulos et al., 1999), it also does not support Migration history which is crucial if you wish to be able to track the movements of the Mobile Agents throughout the network. (Chatzipapadopoulos et al., 1999) investigates five Mobile Agent systems covering all aspects of security, compliance to standards, messaging and migration techniques and persistence. Of the five Agent Systems, the Grasshopper system has the most functionality, among the best for security and is the only one that complies with the Mobile Agent System Interoperability Facility, (MASIF) standard. The research undertaken concludes that Grasshopper is the preferred Mobile Agent Platform for the development of an agent-based system developed using the java language.

2.4.1 Implementing the Mobile Agent Paradigm using Java

Java is currently the predominant programming language used for the development of mobile agent systems and platforms. Java has several features not found in any other languages that support implementation of mobile agents (Funfrockenm S., 1998). This includes features such as:

\textbf{Platform-independence}. Java is designed to operate in heterogeneous networks. To enable a Java application to execute anywhere on the network, the compiler generates architecture-neutral byte code, as opposed to non-portable native code. For this code to be executed on a given computer, the Java runtime system needs to be present. There are no platform-dependent aspects of the Java language. Primitive data types are rigorously specified and not dependent on the underlying processor or operating system. Even libraries are platform-independent parts of the system. For example, the window library provides a single interface for the GUI that is independent of the underlying operating system. It allows us to create a mobile agent without knowing the types of computers it is going to run on.

\textbf{Secure execution}. Java is intended for use on the Internet and intranets. The demand for security has influenced the design in several ways. For example, Java has a pointer model that eliminates the possibility of overwriting memory and corrupting data. Java simply does not allow illegal type casting or any pointer arithmetic. Programs are no longer able to forge access to private data in objects that they do not have access to.
This prevents most activities of viruses. Even if someone tampers with the byte code, the Java runtime system ensures that the code will not be able to violate the basic semantics of Java. The security architecture of Java makes it reasonably safe to host an untrusted agent, because it cannot tamper with the host or access private information.

**Dynamic class loading.** This mechanism allows the virtual machine to load and define classes at runtime. It provides a protective name space for each agent, thus allowing agents to execute independently and safely from each other. The class-loading mechanism in extensible and enables classes to be loaded via the network.

**Multithread programming.** Agents are by definition autonomous. That is, an agent executes independently of other agents residing within the same place. Allowing each agent to execute in its own lightweight process, also called a thread of execution, is a way of enabling agents to behave autonomously. Fortunately, Java not only allows multithread programming, but also supports a set of synchronization primitives that are built into the language. These primitives enable agent interaction.

**Object serialization.** A key feature of mobile agents is that they can be serialized and deserialized. Java conveniently provides a built-in serialization mechanism that can represent the state of an object in a serialized form sufficiently detailed for the object to be reconstructed later. The serialized form of the object must be able to identify the Java class from which the object's state was saved, and to restore the state in a new instance. Objects often refer to other objects. Those other objects must be stored and retrieved at the same time, to maintain the object structure. When an object is stored, all the objects in the graph that are reachable from that object are stored as well.

**Reflection.** Java code can discover information about the fields, methods, and constructors of loaded classes, and can use reflected fields, methods, and constructors to operate on their underlying counterparts in objects, all within the security restrictions. Reflection accommodates the need for agents to be smart about themselves and other agents.

**Java Database Connectivity (JDBC)**
This allows programmers to write a single database interface, enables database management system (DBMS) independent java application development tools and products, and allows database connectivity vendors to provide a variety of different connectivity solutions. However the implementation of JDBC drivers for devices running operating systems such as Windows CE is still not being carried out by many vendors. For example Sun Microsystems still have not implemented such a driver.

### 2.5 Examples of current and future Agent-Based technology
There is a great deal of interest and enthusiasm in the field of Mobile Computing; the widespread use of wireless devices is ever increasing and as the development of cheaper and more powerful devices continues, I believe there will be an increasing demand for robust, intelligent, dynamic, self configuring, distributed, small and secure software. This is where I see the role of the Mobile Agent paradigm may come
into play; but before this can happen there are many changes and challenges the Mobile Agent and Ad-Hoc networks face.

2.5.1 Current Applications

(Fitz, F., et al 2002) describe the use of ad-hoc wireless networks for military applications. On the battlefield, an ad-hoc network among troops could be used to maintain communications, track their position and also for monitoring the movements of enemy forces.

Network and Systems Management (N&SM) applications have been proposed by a number of authors (Gavalas, D., et al 2001, Puliafito, A., and Tomarchio, O., 2000, Lee, J.O. 2000, Papavassiliou, S., et al 2001). Puliafito (2000) points out that the centralized paradigm adopted by SNMP is appropriate in several network management applications, but the quick expansion of networks has posed the problem of its scalability, as well as for any other centralized model (Puliafito, A., et al 2000). Puliafito and Tomarchio investigated the use of an agent-based N&SM solution developed using the Mobile Agent Platform (MAP), a platform for the development of mobile agent systems. The authors ran network management functions using both mobile agent and SNMP approaches, the results showed that the mobile agent approach produced considerably less network traffic overhead. The main reason for this was due to the mobile agent system traffic only consisting of the code of the agent that migrates to the remote node at the beginning of the task. Once the agent has run on a specific node, the code will no longer need to be retransmitted to the network, thus improving performance.

Internet-wide collaborative systems, especially workflow systems, are one of the emerging application areas, as the Internet is becoming the primary computing environment (Tripathi, A.R., et al 2000). Mobile agents can provide the programming abstraction for such applications for several reasons. They can asynchronously execute coordination actions. They can also be implemented in disconnected environments as they can carry all the application specific data and code. The mobility of a shared object can be exploited by implementing it as a mobile agent. This could allow for the object to be moved from one participant to another at various stages of a workflow process with the data and code being updated at each stage of the workflow.

2.5.2 Future Applications

(Luck et al., 2003, p.36.) Outlines a three phase plan for the development of agent-based systems over the next decade:

Phase 1: (c. 2000-2002)
Most systems are being built in-house resulting in lack of industry standards with design approaches following an ad hoc approach resulting non-standard software that does not comply with any regulations.

Phase 2: (c. 2003 – 20005)
Systems will start to be integrated and less “closed”. Development teams will start to communicate and adopt standard agent communication languages, such as FIPA ACL, but interaction protocols will remain non-standard. The systems will be much
larger, will be developed using defined methodologies and will be able to handle very large numbers of Agents.

**Phase 3: (c.2006 – 2008)**

Multi-Agent systems will permit participation by heterogeneous agents, designed by different teams. Any agent will be able to participate in these systems provided their behaviour conforms to a defined set of standards.

I agree with point that (Luck et al., 2003, p.36) make as there is great potential for Mobile Agents over ad-hoc networks to dominate many areas of industrial and home computing, but I remain sceptical with regards to industry complying with a predefined set of communication language and methodologies.

The challenges highlighted by (Niemegeers & Remondo 2002) contradict that of (Luck et al., 2003, p.36.) as there is no mention of lack of standards or methodologies but instead the focus is on the need for self-organisation of complex ad-hoc networks, and the interaction between ad-hoc networks and infrastructure networks. I tend to agree with (Luck et al., 2003), I believe the challenges highlighted by (Niemegeers & Remondo 2002) are more of a short-term challenge rather than over the next decade.

Research carried out by (Werner, M 2002) suggests that the development of future wireless systems is significantly determined by general shaping factors such as society, politics, economy, environment, technology, customers and competition, which are effecting the regulatory environment, the user behaviour and potential future services and applications. (Werner, M 2002) claims the living spaces where future wireless systems will be operated are at home Personal Area Networks (PANs) by short-range connectivity systems, in public spaces Wide Area Networks (WANs), at work and in transportation systems. Werner (2002) defines the future of the wireless in terms of *The Nine building blocks of the Wireless World* (Werner, M. 2002). (See Figure 8).

![Figure 8 – The nine building blocks of the Wireless World.](image-url)

(Remondo, D., et al 2003) supports this research and suggests additional uses for the use of wireless networks and in particular ad-hoc wireless networks, in applications such as:
Krco (2002) suggests the advent of Bluetooth and other similar short-range wireless systems will increase the emergence of the Virtual Personal Distributed Networks (VPDNs) (Krco, S., et al 2002). The authors describe the VPDN as an extension to the PAN, with significant challenges to overcome before deployments become possible (Krco, S., et al 2002). VPDNs are extremely dynamic with continuously changing topology. Securing the data transmissions is a very important aspect of such a distributed environment where many unknown devices and networks have access to user data. This is due to the great variety of devices with different characteristics, capabilities and different wireless technologies that will be involved in making up the VPDN. The protocols for the VPDN will have to take into account all the specifics of each device and each topology. Figure 9 describes the VPDN as depicted by (Krco, S., et al 2002).

![Figure 9 – Distribution of devices in heterogeneous networks](image)

2.6 Routing Protocols

Routing protocols are responsible for determining the best route that the data packets should take. These routing protocols allow routers (mobile nodes) to intercommunicate with other routers so that they can determine the structure of the interconnected networks (Buchanan, W. J., 2000). In ad-hoc networks a direct communication between any two nodes is possible subject to adequate radio propagation conditions and transmission power limitations of the nodes. If there is no direct link between the source and destination nodes, multi-hop routing is used. If there is no infrastructure, each node, beside transmitting and receiving its own packets, takes an active part in establishing and maintaining routes for other nodes (Krco, S., et al 2002). Routing in ad-hoc networks is a very challenging task due to several characteristics of the ad-hoc network. Nodes are mobile, can join and leave
the network at any time thus changing the topology of the network continuously, making routing tables obsolete. Since there is no centralised control keeping the routing tables up-to-date, a distributed algorithm is required. There are two categories of routing protocols: reactive (on-demand) and proactive (table-driven), see Figure 10.

![Ad-hoc routing protocols](image)

**Destination Sequenced Distance Vector (DSDV)** is a proactive distance vector protocol, which maintains a route to all nodes within the network, including those to which no packets are sent. Broadcasted route updates contain the address of the destination, the number of necessary hops to reach the destination, the sequence number of the route information regarding the destination, and a new sequence number unique to that broadcast. Receivers of this route updates can thus update their own routing tables, and if necessary broadcast their own routing tables. Johanasson, P., et al 1999 carried out tests, comparing DSDV with DSR and AODV routing protocols; the conclusions put forward by their research suggested that DSDV was not suitable for wireless ad-hoc networks, due to the fact that it struggles to keep the routing tables up to date in the rapidly changing network. This results in lost packets and an inability to handle the increased traffic throughput as the size of the network increases (Johanasson, P., et al 1999). Further evidence for the unsuitability of this routing protocol is put forward by Aaron (2001) who states that the overheads associated with the DSDV protocol carrying out periodic updates, which includes the whole routing table increased as $O(n^2)$ as the network grows, effectively limiting the number of nodes in the network (Aaron, A., and Weng, J., 2001).

**Ad-Hoc On-Demand Distance Vector (AODV)**

AODV is an example of a reactive routing protocol. This protocol does not continuously maintain the network topology. The topology is maintained only for those routes that are in use. When a route is no longer in use, it is removed from the routing tables. The network is then flooded with a “route request” message when a new route is required. When a “route request” message is received by the destination or a node, which has a route to the destination, a “route reply” message is generated and unicast back to the source node. Various simulations have shown that on-demand protocols perform better in ad-hoc networks than table-driven protocols (Broch, J., et al 1998). The main problem however with the proactive approach in ad-hoc networks stems from the fact that the topology of ad-hoc networks is changing continuously. Hence a dissemination of topology information is required which causes a large routing overhead. Also, depending on the traffic pattern on an ad-hoc network, it is possible that only a small fraction of the routes are actually used (Krc0, S., et al 2002). Figure 11 shows an example of AODV routing protocol.
Dynamic Source Routing (DSR)

DSR is also an example of a reactive on-demand routing protocol, which is based on the concept of source routing. DSR is similar to AODV, the main difference is that AODV stores only the shortest path for each destination, while DSR keeps multiple routes to a destination using aggressive route replies. In DSR, as in all source routing algorithms the sender of a packet determines the complete sequence of nodes through which to forward the packet (see Figure 12). The sender puts the list of the route into the header of the packet to identify each node of the hop sequence. In DSR there are no periodic route advertisements, instead when a new route is needed from a host to another, it dynamically determines on cached information and on the result of a route discovery protocol a route to the destination. Since DSR doesn't use any periodic update messages, using it can reduce the network bandwidth overhead especially during periods when there is no significant host movement in the network. By using DSR some battery power can be conserved on the mobile hosts, the mobile devices can put themselves into sleep or power saving mode. In addition, conventional routing protocols may compute some routes which do not work. A radio wave channel might be asymmetric, due to differing propagation or interference patterns. The primary disadvantages of DSR are two-fold. DSR is not scalable to large networks. The Internet Draft acknowledges that the protocol assumes that the diameter of the network is no greater than 10 hops. Additionally, DSR requires significantly more processing resources than most other protocols. In order to obtain routing information, each node must spend much more time processing any control data it receives, even if it is not the recipient (Aaron, A., et al 2001).
Ad-hoc On Demand routing protocols in mobile ad-hoc networks are of great interest to the research community with one of the main goals being to implement the most efficient routing protocol. (Huang & Marshall, 2003, p.1) are of the opinion that their new route request flooding control scheme which uses pre-location oriented routing to limit its route discovery query within a certain radius, is the “most efficient” routing protocol whereas the vast majority of research tends to lean more towards “Energy Efficient Routing Protocols” this is illustrated in the case of (Ebert et al., 2003) who put forward the claim that transmission power control in wireless packet networks with differing number of hops between the source and destination nodes achieves a substantial energy saving. As (Ebert, J.P., et al 2003) states it is clear that utilising a power control protocol can make substantial energy savings. However as the number of nodes in the clusters decreases the throughput decreases, but the dependence on the gateway node increases. (Brown et al., 2003, Agrawal, P., et al 2001, and Ebert, W., et al 2003) tend to agree on the general principal that energy efficient protocols contribute to a significant saving on battery life. (Agrawal et al., 2001) points out that one of the reasons why so much research is focused on energy aware protocols is that most of the energy savings that can be made at the physical network layer have been achieved therefore the focus must be on the higher layers of the protocol stack. As discussed above the energy consumption of the device plays a major part in the ability of it to route traffic through the network.

Research carried out by Jan, R-H et al (2002) proposes an efficient routing protocol called multiple next hops (MNH) which is based on the ADOV protocol for ad-hoc networks. The proposed MNH protocol establishes multiple paths for a route discovery procedure, this conserving network bandwidth and reducing route reconstruction times when routing paths fail (Jan, R-H., et al 2002).

Although there are many routing protocols available for routing in wireless ad-hoc networks, as well as much research into the improvement of existing and creation of new routing protocols, these improvements and new protocols are focused mainly on improving the efficiency of network, they do not take into consideration the type of traffic that is being routed.
The project will investigate as part of the MARIAN framework as proposed by Migas et al. (2003), the degree of suitability for agent-based technology in routing, topology discovery, and automatic network configuration. This research aims to fill the gap in current research by gathering system metrics that will allow for the creation of a routing protocol that will take into consideration the resources and capabilities of each node on the network. The hypothesis is to only route traffic to nodes that are capable of transmitting the traffic, thus saving valuable battery life of low power nodes by only transmitting to nodes that will be capable of routing the traffic. The use of agent-based technology will allow for the collection of this routing metrics without incurring large network traffic overheads, as the mobile agent will only collect a final metric that will have already been created by the static agent residing on each node.

2.7 Benchmarking software

The need for Quality of Service (QoS) provisioning and resource management in ad-hoc networks is recognised by number of authors (Lee, J.O., 2000, Bellavista, P., et al 2003, Puliafito, A., et al 2000, Papavassiliou, S., et al 2001, Phanse, S., et al 2002). The authors concentrate using mobile agents to gather simple high-level network management information on the health of the network or perform tasks such as allocation IP addresses. Bellavista, et al (2003) proposes the implementation of an agent-based on-line distributed monitoring system at different levels of abstraction (MAPI). At application level, it instruments the Java Virtual Machine to notify several different types of high-level events triggered during the execution of Java applications. At the Kernel level the MAPI system inspects low-level system-specific information via the Java Native Interface (JNI) that is usually hidden from java. The main purpose of the MAPI system is for distributed monitoring and control, although it collects system specific information such as CPU usage and network state it is not concerned with the capabilities of each node in respect to routing.

Although, as mentioned above, there are many research projects on network management and as described in section 2.6, there are various implementations of ad-hoc routing protocols, there is still a lack of research into benchmarking systems to retrieve performance metrics at Application, Network and Kernel level for use in the calculation of routing in ad-hoc networks.

The main software development objective of this project is therefore to develop such a Benchmark Software System (BSS) that will reside on all nodes in the ad-hoc and infrastructure-based wireless network as well as any wired network that the wireless networks connect to. The BSS will gather performance metrics at application level, Network level and Kernel from each node, these metrics will then be made available to mobile agents to aid in the creation of a routing protocol. The routing protocol will base decisions for best routes on Quality of Service requirements and traffic type.

2.8 Access to system resources

Based on the investigations carried out it is clear that Java is most suited as the development language to be used for the project due to the fact that the Mobile Agent systems are created in 100% java compliant code. This is an advantage when it comes to Wireless Programming as Java has various tools for developing wireless
applications (Funfrockenm S., 1998). The problem arises when we need to retrieve performance information from the host, as the Java Virtual Machine does not have direct access to the system resources; these have to be accessed via native code i.e. C/C++. (Bellavista, P., et al 2001) reports that this is indeed the case; they have suggested the use of the Java Native Interface (JNI) to integrate java programs with platform-dependant native code in their research to utilise the JVM for the online monitoring of local resources.

This has created an additional layer of complexity for the monitor and capture of local resources as none of the Mobile Agent Platforms have this functionality built in. The Java language has created the JNI, (Sun Microsystems Inc: Lesson: Overview of the JNI.) to allow Java to interact with native code, (C/C++) to retrieve system performance information.

### 2.8.1 Java Native Interface (JNI)

The JNI is a powerful feature of the Java platform. As mentioned in section 2.8, applications that use the JNI can incorporate native code. The JNI allows programmers to take advantage of the power of the Java platform without having to abandon their investments in legacy code. Because the JNI is part of the Java platform, programmers can address interoperability issues once, and expect their solution to work with all implementations of the Java platform (Laing, S, 1999). This proves to be the case when the JNI is used to access resources that are available to all Java implementations, however the benefits of the platform independence of the Java language are lost when the JNI is used to access system-level resources that call native code that is not common to all operating systems. This problem has been partially solved by Bellavista (2000) who use the java.system API to obtain the operating system type and subsequently call the relevant native code for that operating system using JNI (Bellavista, P., et al 2001).

The report by (Bellavista, P., et al 2001) details how to use the JNI for many operating systems, however their research does not cover Windows® CE. Further investigation will be required to clarify the suitability of using JNI to access system-specific resources for this operating system. Figure 13 shows how the JNI interacts with native C code.

![Figure 13 – Example of the Java Native Interface](image-url)
2.8.2 Java Open Database Connectivity (JDBC)

As a major part of the project was to implement a database to store test and system information, an investigation into the most suitable method to storage and retrieval was required. Sun Microsystems Inc. (JDBCTM API Documentation), suggest the use of the JDBC to access a database as this allows you to access virtually any data source, from relational databases to spreadsheets and flat file, using the industry standard Structured Query Language (SQL). As the project involves development of a system that can be implemented on various devices from a PDA to a Server, the version of the JDBC is restricted to the one that in theory will work all devices including the PDA. Figure 14 shows how the JDBC provides a connection to a database.

![Diagram of JDBC connection](image)

In simplest terms, a JDBC technology-based driver ("JDBC driver") makes it possible to do three things:

- Establish a connection with a data source
- Send queries and update statements to the data source
- Process the results

2.9 Conclusion

The main purpose of this literature review was to investigate the attributes of the mobile agent-based paradigm and its suitability for the future of ad-hoc on demand networks. The research has highlighted areas where the software development industry is failing to keep up with the technological advancement in other areas of the field. Further challenges have also been identified with regards to the accessibility to system resources on Windows CE® via the Java Native Interface due to the fact that the underlying operating system does not support the same system Application Programming Interface (API) calls as all other Windows® operating systems. The next section details the methodology used to produce the software for the project. The findings of this section will be used as a basis for determining the best approach to implementing the proposed system, taking into consideration that the project aims to produce a product that will contribute to the research by Migas, N., et al (2003).
3.1 Introduction
The project is part of ongoing research being carried out by Nikos Migas and Dr. William Buchanan, both of Napier University, Edinburgh. The purpose of the overall research project, being lead by the aforementioned researchers, is to investigate the degree of suitability of agent-based technology in routing, topology discovery, and automatic network configuration in ad-hoc wireless networks.

This section covers the requirements, analysis and design stages of the development of a Benchmark Software System (BSS) based on static agents that will run in the background on each mobile and static device, collecting resource information to be used by the mobile agents to produce routing information. As the project is part of a larger research project, the requirements, analysis and design stages will include references to the overall proposed framework. Figure 15 outlines the stages of the methodology.

3.2 Requirements
The development of the BSS, as described in section 3.1 has to fit into the overall MARIAN framework as proposed by Migas, N et al (2003b). The proposed framework can be logically divided into the following three layers: Foundation, Intermediate, and Core layer (Figure 3). The foundation layer is the physical layer and consists of all mobile nodes in an ad-hoc network. The intermediate layer sits on top of the foundation layer and is divided into two categories: the static agent model, and the mobile agent model. The top layer is the Core layer and is a combination of the static and mobile agent model. The BSS will reside on each mobile or static node in the foundation layer. The requirements for the BSS are therefore based on the technologies used by Migas, N., et al (2003b) in the proposed MARIAN framework; these requirements can be separated into software and hardware requirements for the development of the BSS and system requirements for the output required by the system.
3.2.1 Software Requirements
The BSS system must be able to communicate with mobile and static agents. The software must also run on many different types of operating system and hardware platforms. As the software is to participate within the overall MARIAN framework it must be either developed using Java or a compatible interpreted programming language. The system must be multi-threaded, have the ability to communicate with mobile agents and must also be secure. The BSS after running system tests will have the ability to store the results to file or permanent data storage. There is therefore a requirement to develop a database or file structure that will allow the insertion and retrieval of test results and final system metrics.

3.2.2 Hardware Requirements
The BSS system will have to run on various types of hardware; however as the system will run on top of a Virtual Machine, specific hardware requirements are not required. For the purposes of this project the system will be able to be run and be tested on the following hardware types:

- Windows® 9x, 2000, XP client and server workstation systems; these will have either a wireless network card or be connected to a wireless network via a wireless base station.
- Windows® 9x, 2000, XP Laptops with a wireless network card.
- UnixWare, Linx.
- HP iPaq Pocket PC Personal Digital Assistant (PDA) with wireless LAN card.
- All wireless cards will meet the IEEE 802.11b wireless standards.

3.2.3 System Requirements
The main objective of the BSS as described in section 3.1 is to run tests on the local system to gather system performance information. Performance tests are to be scheduled to run repeatedly in a periodical or on-demand fashion. This is important due to the unstable nature of ad-hoc environments. A good example is where links often die due to mobility and thus other optimal routes need to be discovered. Tests should be designed in such a way that the mobile node’s processing power is not wasted (Agrawal, P., et al, 2001). Access is also required to system-specific performance information such as battery life, available memory, CPU capabilities, system utilisation statistics etc. (Bellavista, P., et al 2001) suggest the use of the JNI to provide access to these resources. As mentioned in section 3.2.1, the performance information gathered will be required to be stored in a permanent data store, which will allow for the retrieval of this information by static and mobile agents. The system must also be able to run in either the foreground or background; this option should be set at runtime.
The purpose of the analysis stage is to compare the requirements for the project with current technologies and research with an aim to develop a design that meets the requirements of the project.

3.3.1 Software Development
The proposed MARIAN framework is currently being developed using the Grasshopper ME agent development platform. Fischmeister (2001) and Chatzipapadopolus (1999) carried out a comparative study of agent standards and available platforms, the authors highlighted the Grasshopper platform as being the only one that conformed to all current standards, including the OMG MASIF standards as described in section 2.3.4.2. As the Grasshopper platform is implemented using 100% java, the proposed BSS will also be developed using java. According to research carried out by various authors (Funfrockenm S., 1998, Bellavista, P., et al 2002, Puliafito, A., 2002 and Tripathi, A.R., et al 2002) the java language is currently the best-suited language for the development of agent-based platforms due to the many networking capabilities as well as characteristics such as platform independence, secure execution, dynamic class loading and multithread programming.

As the BSS must be able to run on multiple platforms the application will have to be written in a version of java that will be supported by all target systems. The stationary nodes can use the Java Standard Edition (J2SE) whereas the PDAs, which do not support the full J2SE Application Programming Interface (API); are supported by the Java 2 Micro Edition (J2ME), which is a subset of J2SE. The version of Java that can be used on the PDAs is a full implementation of J2ME Personal Profile provided by the Jeode java runtime environment. Borland JBuilder 3® will be used to develop the system.

It is stated in the requirements that the system must be able to run in either the foreground or the background. This is to enable the order, scheduling and parameters for the tests to be changed and loaded into the system. The loading of these parameters could be from the database or configuration file.

3.3.2 Performance tests
The performance tests can be grouped in the following categories: kernel level, network level, application level and group level. Kernel level tests will aim to identify the hardware characteristics of the devices. Network level tests will aim to retrieve the current status of an ad-hoc network from the viewpoint of each device. Application level tests will aim to monitor the system’s utilisation at a given time. Group level tests will aim to group devices according to common characteristics that they may share; for example all devices with the same operating system (OS) may require a specific test to be run in a specific way, therefore the OS type will be required to calculate what test to run. These tests will be examined in more detail in the design and implementation sections of this report.

The test results gathered from running identical tests on all available hardware and software platforms will be compared with each other to calculate the metric for each device. A typical result of this may show that a laptop with a 1Gb processor and 512Mb RAM, running Windows® 2000 has a metric of 3 in the scale of 1 to 10 for networking capabilities whereas a PDA running with a 400Mhz processor and 64Mb RAM, Windows® CE has a metric of 10. The routing protocol would check this
metric and, if the traffic to transmit were a video clip, it would choose to use the laptop to route the traffic, even if it was not as close in proximity to the source of the transmission as the PDA. If the traffic were a text-based e-mail the routing protocol would consider using the PDA, as this type of transmission does not require much resources.

3.3.3 System-level resources

Further investigation into the JNI has revealed that the method for accessing system resources for all Windows® 95 operating systems onwards are based around the Win32 Application Programming Interface (API). Windows® CE was not developed using the same API’s as the other versions and also only contains a subset of the JNI, this therefore adds additional complexity to the project as we can not apply the same code to all Windows® systems.

After communications with Bellavista, P., (2003), the authors provided samples of their JNI code, which could provide information such as network state, Java thread count and system state including overall system utilisation (Bellavista, P., et al 2000 and 2003). The principle functionality of this code will be developed to suit the needs of the BSS for the project. Unfortunately there still poses a problem of accessing the system-resources for the PDA, which has only a subset of the JNI. Due to the complexity of the problem of obtaining system-level resources for the PDA it was decided to concentrate on obtaining the most important performance information only.

According to Ebert et al 2003, the control of the power consumption of mobile devices in ad-hoc networks largely improves the capacity and extends the lifetime of the network. This view is also taken by Chang et al (2000) who propose algorithms to select routes and corresponding power levels in a static and wireless ad-hoc network such that the system lifetime (in terms of battery life) are maximised. As discussed in section 2.6, the best suited routing protocols for ad-hoc networks are the “on-demand” protocols, which conserve battery life, this suggests that battery-life is a major factor in the efficient routing in ad-hoc networks. Based on these findings it was decided that battery-life was the most crucial factor in determining the ability of a device to effectively route traffic. The available memory is also a valuable resource on PDAs as all data is held in RAM, as there is no hard drive; this therefore limits the amount of data that can be processed to the amount of available memory.

An alternative to using the JNI was sought for obtaining the battery-life and also the available memory on the PDAs. After research into the possible alternative methods of obtaining this information, it was discovered that Microsoft have recently released a set of development tools for WinCE applications called Embedded Visual Tools. These tools are available free of charge from Microsoft and provide support for C++ and Visual Basic programming on all WinCE devices. They provide an interface to the desired native code via API calls, to potentially allow for extraction of the required battery-life and memory usage.

3.3.4 Permanent Data storage

The use of permanent data storage to store network management information, the use of a database in particular, is not a new paradigm. This approach has been adopted in many traditional Client/Server Network and System Management systems (Gavalas, D., et al 2001 and Puliafito,A., et al 2000). The use of permanent data storage is not
restricted to traditional Client/Server systems; Phanse (2000) propose to use central database depositories to administer policies in a wireless ad-hoc network. However, the investigation into the use of a database or some other type of permanent data storage to store local information to aid in the production of a routing protocol is a new approach, which has various challenges (Migas et al 2003a). These challenges are mainly in respect to a common insertion and extraction method to and from the data store. As discussed in section 2.8.2, when using the Java language to connect to databases, it is recommended to use the Java Database Connectivity (JDBC). JDBC will therefore be used to insert and retrieve the data from a Microsoft® Access database.

3.4 Design

The design of the Benchmark System Software (BSS) and how it interacts with the MARIAN framework is described in this section. The design aims to structure the findings in the requirement and design stages to produce a model of how the system will be implemented. As the BSS is primarily a benchmark system, the design is focused on carrying out and storing the results of system tests, the view of the system can be broken down into four main parts; these being:

- Design of tests to extract the performance information
- Design of data storage and retrieval systems
- Overall design of system
- How the proposed system will participate within the MARIAN framework

3.4.1 Design of tests

The desired outcome of the tests is to obtain a system metric that can be used to calculate the abilities of each device, for this purpose identical tests have to be run on each device. The tests will therefore be designed with the least powerful device in mind; for example, the java classes available on the PDA are a subset of the ones available on all other “full” installations. This restricts the implementation of the tests to using the classes that are available to all systems.

Before designing the tests, we have to firstly decide on what information we wish to extract from the devices and for what purpose. We also need to take into consideration that some tests will only be run once while others need to run continuously. The definition of the experiments were outlined in section 3.3.2, these definitions have to be clarified to detail what tests are needed to facilitate the extraction of the required information. The general categories were: kernel level, network level, application level and group level. These categories, with proposed tests, including the reasons the tests are listed below:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>Obtain maximum and current memory</td>
<td>Memory is limited on mobile devices; this test will show the amount available memory. To ensure that device is not overused.</td>
</tr>
<tr>
<td>(continuous)</td>
<td>capacity.</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>Measure memory speed</td>
<td>Memory is a very important factor in the overall system performance, even more so with the PDAs as they have no hard disk therefore memory is the only storage.</td>
</tr>
<tr>
<td>(Initial test)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>Measure CPU processing</td>
<td>To obtain the ability of the CPU to process instructions</td>
</tr>
</tbody>
</table>
battery strength rather than just raw speed. This will indicate how powerful the device is.

Battery Strength (continuous) Measure current battery strength

Section 3.3.3 shows that this is a extremely limited resource which is most probably the highest contributory factor to the performance capabilities of a mobile device

Table 1 – Kernel level tests
The aim of the Kernel level tests is to identify the hardware characteristics of the device.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connectivity</td>
<td>Check current connectivity status</td>
<td>Check for existence of other devices, this test would also check for an internet connection.</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>Measure available bandwidth</td>
<td>Determine how much data can be transferred through the device in a given time, thus allowing the network to use or not use the device depending on the amount of data to be transferred.</td>
</tr>
<tr>
<td>Latency</td>
<td>Measure the speed of the data transmission</td>
<td>Determine the time taken to send a quantity of data over the network.</td>
</tr>
<tr>
<td>Errors</td>
<td>Errors in communication</td>
<td>Determine the reliability of the connection. If there is a high error rate, it will not be used to route time-critical data.</td>
</tr>
</tbody>
</table>

Table 2 – Network level tests
The purpose of the network tests is to monitor the current status of the ad-hoc network from the viewpoint of each mobile device.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Utilisation</td>
<td>Monitor CPU utilisation for each process in system</td>
<td>To establish the current utilisation state of the system. If the device is constantly being over utilised it will not be used to route traffic.</td>
</tr>
<tr>
<td>Memory Utilisation</td>
<td>Monitor the memory being used per process</td>
<td>To establish if there are any processes that constantly are consuming memory resources to establish the suitability for routing.</td>
</tr>
<tr>
<td>Overall Utilisation</td>
<td>Overall utilisation of system resources</td>
<td>A combination of the CPU and memory statistics to establish the overall state of the system. If over utilised it will not be used for routing.</td>
</tr>
</tbody>
</table>

Table 3 – Application level tests
The purpose of the application level tests is to monitor the systems utilisation at a given time.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Obtain details of operating system</td>
<td>To allow for any tests that can only be run on a specific system. For example if the device is a PDA the calls to native code will be different</td>
</tr>
<tr>
<td>Java Version</td>
<td>Obtain details about the Java Runtime version</td>
<td>There may be additional functions that only later versions of the Java runtime may have, knowing the version used would allow for the use of this extra functionality</td>
</tr>
<tr>
<td>User Details</td>
<td>Obtain user details</td>
<td>To allow for security checks and to group users.</td>
</tr>
</tbody>
</table>

Group level tests

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Reasoning</th>
</tr>
</thead>
</table>
Network details (Continuous)  |  Obtain IP addresses and hostname  |  To allow for routing and configuration of the device.

| Table 4 – Group level tests  |

The purpose of the group level tests is to group devices according to common characteristics that they share.

The four levels of tests can be grouped into two broad categories; general tests and native tests. The general tests being tests that can be run from the Java runtime environment using the available class libraries and native tests being tests that will require access to native code. Examples of these are:

**General Tests**
- CPU processing capabilities
- Measure Bandwidth
- Operating system details
- IP address of Hosts

**Native tests**
- Battery strength
- Network Errors
- CPU Utilisation
- Overall System utilisation

As mentioned above, the general tests will be implemented using the built in java class libraries. The difficulty occurs when we require access to native code, as java does not have direct access (Bellavista, P., et al 2001). To enable the extraction of this type of information the JNI, (Section 2.8.1) will be used on all platforms apart from Pocket PC, which runs on the PDA devices. For the PDA it is proposed to develop a Visual Basic program to extract the battery strength and memory usage.

As previously stated, there will be tests that run once and others that run continuously, this does not pose a problem for the Java language as it is has been designed as a multi-threaded environment, providing support for scheduling multiple threads and processes (Lee, J.O., 2000).

### 3.4.2 Design of data storage and retrieval systems

As described in theory section 2.8.2 and again in the analysis stage, section 3.3.4. The BSS system will provide a permanent data store to enable the insertion and extraction of the results gathered from performing the tests. This section briefly describes the proposed high-level design of the database and how it is utilised by the BSS application. The use of a permanent data store has the advantage of allowing insertion and quick retrieval of the results as well as providing an additional level of security by providing password protection. The proposed design of the database access method is based around the three-tier model. In this model, commands are sent to a "middle tier" of services, which then sends the commands to the data source. The data source processes the commands and sends the results back to the middle tier, which then sends them to the user. The use of the three-tier model is very attractive because the middle tier makes it possible to maintain control over access and the
kinds of updates that can be made to the data. Another advantage is that it simplifies the deployment of applications. Finally, in many cases, the three-tier architecture can provide performance advantages. Figure 16 shows an example of the proposed three-tier architecture.

Figure 16 – Database access using three-tier model

3.4.3 Overall design of system

This section describes the overall design of the Benchmark Software System (BSS), taking into consideration the issues raised in the requirements and analysis stages of the report. The main factors that have been identified are:

- The Java code must be designed once and be compatible with all systems
- The tests can be run once or continuously and may require scheduling
- Tests must be identical for each system to enable system performance comparisons
- The system can run the tests automatically, (in the background) or by the user at runtime. This option will be loaded from a permanent store
- The parameters for the tests can be changed at runtime or loaded at start-up
- The data storage model will be based on a three-tier approach
- The system must function within the MARIAN framework

The proposed high-level design of the system as depicted in Figure 17, outlines the overall design structure of the system. The group, network, kernel and application level tests on the diagram will include separate class objects for each test in their respective groups. The introduction of a “File Transfer” class will be required to allow for loading of test and configuration parameters. The system will be designed using an object-oriented approach with all parts of the system being represented by java classes. The use of this approach will allow for flexibility, scalability, security, modifiability and reuse of code. The creation of the system in java will allow for the integration into the MARIAN framework by encapsulating the Java classes, within a static agent, or agents.
3.5 Proposed system design as part of the MARIAN framework

Figure 18 shows how the BSS system will integrate and what part it plays in the overall MARIAN framework. As mentioned in the previous section, for the proposed BSS to finally function within the MARIAN framework, it will have to act as a static or series of static agents. To enable this functionality, the Java classes created will have to be configured to use the Agent Platform Grasshopper ME. This is a simple process, which involves adding references in the java code to use the functions of the Agent Platform.
4 Implementation

4.1 Introduction
This section outlines how the system as described in Chapter 3 was implemented to run on each of the platforms. Due to the large number of tests, a subset have been selected, these are described in detail, highlighting their purpose. Any issues highlighted with regards to the implementation of the system and how they were overcome, have been discussed briefly. The design section defines the characteristics required for the system, these include:

- The Java code must be designed once and be compatible with all systems
- The tests can be run once or continuously and may require scheduling
- Tests must be identical for each system to enable system performance comparisons
- The system can run the tests automatically, (in the background) or by the user at runtime. This option will be loaded from a permanent store
- The parameters for the tests can be changed at runtime or loaded at start-up
- The data storage model will be based on a three-tier approach
- The system must function within the MARIAN framework

The following sections discuss the steps taken to implement the system, taking into consideration the above requirements:

- Implementation of tests
- Running of tests
- Data storage and retrieval

4.2 Implementation of tests
As discussed in section 3.4.1, the aim of the tests is to obtain a set of results that will determine a system metric to aid in the routing process. As identical tests have to be run on all systems, they have been implemented using the Java 1.1.8, and compiled with the 1.2 Runtime environment, Sun Microsystems Inc recommend the use of these environments when developing applications for low specification devices such as PDAs.

All tests have been designed to run as individual java threads; this allows for the tests to be run independently, have the ability to be scheduled, to run concurrently with other tests and to aid reuse. To facilitate the creation, control, scheduling and destruction of the threads for each test, a BSSRunTests class has been developed. This class controls the threads by creating, when requested, a new thread group, it then is able to control all threads by referring to the group rather than each individual thread, it also passes any required parameters to the threads. Figure 17, in section 3.4.3 shows the relationship between the BSSRunTests class and the tests.

As mentioned in section 4.1, only a subset of the whole range of tests are discussed in this section, this is due to the number of tests and the similar methods in which they are implemented. Test not discussed in any detail may be referred to in the text; a full listing of all tests created can be found in Appendix A, 8.1.

The choices of tests to discuss are as follows:
• Client-Server test
• 4D bubble sort
• Remote file download test
• Total system utilisation test
• Battery Strength test

4.2.1 Client-Server test
This purpose of this test is two fold, firstly it will measure the available bandwidth and secondly the Latency of the wireless connection: bandwidth being the amount of data that can be transferred through a connection in a given time period and Latency being the amount of time between the start of an action and its completion. This test runs in the following sequence:

1. Start Server, listens on defined port
2. Start Client, passing in IP address of server as a parameter
3. Server sends varying amounts of data to client starting at 52 to 33280 bytes
4. Client immediately returns the data
5. Server calculates the time taken to return the data, and then saves the results to the database and file.

As described in section the BSSRunTests class coordinates the test by starting the server first, then the client. The coordination is achieved by the use of the java join method, which waits for the threads to complete; this ensures that the system waits until both client and server parts of the test are complete before continuing with its processing. The code extract below is from the BSSRunTests class, it shows the instantiation of both client and server, the addition of each of them into a thread group and also the join method. (Comments are shown in green)

```java
public void runServer(int ID, int distance, boolean OS){
    // starts server on local PC, listening on port 1111
    BSServer s = new BSServer(ID,distance,OS);
    ThreadGroup serverGroup = new ThreadGroup(runGroup,"serverGroup"); // add thread to group
    Thread Ts = new Thread(serverGroup,s,"serverThread");
    Ts.setPriority(6); // set priority
    Ts.start();
    // if the server is waiting on a Remote client wait until thread is complete
    if (ID > 0){
        try {
            Ts.join();
        }catch(InterruptedException e){System.out.println("Error running "+this.getClass());}
    }
} // End runServer
```

```java
public void runClient(String host){
    // starts client on local PC connecting to IP address passed as a parameter
    BSSClient c = new BSSClient(host);
    ThreadGroup clientGroup = new ThreadGroup(runGroup,"clientGroup"); // add thread to group
    Thread Tc = new Thread(clientGroup,c,"clientThread");
    Tc.setPriority(6); // set priority
    Tc.start();
    // wait until thread is complete
    try {
        Tc.join();
    }catch(InterruptedException e){System.out.println("Error running "+this.getClass());}
}
```

The source code for the Client and Server are listed in Appendix A, section 8.2.
4.2.2 4D bubble sort

The purpose of this and all other bubble sort tests is to test the processing capabilities of the CPU. The reason the bubble sort was chosen was due to the rate that the algorithm grows in computational complexity as the number of values increase, this at a rate of $O^{n^2-1}$. This means that for every additional value to sort the calculation complexity increases exponentially. This is therefore an ideal test to tax any type of CPU. The aim of this test is to provide a more accurate comparison between systems’ processing capability than just comparing processor speeds.

The test allows for entry of the number of values to sort. For example if a value of 13 is entered in the 4D bubble sort, this will equate to a value of $13*13*13*13$ (28561) values being sorted, resulting in roughly $28561*28561*28561*28561-1$ comparisons.

The test works as follows, (using the example above):

1. Start test from BSSRunTests class, passing in a value of 13
2. Create 28561 random numbers
3. Loop through the array sorting the numbers
4. Calculate the time taken
5. Store results to file and database

A single, two, and three dimensional bubble sort test was also created to allow for comparison between all types of tests run on different systems, see Appendix A. See Appendix A section 8.3, for the source code for the 4D bubble sort.

4.2.3 Remote file download test

This test attempts to download a HTML page from the Internet; if successful it calculates the time to connect and then the time to download the file. If it is not successful it records the fact that it currently has no route to the Internet. This test can also be used to determine a few factors with regards to the current state of the network and also the capabilities of the device. The time to connect will identify that there is a link and also the speed/latency; the time to download will identify the current bandwidth using the size of the file and the time to download. The choice of file to download can be entered by the user at runtime or as part of an automated test. The test works as follows:

1. Start test from BSSRunTests class, passing in the full Internet address i.e. [http://www.aib.gov.uk/index.html](http://www.aib.gov.uk/index.html)
2. Check for connection and if it exists store time to connect
3. Strip off the file name from the address and attempt to download it
4. Calculate the time to download, and also the size of the file
5. Record all results to the database and file

This test could play a major part in the calculation of an overall system metric especially where the clients in the network wish to send traffic to the Internet, the fact that the device could advertise the fact that it currently knows no route, could prevent unnecessary route discovery requests. The code extract below, from the class BSSConnectivityCheck shows the run method of the test.
public void run(){
    try{
        this.getDetails(theURL);
        // start timer for connection
        long startTimeC=System.currentTimeMillis();
        this.connect();
        long endTimeC=System.currentTimeMillis();
        timeToConnect =endTimeC-startTimeC;
        // start timer for download
        long startTimeD=System.currentTimeMillis();
        // Only get the file if the connection is up
        if  (isConnectionUp)
            this.getFile();
        long endTimeD=System.currentTimeMillis();
        timeToDownload =endTimeD-startTimeD;
        // update file, then database if not a windows CE system
        this.updateFile();
        if (!isCE)
            this.updateDB();
        // Only disconnect if the connection is up
        if  (isConnectionUp)
            // disconnect from server
            this.disconnect();
        // catch errors
    }catch(NoRouteToHostException ex)
    { System.out.println("No connection "+ex);
        isConnectionUp = false;
    } catch(IOException ex)
    {System.out.println("Error in "+this.getClass()+" "+ex);
    } finally
    { System.out.println("End of Remote Test");
    }
}

See Appendix A, section 8.4 for a full listing of the source code.

4.2.4 Total system utilisation test

This test is one of the three tests that use the JNI to access native code, the other two tests being the Network error test and the java threads test, see Appendix A, section 8.1 for details of the latter two tests. The use of the JNI to access native code is a complex process that removes much of the platform independence benefits of using java if not implemented correctly (Bellavista, P., et al 2001). Much of the complexity of using JNI is due to the native code being “native” to each system, subsequently requiring different low-level API calls specific for each system. Bellavista (2001) suggest the way round this is to create a native C\C++ class and dll for each system then determine what type of operating system the code is currently running on and call the respective low-level code for that operating system, via the JNI. This is the approach that has been taken in this report. This is achieved by running a group level test to determine what type of operating system is running, the appropriate native code for the type of operating system is then called using the JNI.

The code developed for this test calls the code supplied by Bellavista (2001). This code returns system information, which is then used by the class BSSTskManager created for the BSS system; the BSSTskManager class then transforms this data to produce the following metrics:
1. For each process: system memory used, total memory used, virtual memory used, percent of the CPU used.
2. The overall system utilisation for all of the above.

Figure 19 details the steps required to create the necessary files to access native code via the JNI. JNI is a two-way API interface that permits java threads to invoke native method, capturing the output from these methods. The diagram points out the stage at which BSS system class BSSTskManager utilises the files created by this process.

Figure 19 – Implementing tests using JNI and native code

This test can be run concurrently with other tests in a test run, to monitor the system state, then terminated at the end of the test run. It can also be run, as can all JNI tests, in continuous mode updating the database at a schedule specified at runtime. For example if the test is started at given an interval of 5000ms, it will continually gather system information at intervals of 5 seconds, each time it runs it will update the database with the latest information, including a timestamp of last update.

The results of this test will aid in the calculation of whether the device has sufficient resources to perform requested routing tasks. Appendix A, section 8.5 lists the source code for the BSSTskManager. The source code supplied by (Bellavista, P., et al 2001) can be found in the CD in the folder Code\BSS\res.
This test has been run on all Windows systems, but as previously mentioned in section 3.3.3, it does not run on Windows CE. This is an area where further development is required.

4.2.5 Battery Strength test

The purpose of this test is to obtain the current strength of the battery and current memory use for the PDA. As discussed in section 3.3.2, the battery strength is widely recognised as the most important and limited resource on mobile devices. Research carried out by Jones et al (2001) demonstrated that 21% of the total power consumption on the mobile devices tested was used by the CPU\Memory and 18% by the wireless interface. A further issue with Pocket PC 2002, which is the type used for the project, is that the Wireless Interface automatically shuts down when the battery strength drops below 20%. With this in mind, it is important not to use a device when it is low on battery strength as the wireless capabilities could shut down in the middle of a transmission if the battery strength drops below this threshold, resulting in the transmission being dropped.

4.2.5.1 Implement Visual Basic System Monitor Interface

Referring back to section 3.3.3, which states that a tool was discovered to develop this test using the Microsoft\® Visual Basic Embedded Visual Tools kit; a few questions had to be answered before this VB application could be designed using this toolkit. These were:

- What information do we require?
- How can the required information be retrieved?
- How can this information be made available to the java application?

The information required is current battery strength as a percent and current memory usage, also as a percent. This information is only updated each time the current values change. As it is not only the current state we are interested in but also the change in state over a period of time.

The second problem was solved by firstly finding out what Application Programming Interface (API) was required to return the current state of the battery and the current memory; this is the file "coredll.dll". A call was then made to this API to retrieve the information.

It was chosen to store the information in text file format, as reading to and from text files is a relatively simple task to implement in both VB and Java applications. The code extract below shows how the functions were declared to enable access to the file "coredll.dll".

```vbscript
Declare Function calls
Declare Function GlobalMemoryStatus Lib "coredll.dll" (ByVal strBuffer As String) As Long
Declare Function GetSystemPowerStatusEx Lib "coredll.dll" (ByVal strBuffer As String) As Boolean
```

The code full code listing is included in Appendix A, section 8.6.

The application was developed with a user interface, which informs the user of the current state of the battery and memory; the information was also stored in three text files. The files, what information they contain and how java uses the information as described below:
BSSCurrentState.txt: This file overwrites itself every 10 seconds with the current battery strength, memory use and timestamp. Java reads this file to get the current state information.

BSSMemoryCE.txt: This file constantly appends the current memory use and timestamp every 10 seconds. Java then clears this file at the start of the test, then reads it again at the end. The information appended by the VB application during the test details all changes in the memory use during the test and when this change occurred.

BSSMonCEBattery.txt: This file constantly appends the current battery strength and timestamp every 10 seconds. Java then clears this file at the start of the test, then reads it again at the end. The information appended by the VB application during the test details all changes in the battery strength during the test and when this change occurred.

This information can be used to not only determine the actual battery strength and memory use, but it also allows us to calculate exactly how much battery life and memory is affected by each test. For example the Client Server test details how much the battery life has been depleted by transmitting a specific quantity of data. Once a metric is devised to represent this test, it could be used to calculate if a device has enough battery strength to route data, and also how much data.

The VB toolkit provides a development environment, which includes an emulator to test the application before installing it on the PDA. This was an invaluable function, as I did not have access to a PDA while developing this application. Figure 20 shows the VB application running on the emulated and also on the actual device.

![Figure 20 – VB Application running on emulator and PDA](image)
(The Emulator is the image on the left. The PDA is the image on the right)

The test implementation section has attempted to give an overview of a number of tests that were developed. These tests have been implemented with the purpose of obtaining performance information to allow for comparisons between systems.
4.3 Running the tests

This section aims to explain the process of running the test. The following steps are explained:

- The operational modes of the system
- Initial group test
- How the runtime behaviour of the system and tests can be configured
- What type and format of information is produced
- How is the information imported and exported from the system

The section concludes with a diagrammatical representation of these steps

**Operational modes**
The system has two modes of operation: User interface and automatic.

**User Interface**
The user interface is menu driven, as the PDA does not allow for a graphical Java interface. The user is authenticated by a username and password, which is validated against the values in the database. The interface allows the user to:

- Run any test passing in values at runtime
- Display test and configuration data from file/database
- Purge database tables
- Change system configuration data

In a fully implemented system the user interface would not be required as all configuration could either be passed in from a configuration file and test files or even by a mobile agent. Figure 21 shows an example of one of the interface system menus.

**Automatic Mode**
Automatic mode is driven an input file, which stipulates which tests to run. If the test is a continuous one i.e. the Total system utilisation test as described in section 4.2, the system will keep running until manually stopped. If the test is a normal test the system will exit after the test run is complete.

![Figure 21 – Network Test Menu on the user Interface](image)
Initial group test
The Group level test runs automatically at start up, gathering information about the operating system and the device. This information is then used to determine how to run certain tests. For example if the operating system is Windows CE and the mode is user interface, the system will disable database functionality, as this has currently not yet been implemented for the PDA. Section 3.4.2 describes the types of information captured by this test in more detail.

How the runtime behaviour of the system and tests can be configured
As stated in operational mode section above, there is an option to run system and the tests automatically or by the user at runtime. To enable the scheduling and structure of any individual “test run”, a type of mini-scripting language was developed. The objective of this language is to provide a format for passing in parameters to the system. In the system there are two distinct methods of passing in this information, these are: by means of a configuration file, passing in start up parameters and also in the form of a test file, which will instruct the system on what tests to run. These methods are briefly discussed below:

- **Configuration file**
  - “BSSConfig.ini”: This will pass in the mode of operation, user interface or automatic and also what test file or files to run, see

- **Test files**
  - Filename.tst: This is loaded during an automatic “test run”; it passes in its name, what tests to run, how many times to run and the parameters.

The files include full instructions on how to use the scripting language. Appendix A, section 8.7 and 8.8 for a sample files.

To aid in the running of tests a set of templates have been produced for all common scenarios and for all target operating systems; these are in the TEST folder on the CD.

What type and format of information is produced
All tests in the system create a set of test results; for each test, these results are formatted in two ways: For output to file and for insertion into the database.

**File output**
The results for each test are calculated in the individual test class, they are then formatted for output to file. This format includes tab delimiting the values to allow for import directly into a spreadsheet. The data is placed in an ArrayList then sent to file. This result file is only a temporary file that is overwritten each time the same test is rerun. Table 5 shows the contents of a temp result file for a file creation test.

<table>
<thead>
<tr>
<th>Files</th>
<th>Size</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>1132</td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td>2012</td>
</tr>
<tr>
<td>300</td>
<td>1</td>
<td>3095</td>
</tr>
<tr>
<td>400</td>
<td>1</td>
<td>4817</td>
</tr>
<tr>
<td>500</td>
<td>1</td>
<td>5778</td>
</tr>
<tr>
<td>600</td>
<td>1</td>
<td>7701</td>
</tr>
<tr>
<td>700</td>
<td>1</td>
<td>8342</td>
</tr>
<tr>
<td>800</td>
<td>1</td>
<td>8963</td>
</tr>
<tr>
<td>900</td>
<td>1</td>
<td>10435</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>16253</td>
</tr>
</tbody>
</table>

Table 5 – Temporary results from file creation test
The reason that the result files for each test are only temporary is that there can be more than one test in a test run. A test run could include the same test 100 times or a mixture of all tests with different parameters. For each test run there is an overall results file, created by the class “BSSCreateTestData”. This class collates the information from all tests in the test run. A summary of the information collected is as follows:

- Name of source test file ".tst" that was loaded
- Start time of "test run"
- For each test:
  - Test name and parameters passed in
  - Start time
  - Results
  - End time
- Overall test run time
- The end time of the "test run"
- Overall time for all tests to run

The code extract below shows the `addToResults` function from the “BSSCreateTestData” class, which uploads a result file for an individual test, then appends the contents to an ArrayList, which will produce the final overall results file.

```java
private synchronized void addToResults(String file){
    // function builds up the values in the arraylist to create the overall results file. Input to the function is an String representing the file to load
    ArrayList al = new ArrayList();
    BSSFileTrans fileIn = new BSSFileTrans();
    al = fileIn.getDataFromFile(file);
    Iterator alIt = al.iterator();                  //loops through the list adding values from the file into the results
    while (alIt.hasNext()) {
        Object tempValue = alIt.next();  // loop through the array until no values are left
        String s = (String)tempValue;
        alData.add(s);                             // append to the overall results alData ArrayList
    }
}
```

The code extract, also from the “BSSCreateTestData” class, parses the data uploaded from the ".tst" file, extracting the test names to run and also the parameters to send to each test. It then sends the data to another function to start the tests running.

```java
private synchronized void setTestValues(ArrayList al){
    // ArrayList is passed an, testname and value is then passed to the function startTestsFromFile to run the tests
    String StrVal = "";
    alData.add("Tests loaded from file:");           // Input new line stating what tests are being run
    try{
        Iterator alIt = al.iterator();                          // loop through the ArrayList looking for variables
        while (alIt.hasNext()) {
            Object tempValue = alIt.next();  // parse the Object
            String s = (String)tempValue;              // check for non test running character #
            hash = s.indexOf("#"/Delete character #");
            if (hash <= 0){                                       // no hash
                i = s.indexOf(" ");                        // look for value
                if (i >0){
                    StrVal = s.substring(i+1,s.length ());
                    // Get the name of the test
                    this.startTestsFromFile(s,StrVal);   // pass name of test and value to function startTestsFromFile()
                }
            }
        }
    }
}
```
To ensure that each overall “test run” results file is not overwritten a unique naming convention was created for the results files. This is calculated by appending the current time and date to the name of the “.tst” file that was loaded then finally giving it a .txt extension. An example file name is “BSS-level0Test231103-150415.txt”.

For the PDA, there is additional information such as the battery strength and memory usage at the beginning and end of the whole test run and also for each test. The history of the changes in memory and battery strength as recorded by the VB application throughout the test run are also listed at the end of the results file. A sample of a results file is shown in Appendix A, section 8.9. The test run in this example was a four-dimensional bubble sort. This overall test results file provides a comprehensive listing of all test data gathered. This data can then be easily extracted for production of the system metrics. Figure 23 shows the sequence of steps involved in loading, running, collating and saving the results of the 4D bubble sort to file; the Battery file is created by the VB application and is constantly updated during the test.

How is the information imported and exported from the system

In the analysis section 3.4.3 the concept of a FileTransfer class was introduced, see Figure 17 in the aforementioned section. This class has been developed as a generic file reader and writer class which not only reads and writes to file but it also parses each input file stripping out comments, leaving only the required data. When a file is to be read in i.e. a “.tst” file, the class is passed the filename to read. It then reads in and parses the file, the parsed file is then passed to the calling function as an ArrayList. When a file has to be written i.e. the results of a test, the filename is passed to the class along with an ArrayList containing the contents of the file; the class then creates the file and inserts the values. The code extract below shows the function in the FileTransfer that loads, parses then returns the file contents as an ArrayList to the calling function.

```java
public synchronized ArrayList getDataFromFile(String filename) {
    // Receive filename from class and return an ArrayList of values
    ArrayList al = new ArrayList();
    try {
        String line = "";
        this.fileOpen(filename); // open file
        while (line != null) {
            line = input_file.readLine();
            if (line != null) {
                if (line.startsWith("//")) { // do nothing as it is a comment line
                    } else {
                        al.add(line); // add line of file to arraylist
                        }
                    }
            } // while
            input_file.close(); // close file
            } catch (EOFException e) {System.err.println("eof");}
        catch (IOException e) {System.err.println("No File Found");}
    return al; // Return the parsed data to the calling function
            }
```

Database output

For all non-PDA systems the data is also stored the relevant database table. This process will be explained in more detail in the next section.
Figure 22 – Sequence diagram of 4D bubble sort test run

(Only one test is shown in this example. The CreateTest class can create any number of tests passed in by a .tst file. The overall results would then be saved to the overall results file)
4.4 Data storage and retrieval

The database feature, at present, is designed for all non-PDA systems. The reason for this is that after investigation into use of a database with Windows CE (Pocket PC 2002) it was found that the implementation of the JDBC, in this case Windows CE®, was fully documented but was still in the final stages of its draft form. (Ellis, J) from Sun Microsystems Inc, confirms the draft status of the implementation of the JDBC Optional Package for CDC/Foundation Profile in a response to an email enquiry by myself, see Appendix A, section 8.10.

There are various technologies available for connection to a database on this operating system. For example Microsoft® have implemented the ADOCE that interfaces form applications created in the Visual Toolkit that was used to created the VB application for this project. Unfortunately, the use of ADOCE connection method is restricted to Microsoft products and cannot connect directly to a Java application. Other products, such as Sybase SQL Anywhere Studio® and Microsoft SQL CE 2.0® are designed specifically for use by mobile devices. However these products are designed for synchronising with a central database and not for use with a stand-alone database, which is the type required by the project. All data storage for the PDA has therefore been implemented using text-based file formats.

Referring back to Chapter 3, section 3.4.2, the architectural design used for the database storage and retrieval method is based round the Three-tier model, this is shown in Figure 17 on the aforementioned section. The implementation of the middle tier has been realised by creating a DataTrans class. This class deals with all input and output from the database. As the database is password protected the DataTrans class has to provide a database username and password before it will be allowed to connect. The implementation of this model is explained below with the use of a scenario.

Scenario: User runs Client-Server test, then displays results from database, user then purges the database

1. User runs client-server test.
2. Client-server test stores results in database.
3. User requests the results of the test from the database.
4. Database returns results of test.
5. User then clears the results by purging the database table.

A high-level view of this scenario is depicted by Figure 23.

As the DataTrans class has many functions, full code listings are not included in this document. See enclosed CD for full code listings. I have listed some of the functions use to carry out the above scenario in Appendix A, section 8.11.

The scenario above utilises three functions of the database; these are: Insert test data into, query, return a result-set, and purging the database. In this scenario the DataTrans class also opens and closes connections to the database using the JDBC Bridge. In the scenario above the database only returns the test results for display to the output screen. The system also has the ability, using the DataTrans class to retrieve data to be used in to perform calculations or make computational decisions at runtime. This functionality was designed for use by Mobile agents to retrieve the required system performance metrics.
As mentioned above, the *DataTrans* class performs various tasks; these tasks are performed by separate functions within the *DataTrans* class. This design is to enable reuse of these functions. For example, the function `getDataAndMetaData` is passed in an SQL query statement, it then runs the query against the database and returns the results as well as the metadata describing the database table and columns. This function can query any table from any database. See Appendix A, section 8.12 for layout and description of all current database tables.

### 4.5 Conclusions

This section has provided an in-depth look at how the implementation of the requirements, analysis and design stages from Chapter 3 was completed. The three main stages discussed were: implementation of tests, running of tests and data storage and retrieval. The two major problems highlighted with regards to implementing the system were both in relation to the PDA. The first problem was the incompatibility of the Pocket PC operating system with the JDBC, this problem had to be addressed by using text-based files for the PDA. Ellis, J (2003) from Sun Microsystems has advised me that a full implementation of the JDBC for the Pocket PC will be available in the near future; this would allow the database features of the system to be extended to all platforms. The second issue raised was again in respect to the Pocket PC operating system. This was concerning the implementation of the JNI to access native code to allow for monitoring of system resources such as battery strength and memory usage. This problem is due to the lack of support for common native code libraries. To overcome this problem without using the JNI, as this is a complex process (see Figure 17), a small VB application was created to monitor the battery strength and memory of the PDA. This provided valuable information, which is crucial in determining the overall routing capabilities of the device Migas, N., et al (2003b).
5 Testing and Analysis

5.1 Introduction
The purpose of this section is review the information gathered as a result of running the tests implemented in the Chapter 4. As described in the aforementioned Chapter, the aim of the tests is to obtain a set of results that will determine a system metric to aid in the routing process. The determination of the fitness of mobile devices is an important issue. Determination outcomes can be used to build robust, effective, and efficient routing protocols (Migas, N, et al 2003b). As described in Chapter 3, the tests have been categorised into levels; these were: kernel level, network level, application level and group level. To assist in the determination of the overall fitness metric for each device the following steps have been carried out for each type of test:

- Run identical tests on each device
- Produce test results for each device
- Compare the results of each device to determine the performance level
- Run a continuous test determine how the battery strength is affected by the test
- Produce a metrics table for each type of test, using a simple metric
- Discuss findings of test

Once we have carried out all tests, a final metric table will be compiled from all of the individual tables to determine the overall performance of each device compared to all others tested. See Appendix A, section 8.1 for a complete list of all tests.

5.2 Hardware device specifications
The hardware devices used for carrying out the tests, including their specifications and the name they are referred to by during the test, are described below:

**Personal Digital Assistant (PDA):** Hewlett Packard iPAQ, 450Mhz CPU, 64Mb RAM, wireless LAN card; was running Pocket PC 2002.

**Laptop (Laptop):** Dell Inspiron, PIII 1100 Mhz CPU, 512 Mb RAM, Dell TrueMobile wireless LAN card; was running Windows 2000 Professional.

**Server (Server):** Dell Desktop PC, PIII 450Mhz CPU, 256Mb RAM, connected to Dell TrueMobile wireless base station. Also connected to broadband router providing Internet access. Was running Windows 2000 Server

**Workstation (PC):** JKCC Cluster 10 workstation. P4 1999Mhz CPU, 512 RAM, was running Windows XP Professional. Had no wireless connection.

As mentioned in the introduction section, the tests were run on each of these devices with identical parameters.
5.3 Kernel Level tests

The purpose of the kernel level tests is to measure the capabilities of the CPU and memory of the device. The test carried out are broken down as follows:

- Bubble sort tests
- File creation tests
- Client-Server on local device

5.3.1 Bubble sort tests

The following bubble sort tests were run with comparative values for each dimension. For example, the 1D bubble sort was run with 30000 integers, the equivalent values in the further dimensions were: 2D with 173, \((173 \times 173 = 29929)\), 3D with 31, \((31 \times 31 \times 31 = 29791)\), 4D with 13, \((28561)\). The purpose of using different dimensions was to compare the performance of the CPU with different complexity of algorithms rather than just use the same dimension with different values.

Comparisons

The following graphs present the results from running all dimension bubble sorts on the same device and comparing the performance between each sort.

![Graphs showing bubble sort comparisons for different devices](image)

**Figure 24 – Bubble sort comparisons, per device**

It can be seen from Figure 24, results for each dimensional sort are consistent, with the results from each sort seldom overlapping. However the results for the PDA can vary dramatically, with continual overlapping. This highlights the difficulty in determining the performance of the PDA. It can also be observed that the PDA is over 50 times slower than the PC in most cases and 20 times slower than the Laptop.
G. Sinclair, BEng (Hons) Software Engineering, 2003

Figure 25 compares the 1D and 4D bubble sorts for each device; this clearly shows the different performance capabilities of each device.

![Figure 25 – Comparisons between devices running Bubble sort](image)

**Effects on battery Strength**

Figure 26 illustrates the time consumed for the PDA battery to discharge by 1%. This was measured during a continuous 1D and also 3D bubble sort test. The tests started with battery strength at 100% and stopped when it reached 10%.

![Figure 26 – Bubble sort: time to discharge 1% of battery](image)

Figure 26 shows a couple of anomalies in the battery changes. The battery, in both tests normally discharges one percent per minute and a half. However, during the 3D, in a couple of instances the battery discharges quicker, almost in 20 seconds. This is most probably due to the fact that the battery is not new. The discharge rate was very similar in both tests, 84 seconds per 1% for the 1D sort and 83 for the 3D; this suggests that the battery will not necessarily discharge at a higher rate while the device is performing a more complex algorithm.

**Metrics table**

The metrics shown in Table 5 have been calculated based on the average time taken to run the test, in seconds, over 20 iterations. The metric is scaled between 1 and 100, with 1 being the strongest device and 100 being the weakest. The threshold for a device being the most weak is 500. The metric is calculated by dividing the average for each test by 500 then multiplying the result by 100; this returns a metric between 1 and 100. It can be observed by the figures in Table 5 that the PC, as you would expect has been given a metric of 1.88 for the 4D sort, whereas the PDA has a metric of 77.04.
Table 5 – Bubble sort metrics table

<table>
<thead>
<tr>
<th>Metric</th>
<th>PDA</th>
<th>Laptop</th>
<th>Server</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D</td>
<td>318.32</td>
<td>13.94</td>
<td>26.11</td>
<td>5.60</td>
</tr>
<tr>
<td>2D</td>
<td>348.71</td>
<td>22.23</td>
<td>44.50</td>
<td>10.01</td>
</tr>
<tr>
<td>3D</td>
<td>379.99</td>
<td>21.78</td>
<td>61.76</td>
<td>8.90</td>
</tr>
<tr>
<td>4D</td>
<td>385.18</td>
<td>25.57</td>
<td>71.91</td>
<td>9.41</td>
</tr>
</tbody>
</table>

Figure 27 – Comparison between devices creating files of varied size.

Figure 28 – Comparison between devices creating files of fixed size.

5.3.2 File creation tests

The purpose of this type of test is to measure the performance of the system memory and hard drive access times. There are two types of test used to measure these capabilities:

- Create a fixed number of files of increasing sizes, ranging from 0KB to 2000 KB, in increments of 100KB. The number of the files of each size created during the tests was: 1 in the first test, and 16 in the second test.
- Create varied amounts of files of fixed size, ranging from 100 files to 2000 files, in increments of 100. The size of the files created during the tests was: 1KB in the first test, and 16KB in the second test.

The values presented in Figure 27 show the results of the first type of test. Figure 28 shows the results of the second type of test.
The results presented above clearly show the differences in the performance capabilities of each device in relation to their specifications. The difference is however not nearly as great as that of the bubble sort tests. For example, for the test creating 16 files per size the PDA is only, on average 12.68 times slower than the PC. This could be mainly due to the fact that the PDA has no hard drive to store the files, resulting in the use of RAM, which has a far quicker access time than a hard drive. This pattern is slightly different for the second test, which demands more processing power from the device, as it is creating more files. In this test the PDA is on average 30 times slower than the PC. A possible reason for this is due to the PDA running low on memory as it creates the files in its RAM. For example for the PDA to create 2000 files of 16KB it will use nearly 32Mb of its memory. This would severely impair performance as it has only a total of 64Mb.

**Effects on Battery Strength**

Figure 29 and Figure 30 illustrate the time consumed for the PDA battery to discharge by 1%. Figure 29 shows the results from obtained during a continuous file creation test, creating files of a varied size. Figure 30 shows the results from obtained during a continuous file creation test, creating files of fixed size. The tests started with battery strength at 100% and stopped when it reached 10%.

The results presented in Figure 29 show that on average it takes 119 seconds for the battery to discharge 1% when creating 1 file, whereas it takes 93 seconds when creating 16 files. This clearly indicates that the more processing power used the greater the effect on battery power. These findings are reiterated in Figure 30 where the increase between creating 1KB files and 16KB files is even greater. For example, on average the 1KB test causes the battery to discharge 1% every 114 seconds whereas the 16KB test causes a discharge every 62 seconds.
The metrics shown in Table 6 and Table 7 represent the results from the varied and fixed file creation tests respectively. The metrics have been calculated based on the average time taken, in seconds, to run the test over 20 iterations. As described in the previous section, the metric is scaled between 1 and 100, with 1 being the strongest device and 100 being the weakest. The threshold for the varied file size has been set at 300 whereas the threshold for the fixed file test, which requires greater processing power, has been set at 7000. Dividing the average for each test by the threshold value, then multiplying the result by 100 produces the metric. When comparing the metrics produced in table 7 to that of table 6, the figures illustrate that the PDA is better suited to less process intensive tasks. This is also shown in table 7 where the increase in the time and metric for each device between the 1KB and 16KB tests are 19% whereas the PDA the increase is over 22%.

![Figure 31 – Client server test on local device, comparison between devices](image)

**5.3.3 Client-Server on local device**

The purpose of this test is to measure the processing capabilities of the device by sending data down the TCP stack using the server test and returning that data back through the stack using the client test. The test sends a series of byte streams starting with 52 bytes and ending with 33,280 bytes, incrementing the amount of bytes by 52 each time. A total of 560040 bytes are sent through the TCP stack and back again. The following graph presents the results gathered by running the test on all devices.

<table>
<thead>
<tr>
<th>Table 6 – Metrics produced for varied file test</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Metrics 1= highest, 100 = lowest. Value of 100 = 7000</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1 KB</td>
</tr>
<tr>
<td>16 KB</td>
</tr>
<tr>
<td>Metric</td>
</tr>
<tr>
<td>16KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 7 – Metrics produced for fixed size file test</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Metrics 1= highest, 100 = lowest. Value of 100 = 7000</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1 KB</td>
</tr>
<tr>
<td>16 KB</td>
</tr>
<tr>
<td>Metric</td>
</tr>
<tr>
<td>16KB</td>
</tr>
</tbody>
</table>
The results as illustrated in Figure 31 re-emphasise the difference in processing capabilities that exist between the weakest and the strongest devices. There is however much less a gap in this test than the others. In contrast to results presented in sections 5.3.1 and 5.3.2, this set of test results show the devices as being much closer in performance. For example the average time for the PDA was 43.35 seconds; this is only 6.86 seconds slower than the laptop which was 36.46. The performance of the PDA is however inconsistent, this is shown in Figure 31.

**Effects on battery Strength**

Figure 32 illustrates the time consumed for the PDA battery to discharge by 1%. This was measured during a continuous Client-Server test as described above. The tests started with battery strength at 100% and stopped when it reached 10%.

In contrast to the results from the bubble sort test from section 5.3.1 and the file creation tests from section 5.3.2, the average time for the battery to discharge in this test was 172 seconds in comparison to the 1D bubble sort, which was 83 and the fixed file test creating 1KB files, which was 114 seconds. This is most likely due to the fact that the client-server test is less processor intensive. As discussed in section 5.3.3, this highlights the relationship between the complexity of processing and the time taken to discharge the battery.

**Metrics table**

The metrics shown in Table 8 represent the results from the above client-server test. The metrics have been calculated based on the average time taken, in seconds, to run the test over 20 iterations. The metrics are calculated in the same manner as described in Sections 5.3.1 and 5.3.2. The threshold value used to calculate the metric for this test has been set at 50; this is due to the low processing power required to carry out the test. As discussed previously, the results suggest that the devices are far closer in performance capabilities that all other tests carried out so far. This is one of the main reasons why the overall system metrics will be based on a combination of all test, rather than on a single set of tests.

<table>
<thead>
<tr>
<th>Metric</th>
<th>PDA</th>
<th>Laptop</th>
<th>Server</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>43.35</td>
<td>36.46</td>
<td>43.68</td>
<td>27.93</td>
</tr>
<tr>
<td>Value of 100 = 50</td>
<td>86.71</td>
<td>72.92</td>
<td>87.37</td>
<td>55.87</td>
</tr>
</tbody>
</table>

Table 8 – Metrics table for Client-Server on local device
5.4 Network level tests

The purpose of the network level tests is to measure not only the networking capabilities of the device but also to monitor the state of the network. The test carried out are broken down as follows:

- Remote Client-Server tests
- Remote file download tests
- Network traffic monitor test

A full list of all tests is shown in Appendix A, section 8.1.

5.4.1 Remote Client-Server tests

The purpose of this test is to allow for checking the Bandwidth and latency of the network by transmitting data between nodes, then calculation the time taken. The test is carried in similar manner as the local client-server test described in section 5.3.3, the only difference being the fact that server and client are on separate devices. The server class sends a series of byte streams to a remote client starting with 52 bytes and ending with 33,280 bytes, incrementing the amount of bytes by 52 each time. The client immediately returns the data to the server, which calculates the time taken. The tests were carried out firstly, with the devices being 1 metre apart and again with the devices being 10 metres apart.

Figure 33 presents the results gathered by running the Client-Server tests for all devices as described below:

- Laptop1 acts as server, sending data to a client on another wireless laptop2
- PDA acts as server, sending data to a client on laptop1
- Server machine, acting as server, sends data to client on laptop1 via a wireless base station

Note: This test was only carried out on devices that were able to transmit over a wireless connection.

Results presented in Figure 33 show distinct difference in performance at a distance of 10 metres however the results of the 1 metre test do not reflect the corresponding specifications of the devices. This could be due to the fact that the size of the data streams being sent, and also the distance between the devices being relatively small.
and subsequently not overburdening the slower devices. This suggestion is reflected in the average times for the devices for the 1 metre test, these being: 52.55 for the PDA, 38.11 for the Laptop and 44.52 for the Server. And for the 10 metres test: 70.17 for the PDA, 38.14 for the Laptop and 43.90 for the Server. To assist more accurately determining the networking capabilities of the devices in respect to this type of test, I suggest further tests be carried out, which would send much larger amounts of data through each device.

**Effects on battery Strength**

Figure 34 illustrates the time consumed for the PDA battery to discharge by 1%. This was measured during a continuous remote Client-Server test as described above. The tests started with battery strength at 100% and stopped when it reached 10%.

![Figure 34 – Remote Client-Server test: time to discharge 1% of battery](image)

The results presented in Figure 34 show that the battery consumption on both tests is relatively smooth apart from one anomaly in the 1 metre test. Referring to the similar result in section 5.3.2, Figure 26, this is likely to be the result of an ageing battery. The average time taken for the battery to discharge by 1% during the tests was 72 seconds for the 1 metre test and 67 seconds for the 10 metre test. These figures suggest that there is a direct relationship between the transmission range and the use of battery power. Ebert et al (2003) came to this conclusion also, in their research into the power consumption of mobile devices in ad-hoc networks. The drain on battery power by the wireless network card can be clearly seen when we compare the average time for the tests above with that of the local Client-Server test as described in section 5.3.3. The average time taken for battery to discharge by 1% during the local test was 172 seconds whereas the same test over the network, even at just 1 metre took 72 seconds to discharge by 1%. This suggests that the wireless card doubles the rate of battery discharge during the tests carried out.

**Metric Table**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value of 100 = 80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Metre PDA</td>
<td>70.17</td>
</tr>
<tr>
<td>1 Metre Laptop</td>
<td>38.14</td>
</tr>
<tr>
<td>1 Metre Server</td>
<td>43.90</td>
</tr>
<tr>
<td>10 Metres PDA</td>
<td>52.55</td>
</tr>
<tr>
<td>10 Metres Laptop</td>
<td>38.11</td>
</tr>
<tr>
<td>10 Metres Server</td>
<td>44.52</td>
</tr>
</tbody>
</table>

Table 9 – Metrics table for Remote Client-Server test
5.4.2 Remote file download test

This test attempts to download a HTML page. If successful it calculates the time for download and shows that there is a connection, if not it shows that there is no Internet connection. The information gained from carrying out this test could therefore be used to determine if there is an Internet connection available and if so, what is the connection and download times for a specific file size. The test attempted to download a single HTML file of 26823 bytes. Three different measurements were extracted from running the tests, these were:

- Time taken to connect
- Time taken to download
- Total time taken

The results from running these tests are illustrated by Figures 35, 36, and 37.

![Connection times](image1)

**Figure 35 – Remote file download, connection times**

![Download times](image2)

**Figure 36 – Remote file download, download times**

![Total time for test](image3)

**Figure 37 – Remote file download, total time taken**
The results presented above show consistent connect, download and total times for all devices apart from the PDA. The results for the PDA are similar to both Local and remote Client-Server tests with respect to the sporadic nature of the results. Table 10 shows the average, connection, download and total times per device.

<table>
<thead>
<tr>
<th>Average times by device</th>
<th>PDA</th>
<th>Laptop</th>
<th>Server</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>7.46</td>
<td>1.27</td>
<td>0.22</td>
<td>0.15</td>
</tr>
<tr>
<td>Download</td>
<td>91.47</td>
<td>1.18</td>
<td>1.63</td>
<td>0.96</td>
</tr>
<tr>
<td>Total</td>
<td>98.93</td>
<td>2.46</td>
<td>1.85</td>
<td>1.12</td>
</tr>
</tbody>
</table>

Table 10 – Average Remote download times by device

As illustrated by Table 10, the PDA is struggling to compete with all other devices, especially with regards to the downloading of the file. The PDA is nearly 100 times slower than the PC and 57 times slower than the server. The other devices are very similar in performance with the Laptop taking less than one tenth of a second to connect than the PC. This result was expected due to fact that both the PC and the server are connected to the Internet via a fixed connection.

The metrics calculated for this test are shown in Table 11. These metrics bare no relation to the Remote Client-Server test where the PDA performed much more favourably against the other devices.

<table>
<thead>
<tr>
<th>Metrics</th>
<th>PDA</th>
<th>Laptop</th>
<th>Server</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric</td>
<td>98.93</td>
<td>2.46</td>
<td>1.85</td>
<td>1.12</td>
</tr>
</tbody>
</table>

Table 11 - Metrics for Remote file download test.

### 5.4.3 Network traffic monitor test

The purpose of this test is to monitor the local network traffic, gathering statistical information on the amount of traffic being processed by the device as well as any errors. The metrics gained, in conjunction will all other metrics, can be used to determine if the device, coupled with the state of its immediate network are suitable for routing particular types of data. The types of traffic monitored by this test are TCP, IP, and UDP. The statistics gathered for each protocol include: packets in, packets out and errors. The test can be scheduled to repeat at a given interval and also may be run as a background task or manually from the user interface. After a snapshot of the current network state is captured, the data is stored in the database, overwriting the previous outdated data. Table 12 shows an example snapshot of data captured during the test.

This test can be run on all devices apart from the PDA. This is due to the fact that the test requires access to native code to retrieve the required statistics. Section 3.3.3 describes the problems associated with accessing native code from the PDA.
es only, a simplified method of creating metrics below were calculated as follows:

To calculate the error rate, divide the total number of packets in and out, and then finally multiply the value by 100 to obtain a metric in the range of 1-100. Using the udp protocol below as an example, this equates to 1/(0+2)*100, which equals 50%. These metrics can then be used alongside all other metrics gathered to determine an overall system metric.

Table 13 – Metrics table for network monitor test

5.5 Application level tests

The purpose of the application level tests is to monitor the system utilisation per process both in terms of CPU and memory. Further information gathered, such as total system utilisation and memory capacity as well as information on current java threads, will contribute to the overall system utilisation metric. Due to the tests requiring access to native code via the JNI, they cannot be run on the PDA. Application tests to obtain this information include:

- System utilisation test
- Java Threads test
5.5.1 System utilisation test

The purpose of this test is to monitor the utilisation of the device to provide a current system state. This is an important test in terms of determining the routing capabilities of a device. Unlike most other tests, the metric from this test can be described as dynamic, as it vary dramatically depending on the state of the device. For example, if the system usually had a very good metric based on the other tests alone, the routing protocol may route large amounts of data through the device. However, if we include the results of this test in the calculation of the metrics, the protocol would not route data through the device if the system were already being over utilised. The test can be scheduled to repeat at a given interval and also may be run as a background task or manually from the user interface. After a snapshot of the current system state is captured, the data is stored in the database, overwriting the previous outdated data. Table 14 shows an example snapshot of data captured during the test.

Table 14 – Data captured during the system utilisation test.

Metrics gathered from this test

As discussed above, the metrics from this test could make a significant difference to the overall system metrics. The metrics shown in Table 15 have been calculated from the snapshot of data shown in Table 14. These figures illustrate a simplified metric, taking into consideration only the average of the CPU and memory utilisation. In a fully implemented system it would be more appropriate to take into consideration all data produced by this test and place a weighting on each metric produced depending on the type of task the device was required to run. For example if the device was required to calculate complex routing tables, which is a processor intensive task, the CPU utilisation metric would be given a higher weighting as it would be utilised more during the test. The simplified metric below was calculated as follows: The percent of memory used was added to the percent the CPU was being utilised. The two were added together, and then divided by 2 to obtain the overall system metric of 66.5%.

<table>
<thead>
<tr>
<th>BSSTskManager: sample Metrics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cpuPercentUsed</td>
<td>91</td>
</tr>
<tr>
<td>percentFreeMemory</td>
<td>58</td>
</tr>
<tr>
<td>Test Metric</td>
<td>66.5</td>
</tr>
</tbody>
</table>

Table 15 – Metric for overall utilisation test
5.5.2 Java Thread Test

The purpose of this test is to monitor all Java threads in the system and store a snapshot of the data. As the BSS system, the Grasshopper Mobile Agents platform and the mobile agents themselves are written in Java, this data could be used to monitor their overall activity. As with the Network Monitor test and the System utilisation test, this test can be scheduled to repeat at a given interval and also may be run as a background task or manually from the user interface. After a snapshot of the current system state is captured, the data is stored in the database, overwriting the previous outdated data. The data captured in this test is also captured by the overall system.

The figures shown in Table 16 show a sample data set from a Java thread test.

<table>
<thead>
<tr>
<th>Month</th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Aug</th>
<th>Sep</th>
<th>Oct</th>
<th>Nov</th>
<th>Dec</th>
<th>Jan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 2</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 3</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 16 – Sample data captured from a Java threads test.

In addition to the capture of the individual Java threads at a scheduled interval, the system also calculates the total utilisation and stores it in a database table. These totals as illustrated in Table 17 provide a history of the utilisation of the Java threads. This could be used to monitor the utilisation of all Java applications over a period of time.

<table>
<thead>
<tr>
<th>Task 1</th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Aug</th>
<th>Sep</th>
<th>Oct</th>
<th>Nov</th>
<th>Dec</th>
<th>Jan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 2</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 3</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 17 – History table of Java Thread totals
The purpose of this test is to provide the BSS with system information at runtime to enable the correct programmatic actions to be taken in the case where there are different actions or tests that are run on different systems. This was one of the simplest tests to implement as all of the data acquire was accessible via the java.system API. When the system is started, the test is run automatically to ensure that any system or user changes are taken into consideration at runtime. The test may also be run at runtime from the user interface. Table 18 illustrated an example of the information captured by this test.

There are various uses for this test, examples of some of these uses could be:

- Check IP address of system, or if there are more than one address
- Check user to enable authentication
- Determine Java Runtime Version to establish what java classes are supported
- Check operating system; this is used in the BSS to check for the PDA.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Aug</th>
<th>Sep</th>
<th>Oct</th>
<th>Nov</th>
<th>Dec</th>
<th>Jan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info on system capabilities</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Individual software and hardware</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relevant initially but not functionality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baseline/Version</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complex functionality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component/Build part</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation of baseline version</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation of feature functionality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baseline/Version</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relevant initially but not functionality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Individual software and hardware</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Info on system capabilities</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 18 - Sample data captured by the group level test

**Metrics**

This test does not contribute directly to the calculation of the system metrics as none of the data represents a system performance indicator. However, the data captured can be used by the system to help determine what test can be run and therefore what types of system metrics can be obtained.
G. Sinclair, BEng (Hons) Software Engineering, 2003

5.7 Conclusions

In this section we analyse the metrics that have been gathered by all of tests to obtain a total system metric. As discussed in all Sections of Chapter 5, this proposed metric is a simplified version, which is being used to illustrate how a more precise set of metrics may be obtained in a fully implemented system. For each of the comparative tests run, the time to discharge the battery was examined to determine the amount of battery strength was lost as a percentage during the tests. The main purpose for this was to establish a pattern, which could be used to determine if there is sufficient battery power left to perform a specific task. Table 19 represents the average percent of battery strength used during the continuous tests.

5.7.1 Comparison of how each test depreciates battery power

<table>
<thead>
<tr>
<th>Battery % reduced by test (Avg over 20 tests)</th>
<th>% Battery used</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D Bubble</td>
<td>3.00</td>
</tr>
<tr>
<td>2D Bubble</td>
<td>3.20</td>
</tr>
<tr>
<td>3D Bubble</td>
<td>3.40</td>
</tr>
<tr>
<td>4D Bubble</td>
<td>5.00</td>
</tr>
<tr>
<td>File test 1 - 1KB</td>
<td>0.67</td>
</tr>
<tr>
<td>File test 1 - 16KB</td>
<td>2.90</td>
</tr>
<tr>
<td>File test 2 - 1KB</td>
<td>27.50</td>
</tr>
<tr>
<td>File test 2 - 16KB</td>
<td>34.40</td>
</tr>
<tr>
<td>Client-Server Remote</td>
<td>2.00</td>
</tr>
<tr>
<td>Client-Server Local</td>
<td>1.00</td>
</tr>
<tr>
<td>Merge Sort</td>
<td>0.55</td>
</tr>
<tr>
<td>Download HTML file</td>
<td>0.79</td>
</tr>
</tbody>
</table>

Table 19 – Average battery strength used during the continuous tests

The results presented in Table 19 highlight various patterns in respect to the loss of battery power. For example it can be clearly seen by the results of the bubble sort test that, as we increase the complexity of the processing task the amount of battery strength used also increases. A similar pattern can be seen when comparing the Client-Server Remote test with the results from Client-Server Local test. These tests are identical in terms of structure and values used, the only difference being that the Remote test uses the wireless LAN card to transmit and receive the data over the network. From this we can deduce that the process of transmitting data over the network doubles the rate of the depreciation of the battery. It can be observed from the results of the File 2 tests that the creation of large amounts of files has a severe effect on the depreciation of the battery power. This result could be used by the routing protocol to determine if the device is suitable for routing large amounts of files.
5.7.2 Overall system metrics

This section briefly discusses the overall system metric gathered from all tests. The metrics illustrated in Table have been gathered from the results described throughout all sections in this Chapter. The purpose of gathering all of the metrics into a central table is to balance the results to give a clearer picture of the overall performance of the devices in comparison with each other. Table 20 presents the results for each device tested and also proposes a final system metric. The final metric shown in the metric was calculated in an over simplified manner to demonstrate the theory of using a metric-driven protocol; each metric shown represents a figure in the range of 1-100, 1 being the strongest device and 100 being the weakest.

The metric was calculated as follows:

1. For metric number 1 to 14, the average was calculated, as seen in metric 15.
2. For metric number 16, the value of the metric contributes more towards the total metric than all other metric, who have average contribution only. For example if the battery only had 10% of its charge left, the battery metric would influence the total metric by such as state, irrespective of all other metrics, that no traffic would be routed through the device.
3. The total metric is then calculated by adding the metric for the battery to the average.

### Sample: System Total Metrics Table

<table>
<thead>
<tr>
<th>Metric Number</th>
<th>Test</th>
<th>CE</th>
<th>Laptop</th>
<th>Server</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1D</td>
<td>63.66</td>
<td>2.79</td>
<td>5.22</td>
<td>1.12</td>
</tr>
<tr>
<td>2</td>
<td>2D</td>
<td>69.74</td>
<td>4.45</td>
<td>8.90</td>
<td>2.00</td>
</tr>
<tr>
<td>3</td>
<td>3D</td>
<td>76.00</td>
<td>4.36</td>
<td>12.35</td>
<td>1.78</td>
</tr>
<tr>
<td>4</td>
<td>4D</td>
<td>77.04</td>
<td>5.11</td>
<td>14.38</td>
<td>1.88</td>
</tr>
<tr>
<td>5</td>
<td>100000</td>
<td>8.65</td>
<td>0.18</td>
<td>1.01</td>
<td>0.08</td>
</tr>
<tr>
<td>6</td>
<td>800000</td>
<td>76.34</td>
<td>1.69</td>
<td>7.38</td>
<td>0.74</td>
</tr>
<tr>
<td>7</td>
<td>16 files</td>
<td>82.19</td>
<td>26.77</td>
<td>16.46</td>
<td>6.48</td>
</tr>
<tr>
<td>8</td>
<td>16 KB</td>
<td>92.44</td>
<td>4.75</td>
<td>19.26</td>
<td>3.34</td>
</tr>
<tr>
<td>9</td>
<td>1 Metre</td>
<td>84.76</td>
<td>71.08</td>
<td>63.52</td>
<td>32.22</td>
</tr>
<tr>
<td>10</td>
<td>10 Metres</td>
<td>71.08</td>
<td>63.52</td>
<td>63.52</td>
<td>31.00</td>
</tr>
<tr>
<td>11</td>
<td>CS Local</td>
<td>86.71</td>
<td>72.92</td>
<td>87.37</td>
<td>55.87</td>
</tr>
<tr>
<td>12</td>
<td>Remote F</td>
<td>98.93</td>
<td>2.46</td>
<td>1.85</td>
<td>1.12</td>
</tr>
<tr>
<td>13</td>
<td>Net Error %</td>
<td>79.00</td>
<td>0.00</td>
<td>10.00</td>
<td>56.00</td>
</tr>
<tr>
<td>14</td>
<td>Utilisation</td>
<td>60.00</td>
<td>30.00</td>
<td>45.00</td>
<td>10.00</td>
</tr>
<tr>
<td>15</td>
<td>Average</td>
<td>73.32</td>
<td>20.72</td>
<td>25.44</td>
<td>14.55</td>
</tr>
<tr>
<td>16</td>
<td>Battery</td>
<td>50.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>17</td>
<td>Metric</td>
<td>73.32</td>
<td>20.72</td>
<td>25.44</td>
<td>14.55</td>
</tr>
</tbody>
</table>

Table 20 – Example of total system metrics

As discussed in section 5.7, the metrics presented in Table 20 could be used as independently or as a whole system metric. The use of the individual metric would allow for a more precise calculation if the task involved were known to utilise a certain aspect of the devices’ capabilities. For example if the traffic to send consisted of large files, the routing protocol may only be interested in the metric for the file tests and the overall battery metric.
6 Conclusions

During this project, the aim was to develop a benchmark system able to run as a static agent within the MARIAN ad-hoc wireless network framework. The purpose of this product, which I named Benchmark System Software (BSS), was to gather performance metrics from each device by comparing the results of each test against a defined threshold value. This section analyses the achievements of the project, and whether the aims and objectives were met, highlighting any weaknesses.

6.1 Evaluation of Achievement

To evaluate whether the project achieved its aims we must first look at why it was carried out and what results were expected. We can then investigate what achievements were accomplished, presenting the benefits of the research and discussing any novel connections made with respect to the problem domain.

The problems posed by Migas, N., et al (2003b) suggested that a new type of routing protocol was required for on-demand ad-hoc wireless networks. The proposed routing protocol would take into account the processing capabilities of each node on the network. The information expected was a set of metrics that would be calculated by a static agent running on the network nodes themselves. The metrics would then be made available to Mobile Agents to aid in the production of a routing protocol.

Before the problem domain could be addressed, a thorough background research study had to be carried out. The aim of which was to gain an understanding of why current ad-hoc protocols were not suitable for current and future mobile communications, what technologies were suited for the realisation of the proposed design and what benefits would this system bring. The implementation would then be based on these findings.

The research carried out in Chapter 2 of the report highlighted the fact that current On-demand protocols, although efficient in routing traffic, were still wasting valuable resources on mobile devices, by indiscriminately broadcasting route requests to all nodes in search for a suitable route. The term “suitable route” however was based on whether a route was discovered and not on whether the route was suitable for the transmission. These findings suggested that there was indeed a problem, which required further research. The next stage of my research was to identify the technologies available for the implementation of this model. This part of the research highlighted the java programming language as a suitable candidate for the project. This was due to it having characteristics such as, platform-independence secure execution, multithread programming and dynamic class loading. As described in Chapters 2 and 3, there are problems when using java to obtain system performance information that was only accessible via native methods. This problem was partially solved by using the Java Native Interface (JNI) to extract information such as current system utilisation in terms of CPU and memory, network state in terms of quantity and quality of TCP, UDP and IP transmissions. The JNI however would have required more time than was available to enable its functionality to be extended to the PDA. This prompted me to investigate what the most important performance measurements required from the PDA were and how could these be extracted.

Research carried out by various authors, including (Ebert et al 2003 and Chang et al 2000) suggested that the battery strength played the most crucial role in the performance capabilities of the mobile device. This proved to be a challenging task, as there was currently no java API’s or documented native code libraries that were
Further investigation discovered a new tool provided by Microsoft, called Embedded Visual Tools®. These tools are available free of charge from Microsoft and provide support for C++ and Visual Basic programming on all WinCE devices. Using these tools, I implemented a Visual Basic program that monitored the battery and also the system memory. The product of this application was a text-based file that contained the current state of the battery and also the system memory. This file could be read by java to use this information in the creation of a suitable metric. With sufficient technology problems addressed, it was now time to implement the design to extract the required results.

The system designed, as described in Chapter 3 was based around the acquisition of the required performance characteristics of each device by the means of running tests, and then calculating the results of these tests. The results were then compared against all other devices tested and also against a threshold value, which indicated a worst-case scenario. See Chapter 5 for a full description of this process.

Once the system had been designed and implemented, the analyses of the results had to be carried out to identify the following:

- Did the system actually provide the performance information required.
- Could this information be used to create a system metric
- What benefits could be brought about by the system
- Identify novel connections/patterns in the results

**Did the system actually provide the performance information required.**

Analysis of the test results gathered in all sections of Chapter 5 illustrate that, for all tests carried out, I was able to successfully extract performance statistics. These statistics were shown in terms of how each system performed each test in comparison with the other devices tested. Further performance indicators monitoring how long it took the battery strength to depreciate by 1% were run for each test that could be run on the PDA. The results of the battery tests illustrated how certain tests caused the battery strength to depreciate at a far higher rate than others. Table 21 illustrates, for example, that file test 1 creation test for a 1KB file caused the battery to reduce by 1% every 116 seconds whereas the same test creating a larger file of 16KB caused the battery to depreciate at a far higher rate of 62 seconds. This shows creating larger files consumes more battery life.

<table>
<thead>
<tr>
<th>Time taken to reduce Battery by 1%</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test</strong></td>
</tr>
<tr>
<td>1D Bubble</td>
</tr>
<tr>
<td>3D Bubble</td>
</tr>
<tr>
<td>File test 1 - 1KB</td>
</tr>
<tr>
<td>File test 1 - 16KB</td>
</tr>
<tr>
<td>File test 2 - 1KB</td>
</tr>
<tr>
<td>File test 2 - 16KB</td>
</tr>
<tr>
<td>Client-Server Remote 1m</td>
</tr>
<tr>
<td>Client-Server Remote 10m</td>
</tr>
<tr>
<td>Client-Server Local</td>
</tr>
</tbody>
</table>

**Table 21 - Time taken by tests to depreciate the battery by 1%**
Could this information be used to create a system metric
To enable for the creation of a metric, firstly for each test then for the overall system I had to run each test over 20 iterations to gain an average run time for the tests, this was to ensure that any anomalies were identified and removed from the test data. Once this performance data was created, a value was placed on each test. This threshold value was set as the “Worst Case Scenario”, for example the threshold value set for a file download test was set at 100, this was due the slowest device, the PDA taking an average of 91 seconds to download the file. I then compared the results for each of the other devices against this threshold to obtain a comparative metric for the devices. Each section in Chapter 5 details how these metrics are created for their respective tests. Once I obtained a metric for each test, I then collated the information into a total system metric see Table 20 in Section 5.7.3 for a description of these metrics and how they were calculated. This metric was designed to provide a general idea of the performance capabilities of the devices.

What benefits could be brought about by the system
The main benefits provided by the system are the provision of a test system that allows for basic system tests to be performed. The performance information along with the metrics gathered from it could be developed to obtain a more accurate system metric. Referring back to section it is suggested that the use of the individual metric would allow for a more precise calculation if the task involved were known to utilise a certain aspect of the devices’ capabilities. For example if the traffic to send consisted of large files, the routing protocol may only be interested in the metric for the file tests and the overall battery metric. Further more complex ways of analysing the basic performance information could also produce a more accurate metric.

Identify novel connections\patterns in the results
Various patterns have emerged from the analysis of the test results; a brief investigation is carried out below, detailing the anomalies and suggested reasons.

During the comparative tests, that is the tests where all devices were compared against each other to show the differences in their capabilities, it was apparent that the actual specifications of the device did not always match the results. Referring back to section 5.3.2, for the test creating 16 files per size the PDA is only, on average 12.68 times slower than the PC, in contrast to being 50 times slower on the 4D bubble sort test. This could be mainly due to the fact that the PDA has no hard drive to store the files, resulting in the use of RAM, which has a far quicker access time than a hard drive. This pattern is slightly different for the second test, which demands more processing power from the device, as it is creating more files. In this test the PDA is on average 30 times slower than the PC. A possible reason for this is due to the PDA running low on memory as it creates the files in its RAM. For example for the PDA to create 2000 files of 16KB it will use nearly 32Mb of its memory. This would severely impair performance as it has only a total of 64Mb.
Another example is where the difference for the average time taken for the battery to discharge by 1% was during the Client-Server Remote tests. During the tests the battery lost % of its charge every 72 seconds for the 1 metre test and 67 seconds for the 10 metre test. These figures suggest that there is a direct relationship between the transmission range and the use of battery power. As mentioned in section 5.4.1 Ebert et al (2003) also came to this conclusion, in their research into the power consumption of mobile devices in ad-hoc networks. Further observations have been made throughout Chapter 5.

Based on the findings discussed in this Chapter, I feel that the system has provided a valuable prototype to assist in the further development of the MARIAN framework as proposed by Migas, N., et al (2003b). The strengths of the project are demonstrated by the actual achievement of obtaining system performance criteria. This was achieved by the introduction of an automated test system that allowed for continuous tests to be carried out with no user intervention, apart from setting the tests up. The amount of test data created allowed for a detailed analysis of the results, in which I am confident are a fairly accurate depiction of the actual performance of the devices tested. The main weakness in the project was mainly down to the lack of integration of the system into the MARIAN framework. Although the process of transforming the java classes into a static agent to enable it to run under the Grasshopper Agent Platform should be a fairly simple task as both the BSS and Grasshopper are built using java classes. I feel I could have been attempted the integration process to a greater extent.

6.2 Suggestions for future work

This section outlines areas that have been discussed throughout the report where, due to constraints such as time and technological challenges and barriers, certain aspects of the system have had to be developed to “get it working” rather than implementing a single solution across all platforms. Most suggestions are aimed at the further development of the system to run within the MARIAN framework, especially with regards to implementing functions for the PDA that have already been implemented for all PC based systems. Some of the suggestions for future work are:

- There is a need to develop all system to be able to store their test results into a database. This will allow for a more secure environment, which will enable authentication by Mobile Agents. The main obstacle encountered with regards to implementing the database on the PDA is due to the lack of a JDBC driver by Sun Microsystems Inc. However, as declared by Ellis, J (2003) from Sun Microsystems Inc, this technology is due for its immanent release.

- A range of tests could be developed to include further tests that are able to access native methods. This can be achieved by further research into the JNI for use on the PDA as well as the PC-based systems. Types of test could be: battery tests for laptops and PDAs, CPU usage for the PDA, temperature of the mobile devices.
To enable active participation in the MARIAN framework, the BSS system will have to be developed into a static agent or agents. This will allow for communication of the performance metrics to the Mobile Agents. This will also enhance the security of the system as Grasshopper Agent Platform has many built in security features such as Secure Socket Layer communications, and symmetric key encryption.

A more complex process of calculating the system metrics will need to be implemented to allow for a more accurate system metric to be calculated. This could be implemented by the introduction of a specific “metric agent” in the form of a java class, which would have the ability to interrogate the database and file structures to performance information for the creation of a metric.

The BSS system could be developed to run continuously, with the ability to dynamically run and rerun tests that are required to update its system metrics.

6.3 Gantt Chart of project timescales

<table>
<thead>
<tr>
<th>Event</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>June</th>
<th>July</th>
<th>Aug</th>
<th>Sept</th>
<th>Oct</th>
<th>Nov</th>
<th>Dec</th>
<th>Jan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit project Contract</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collate relevant research materials</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find and evaluate available software</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review and critically evaluate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>materials found for literature</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>review (ready for interim report)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plan software Evaluation,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation, and testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(liaising with colleagues)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete Literature review,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>using findings to finalise software</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>development strategies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Continue software development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lifecycle, Evaluation, Implementation, and testing at each stage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implement software solutions within</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>framework of overall research topic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test software solutions against</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>project definitions, evaluate project</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write up project based on findings</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>from literature review and software</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>development, (Draft)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deliver final written project report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>and software solution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Project Viva</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7 References


G. Sinclair, BEng (Hons) Software Engineering, 2003


8 Appendix A
This appendix lists code extracts, tables and diagrams referred to in the report. All comments in the source code listings are shown in Green.

8.1 Details of all tests implemented
Note: Some tests are applicable to the PDA only, denoted by “PDA only”, where others are not suited to the PDA, denoted by “PDA n/a”. All other tests are applicable to all systems. All tests have been implemented in the system although only a subset is discussed in the report. See the enclosed CD for full code listings (.java files).

<table>
<thead>
<tr>
<th>Level</th>
<th>Type</th>
<th>Class name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>Memory, Battery</td>
<td>BSSMonitor.vb</td>
<td>VB application running on PDA to return Current Memory State as well as Battery strength on PDA. This creates a text file allowing the java program access to the data.</td>
</tr>
<tr>
<td>PDA only</td>
<td>Strength (continuous)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel</td>
<td>Memory</td>
<td>BSSKernelTestP1 and BSSKernelTestP2</td>
<td>Time the creation of dummy files of varying sizes. This will test the memory on the PDA as it stores all data in RAM. It will also test the hard drive on all other system.</td>
</tr>
<tr>
<td></td>
<td>(Initial test)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel</td>
<td>CPU</td>
<td>Bubble, BSSbubble2D, BSSbubble3D, BSSbubble4D</td>
<td>There are 4 levels of bubble sort algorithmic tests; these are designed to perform calculations or varying complexity to evaluate the processing capabilities of the CPU.</td>
</tr>
<tr>
<td></td>
<td>(Initial test)</td>
<td>merge</td>
<td>This test runs a less complicated algorithm to test the processing capabilities of the CPU. As with the bubble sorts, the size of the values sorted can be varied.</td>
</tr>
<tr>
<td>Kernel</td>
<td></td>
<td>BSSConnectivityCheck</td>
<td>This test attempts to download a HTML page. If successful it calculates the time for download and shows that there is a connection, if not it shows that there is no internet connection. The HTML page to download can be changed.</td>
</tr>
<tr>
<td>Network</td>
<td>Connectivity</td>
<td>BSSServer, BSSClient, BSSProxy</td>
<td>The combination of these tests allow for checking the Bandwidth and latency of the network by transmitting data between nodes, then calculation the time taken. The tests can also be used to calculate the speed of the Local Network protocol stack by running a client/server test on itself.</td>
</tr>
<tr>
<td></td>
<td>(Continuous)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>Bandwidth</td>
<td>BSSNetworkMon, ProcessMonitor, jvmmonitor.dll, ps.dll</td>
<td>Calls native code via the JNI and the dll’s to calculate current TCP, UDP and IP data rates and errors. Used to monitor the current network state.</td>
</tr>
<tr>
<td></td>
<td>Latency (Continuous)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>Errors</td>
<td>BSSNetworkMon, ProcessMonitor, jvmmonitor.dll, ps.dll</td>
<td>Calls native code via the JNI and the dll’s to calculate current TCP, UDP and IP data rates and errors. Used to monitor the current network state.</td>
</tr>
<tr>
<td>PDA n/a</td>
<td>(Continuous)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td>CPU, Memory,</td>
<td>BSSThreadManager, ProcessMonitor, jvmmonitor.dll, ps.dll</td>
<td>Calls native code via the JNI and the dll’s to obtain current CPU and memory usage per process running in the system as well as total utilisation. This can be used to determine whether the device has available resources.</td>
</tr>
<tr>
<td></td>
<td>Overall</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Utilisation (Continuous)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td>Java Threads</td>
<td>BSSJavaThreadInfo, ProcessMonitor, jvmmonitor.dll, ps.dll</td>
<td>Calls native code via the JNI and the dll’s to obtain the total number of Java threads running and what % of CPU they are utilisating. This will allow for the monitoring of the system to determine the resources used by java threads.</td>
</tr>
<tr>
<td>PDA n/a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group</td>
<td>Operating</td>
<td>BSSGroup</td>
<td>This test obtains operating system, java runtime, user and also network information such as IP address and hostname of local host. These</td>
</tr>
</tbody>
</table>
General Background

This test is for evaluation purposes only; it is designed to periodically run in the background while other tests are running to determine whether it affects the results of the test.

8.2 Client-Server test

8.2.1 Server source code

```java
public class BSSServer implements Runnable {
    private static final int columns = 7;
    private String[] array = new String[rows][columns];
    private boolean isCE;
    private String fileWriter;
    private ArrayList al = new ArrayList();
    private long tBytes = 0;
    private long tTime = 0;

    public BSSServer(int testID_, int distance_, boolean OS) {
        testID = testID_;     // testID indicates what test is being run i.e local or remote
        dist = distance_;    // dist = distance_;    // will need to change if for loops changed
        isCE = OS;
    }

    public void run() {
        if (testID < 1) {
            fileWriter = "BSSServerTestLocal.txt";
        } else {
            fileWriter = "BSSServerTestRemote.txt";
        }

        this.server();
        this.updateFile();
        if (!isCE) {
            this.updateDB();
        }

        private synchronized void server() {
            al.add ("SPort	CPort	tBytes	CPort	ClientIP");  // add file header details
        }
    }

    private synchronized void server() {
        try {
            System.out.println("Listening............");
            ServerSocket sock = new ServerSocket(listenPort);
            Socket sock1 = sock.accept();
            System.out.println("Accepting............");
            System.out.println(sock1.toString());
            remoteIP = String.valueOf(sock1.getInetAddress());
            System.out.println("Remote address : "+remoteIP);
            remotePort = String.valueOf(sock1.getPort());
            System.out.println("Remote port : "+remotePort);
            remoteHostName = String.valueOf(sock1.getInetAddress().getHostName());
            System.out.println("Remote hostName : "+remoteHostName);
        } catch (IOException e) {
            System.out.println("Error in accept() : "+e.getMessage());
        }
    }
}
```

System.out.println("Sending: Welcome to remoteHost:"). It is new Date();
String inp;
BufferedReader is = new BufferedReader(new InputStreamReader(sock1.getInputStream()));
DataOutputStream out = new DataOutputStream(sock1.getOutputStream());
String bites=new String();
String small;
String big;
String bites2=new String();
int count = 0;
small="abcdefghijklmnopqrstuvwxyz";
big="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
for (int z=0;z<20;z++)
{
    bites=bites+small+big;
    long startTime=System.currentTimeMillis();
    out.writeBytes(bites+"n");
    inp = is.readLine();
    long endTime=System.currentTimeMillis();
    long timeTaken;
    timeTaken=endTime-startTime;
    System.out.println("Time to send and receive "+bites.length()+"bytes "+timeTaken+"ms");
    this.fillArray(String.valueOf(bites.length()),String.valueOf(timeTaken),count);        //Insert into array
    count++;
}
for (int y=0;y<32;y++)
{
    bites2=bites2+bites;
    long startTime=System.currentTimeMillis();
    out.writeBytes(bites2+"n");
    inp = is.readLine();
    long endTime=System.currentTimeMillis();
    long timeTaken;
    timeTaken=endTime-startTime;
    System.out.println("Time to send and receive "+bites2.length()+"bytes "+timeTaken+"ms");
    this.fillArray(String.valueOf(bites2.length()),String.valueOf(timeTaken),count);        //Insert into array
    count++;
} al.add ("Totals:\t\t\t"+tlBytes+"\t\t\t"+tlTime+"\t");      // add totals to output file
long latency = (tlBytes / tlTime);
System.out.println("Waiting for client .......");
inp = is.readLine();
System.out.println("From client: " + inp);
inp = is.readLine();
sock1.close();
sock.close();
out.close();
}
catch(InterruptedException err)
{
    System.out.println(err.getMessage());
}
finally
{
    System.out.println("End of the server program");
}
} // end run
private void fillArray(String bytes, String time, int count){
    strArray[count][0] = String.valueOf(listenPort);  // fill the array with the static as well as the runtime values
    strArray[count][1] = remotePort;
    strArray[count][2] = remoteIP;
    strArray[count][3] = remoteHostName;
    strArray[count][4] = bytes;
    strArray[count][5] = time;
    strArray[count][6] = String.valueOf(dist);
al.add (listenPort+"\t"+remotePort+"\t"+remoteIP+"\t"+
private void updateDB()
    if (strArray.length < 1)
        System.out.println("Error in data, please re-run the test\n");
    else
        System.out.println("Updating database .................\n");
        BSSDataTrans insertValues = new BSSDataTrans();
        insertValues.updateClientServerTestInfo(testID,strArray);
    }
private void updateFile(){  // send ArrayList of output to file
    BSSFileTrans ft = new BSSFileTrans();
    al.trimToSize();
    ft.bssFileWriter(fileWriter,al);
} // end class

8.2.2 Client source code

// Client class, this class is used to carry out network tests, in
// particular to sending and receiving data packets to a Server Class
import java.net.*;
import java.io.*;
import java.util.*;
public class BSSClient implements Runnable{
    private static final int port = 1111;
    private String ip = "127.0.0.1"; // default to local host
    public BSSClient(String newIP) {
        if (newIP.length () < 7)          // ensure a non null value passed
            System.out.println("Must enter a remote IP address to connect to......\n");
        else{
            ip = newIP;
        }
    }
    public void run (){
        try
        {
            Socket sock = new Socket(ip,port);
            System.out.println();
            System.out.println("address : " + sock.getInetAddress());
            System.out.println("port : " + sock.getPort());
            System.out.println();
            String inp;
            BufferedReader is = new BufferedReader( new InputStreamReader (sock.getInputStream()));
            DataOutputStream out = new DataOutputStream(sock.getOutputStream());
            for (int z=0;z<52;z++)
            {
                inp = is.readLine();
                out.writeBytes(inp+"\n");
            }
            System.out.println("Send a goodbye");
            out.writeBytes("Thank you\n");
            out.writeBytes("Goodbye\n");
            sock.close();
            is.close();
        }
        catch (UnknownHostException e)
        {
            System.out.println(" Known Host : " + e.getMessage());
        }
        catch (IOException e)
        {
            System.out.println("error I/O : " + e.getMessage());
        }
} // end class
8.3 Four dimensional bubble sort source code

```java
// This class performs a bubble sort on n Integers where n is a value either passed in to the sort function or as a
// command line argument. This is a 4 Dimensional version of the class bubble sort.

public class BSSbubble4D implements Runnable {
    private int nums;
    private long timeTaken = 0;
    private boolean isCE;
    private String fileWriter = "BSSBubbleSort4D.txt";
    private ArrayList al = new ArrayList();

    public BSSbubble4D(int n, boolean OS) {
        nums = n;
        isCE = OS;
        System.out.println("Run 4D Bubble sort on, " + nums + " integers.");
    }

    public void run() {
        this.sort();
        this.updateFile();
        if (!isCE) {
            this.updateDB();
        }
    }

    private void sort() {
        al.add("TypOfSort\t\t\tValue\t\tQuantity\t\tTimeTaken"); // insert header for output file
        long startTime = System.currentTimeMillis();
        System.out.println("Sorting", nums + " elements.");
        Random rn = new Random();
        int[][][] a = new int[nums][nums][nums][nums];
        for (int w = 0; w < nums; w++) {
            for (int x = 0; x < nums; x++) {
                for (int y = 0; y < nums; y++) {
                    for (int z = 0; z < nums; z++) {
                        a[w][x][y][z] = rn.nextInt();
                    }
                }
            }
        }
        int hold; // temporary holding area for swap
        int lastCol = nums - 1; // value of last column
        for (int i = 0; i < nums; i++) {
            for (int j = 0; j < nums; j++) {
                for (int k = 0; k < nums; k++) {
                    for (int l = 0; l < nums; l++) {
                        if (l != lastCol) {
                            if (a[i][j][k][l] > a[i][j][k][l + 1]) {
                                hold = a[i][j][k][l];
                                a[i][j][k][l] = a[i][j][k][l + 1];
                                a[i][j][k][l + 1] = hold;
                            } else if (l == lastCol && k < lastCol && j <= lastCol && i <= lastCol) {
                                if (a[i][j][k][l] > a[i][j][k][l + 1]) {
                                    hold = a[i][j][k][l];
                                    a[i][j][k][l] = a[i][j][k][l + 1];
                                    a[i][j][k][l + 1] = hold;
                                }
                            }
                        } else if (last column in 4th for
                            (l == lastCol && k < lastCol && j <= lastCol && i <= lastCol) {
                                if (a[i][j][k][l] > a[i][j][k][l + 1]) {
                                    hold = a[i][j][k][l];
                                    a[i][j][k][l] = a[i][j][k][l + 1];
                                    a[i][j][k][l + 1] = hold;
                                }
                            } else { // last column in 4th for
                                System.out.println("End of Client program");
                                System.out.println("Press any key to return to the Main menu");
                            }
                        }
                    }
                }
            }
        }
    }
}
```

G. Sinclair, BEng (Hons) Software Engineering, 2003

```java
hold = a[i][j][k][l];
a[i][j][k][l] = a[i][j][k+1][0];
a[i][j][k+1][0] = hold;
}
} else if // last column in 3rd for
  (k == lastCol && l == lastCol && j < lastCol && i <= lastCol) {
  if (a[i][j][k][l] > a[i][j+1][0][0]) {
    hold = a[i][j][k][l];
a[i][j][k][l] = a[i][j+1][0][0];
a[i][j+1][0][0] = hold;
  }
} else if // last column in 2nd for
  (j == lastCol && k == lastCol && l == lastCol && i < lastCol) {
  if (a[i][j][k][l] > a[i+1][0][0][0]) {
    hold = a[i][j][k][l];
a[i][j][k][l] = a[i+1][0][0][0];
a[i+1][0][0][0] = hold;
  }
}
long endTime = System.currentTimeMillis();
timeTaken = endTime-startTime;
al.add ("Bubble 4D"+"t"+"Integer"+"t"+nums+"t"+timeTaken);
System.out.println ( "Time taken to sort " +nums+ " integer values " +timeTaken + "ms");
} // end sort
private void updateDB(){ // update the database
  BSSDataTrans insertValues = new BSSDataTrans();
  insertValues.updateAlgorithmInfo(4,"BubbleSort4D","Integer",nums,timeTaken);
}
private void updateFile(){ // send ArrayList of output to file
  BSSFileTrans ft = new BSSFileTrans();
  ft.bssFileWriter(fileWriter,al);
}
} // end class
```

8.4 Remote file test source code

// BSSConnectivityCheck class, this class is used to carry out network connectivity tests, in particular to sending // and receiving data from a specified Web Server
import java.net.*;
import java.io.*;
import java.util.*;
public class BSSConnectivityCheck implements Runnable{
  protected int port = 80; // can be overwritten by constructor
  protected String host, file;
  protected String theURL;
  protected Writer writer;
  protected BufferedReader reader;
  protected boolean isCE;
  protected String fileWriter = "BSSRemoteFileDownTest.txt";
  protected ArrayList al = new ArrayList();
  protected long timeToConnect;
  protected long timeToDownload;
  protected boolean isConnectionUp = true;
  public BSSConnectivityCheck(String newURL, boolean isCEIn) {
    theURL = newURL;
    isCE = isCEIn;
  }
  public void run (){
    try{
      this.getDetails(theURL);
      long startTimeC=System.currentTimeMillis(); // start timer for connection
      this.connect();
    }
  }
}
long endTimeC=System.currentTimeMillis();

timeToConnect =endTimeC-startTimeC;

long startTimeD=System.currentTimeMillis();        // start timer for download

if  (isConnectionUp)    // Only get the file if the connection is up
this.getFile();

long endTimeD=System.currentTimeMillis();        // only get the file if the connection is up

this.updateFile();         // update file, then database if not Windows CE

if (!isCE)
this.updateDB();

if  (isConnectionUp)   // Only disconnect if the connection is up
// disconnect from server
this.disconnect();

} catch(NoRouteToHostException ex)
{ System.out.println("No connection "+ex);
  isConnectionUp = false;
}

} catch(IOException ex)
{System.out.println("Error in "+this.getClass()+" "+ex);
}

finally {
    System.out.println("End of Remote Test");
}

protected void getDetails(String theURL) throws MalformedURLException{
    URL url = new URL(theURL);       // function to dissect the URL into port, url and file to fetch
    host = url.getHost();
    port = url.getPort();
    if (port == -1)          // check for default port
        port = 80;
    file = url.getFile();
    System.out.println("Host : "+host);
    System.out.println("Port : "+port);
    System.out.println("File : "+file);
}

protected void connect() throws IOException{    // function to get connection to the specified URL
try{
    System.out.println("Connecting to "+host+" on port "+port);  
    Socket socket = new Socket(host,port);
    OutputStream out = socket.getOutputStream();
    writer = new OutputStreamWriter(out,"Latin1");
    InputStream in = socket.getInputStream();
    Reader reader = new InputStreamReader(in, "Latin1");
    this.reader = new BufferedReader(reader);
} catch(NoRouteToHostException ex)
{ System.out.println("No connection "+ex);
  isConnectionUp = false;
}

} catch(UnknownHostException ex)
{ System.out.println("No connection "+ex);
  isConnectionUp = false;
}

}

protected void getFile() throws IOException{    // function to actually receive the file from the server
try{
    System.out.println("Retrieving file "+file);
    writer.write ("GET "+ file + " HTTP/1.0\r\n\n");
    writer.flush();
    PrintWriter console = new PrintWriter(System.out);
    String input;
    while((input = reader.readLine()) != null)
        console.println(input);

    console.println("Transfer complete. "+file);
    writer.flush();
} finally {
    writer.close();
    reader.close();
}

} catch(EOFException ex)
{ System.out.println("EOF read error "+ex);
  isConnectionUp = false;
}

    System.out.println("End of Remote Test");
}

}
8.5 System utilisation test, BSSTskManager source code

```java
import res.*;
public class BSSTskManager implements Runnable {
    private int frequency;
    private long tlMem = 0;
    private long phyMem = 0;
    private long virMem = 0;
    private long freeMem = 0;
    private int cpuPer = 0;  // minus idle process id 0 from total
    private int numPro = 0;
    private int tlThread = 0;
    private int pctMemFree = 0;
    private String t_name;

    public BSSTskManager(int frequencyIn) {
        frequency = frequencyIn;
    }

    public void run() {
        this.runTskManager();
    }

    private synchronized void runTskManager() {
        int i;
        ProcessMonitor man = new ProcessMonitor();
        ProcessInfo[] data;
        System.out.println(man + " Task manager Starting..........................");
        for (;;) {
            data = man/processInfo(frequency);
            System.out.println("BSS Task Manager updating..............");
            for (i=0;i<data.length;i++) {
                phyMem = phyMem + data[i].phys_mem;  // Running TOTAL memory, virtual memory, total threads
                virMem = virMem + data[i].virt_mem;
                tlThread = tlThread + data[i].thread.length;
                if (i > 0) {
                    // do not include idle process, always first in list
                }
            }
        }
    }
}
```
cpuPer = cpuPer + (int)data[i].cpu;
}
if (i >= (data.length - 1)) {
    // capture the total memory, free memory, and number of threads
    tlMem = data[i].total_mem;
    freeMem = tlMem - phyMem;
    numPro = data.length;
    tlMem = tlMem / 1024; // get KB from byte values
    phyMem = phyMem / 1024;
    virMem = virMem / 1024;
    freeMem = freeMem / 1024;
    pctMemFree = (int)freeMem * 100 / (int)tlMem;
    // update database
    this.updateDB(tlMem, phyMem, virMem, freeMem, cpuPer, numPro, tlThread, pctMemFree);
    tlMem = 0; // clear Values
    phyMem = 0;
    virMem = 0;
    freeMem = 0;
    cpuPer = 0;
    numPro = 0;
    tlThread = 0;
}
// for (; ;)

private void updateDB(long tlMem, long phyMem, long virMem, long freeMem, float cpuPer, int numPro, int tlThread, int pctMemFree)
{
    try{
        BSDataTrans insertValues = new BSDataTrans(); // update the database
        System.out.println("\n" + this.getClass() + " \t\tUpdating database........\n");
        insertValues.updateTskManager(tlMem, phyMem, virMem, freeMem, cpuPer, numPro, tlThread, pctMemFree);
    }catch(Exception ex){}
}

8.6 Visual Basic system monitor source code

' Declare Function calls
Declare Function GlobalMemoryStatus Lib "coredll.dll" (ByVal strBuffer As String) As Long
Declare Function GetSystemPowerStatusEx Lib "coredll.dll" (ByVal strBuffer As String) As Boolean
Dim lngMemoryGlobal As Long
Dim bytBatteryGlobal As Long
Dim myDate As Date
Dim dummy As Boolean
Dim state As String
Dim countBattery As Integer
Dim countMemory As Integer
Public Function Show_Error(strNumber As String, strDescription As String, strFormModule As String, strFunctionSub As String) As Boolean
    Dim strMessage As String
    Dim strTitle As String
    Show_Error = False ''''''''''''Error_Handling
    strTitle = "Error"
    strMessage = "Error in ": vbcCrLf
    strMessage = strMessage & vbCrLf
    strMessage = strMessage & "Function or sub ": & strFunctionSub
    strMessage = strMessage & vbCrLf
    strMessage = strMessage & strNumber
    strMessage = strMessage & vbCrLf
    strMessage = strMessage & strDescription
    MsgBox strMessage, vbCritical, strTitle
    Err.Clear
    Show_Error = True
End Function

Function UnitStatistics() ' get values from the dll
On Error Resume Next
Dim strBuffer
Dim lngMemory
Dim bytBattery
strBuffer = LongToString(32) & Space(30)
GlobalMemoryStatus strBuffer
lngMemory = StringToLong(MidB(strBuffer, 5, 4))  'This gives us %age memory used between 0 and 100
If lngMemory <> lngMemoryGlobal Then
    txtCPU.Text = lngMemory
    fileWriterMem
End If
lngMemoryGlobal = lngMemory
strBuffer = Space(24)
GetSystemPowerStatusEx strBuffer
bytBattery = AscB(MidB(strBuffer, 3, 1))               'This gives us %age battery charge between 0 and 100
If bytBattery <> bytBatteryGlobal Then
    txtBattery.Text = bytBattery
    fileWriter
End If
bytBatteryGlobal = bytBattery
If Err.Number <> 0 Then
    dummy = Show_Error(CStr(Err.Number), Err.Description, "BSSMonitor", "UnitStatistics")
End If
End Function

Private Function StringToLong(strValue As String) As Long  ' Convert values to long
Dim strTemp As String
Dim intByte As Long
On Error Resume Next
For intByte = 4 To 1 Step -1
    strTemp = strTemp & Right(“0” & Hex(AscB(MidB(strValue, intByte, 1))), 2)
Next intByte
StringToLong = CLng(“&H” & strTemp)
If Err.Number <> 0 Then
    dummy = Show_Error(CStr(Err.Number), Err.Description, "BSSMonitor", "StringToLong")
End If
End Function

Private Function LongToString(ByVal lngValue As Long) As String  ' convert values returned by dll to long
Dim strTemp
Dim intByte
strTemp = Hex(lngValue)
If strTemp <> “0” Then
    strTemp = Right(“00000000” & strTemp, 8)
For intByte = 7 To 1 Step -2
    LongToString = LongToString & ChrB(CInt(“&H” & Mid(strTemp, intByte, 2)))
Next intByte
Else
    LongToString = ChrB(CInt(“&H00”))
End If
If Err.Number <> 0 Then
    dummy = Show_Error(CStr(Err.Number), Err.Description, "BSSMonitor", "LongToString")
End If
End Function

Private Function StringToLong(ByVal strValue As String) As Long  ' convert values
On Error Resume Next
Dim strTemp
Dim intByte
strTemp = Hex(strValue)
If strTemp <> “0” Then
    strTemp = Right(“00000000” & strTemp, 8)
For intByte = 7 To 1 Step -2
    LongToString = LongToString & ChrB(CInt(“&H” & Mid(strTemp, intByte, 2)))
Next intByte
Else
    LongToString = ChrB(CInt(“&H00”))
End If
If Err.Number <> 0 Then
    dummy = Show_Error(CStr(Err.Number), Err.Description, "BSSMonitor", "LongToString")
End If
End Function
Next intByte
StringToLong = CLng("&H" & strTemp)
If Err.Number <> 0 Then
  dummy = Show_Error(CStr(Err.Number), Err.Description, "BSSMonitor", "StringToLong")
End If
End Function

Private Sub fileWriter()
' write information to file
On Error Resume Next
' Uses "Microsoft CE File System Control 3.0" to your form
Dim myArray(2) As Variant
Dim iCounter As Integer
If (countBattery > 1440) Then
  File1.Open "," + txtFileName.Text, fsModeOutput  ' clear the file if more than 1440 entries
  countBattery = 0
Else
  File1.Open "," + txtFileName.Text, fsModeAppend  ' Append to file
End If
  myArray(0) = txtBattery.Text       'Write 2 lines, Battery % and Time
  myArray(1) = Now
  For iCounter = 0 To 2
    File1.LinePrint myArray(iCounter)
  Next
  File1.Close
  countBattery = countBattery + 1
  txtBattLastUpdate = Now
  If Err.Number <> 0 Then
    dummy = Show_Error(CStr(Err.Number), Err.Description, "BSSMonitor", "fileWriter")
  End If
End Sub

Private Sub fileWriterMem()  ' write to CPU file
On Error Resume Next
Dim myArray(2) As Variant
Dim iCounter As Integer
If (countMemory > 1440) Then
  File2.Open "," + txtFileMemory.Text, fsModeOutput      ' clear the file
  countMemory = 0
Else
  File2.Open "," + txtFileMemory.Text, fsModeAppend  ' Append to file
End If
  myArray(0) = txtCPU.Text          'Write 2 lines, CPU % and time
  myArray(1) = txtBattery.Text
  myArray(2) = Now
  For iCounter = 0 To 2
    File2.LinePrint myArray(iCounter)
  Next
  File2.Close
  txtMemLastUpdated = Now
  countMemory = countMemory + 1
  If Err.Number <> 0 Then
    dummy = Show_Error(CStr(Err.Number), Err.Description, "BSSMonitor", "fileWriterMem")
  End If
End Sub

Private Sub currentState()  'update current state file
On Error Resume Next
state = "BSSCurrentState.txt"
Dim myArray(3) As Variant
Dim iCounter As Integer
File3.Open "," + state, fsModeOutput  ' always overwrite file to have current value only
  myArray(0) = txtCPU.Text           ' insert CPU%, Battery % and time
  myArray(1) = txtBattery.Text
  myArray(2) = Now
  For iCounter = 0 To 3
    File3.LinePrint myArray(iCounter)
  Next
  File3.Close
  If Err.Number <> 0 Then
8.7 Sample configuration file

```plaintext
// Config file: Start up parameters for system.
// All lines starting with "//" will be ignored when loading the file.
// *************** HOW TO ENTER DATA **********************
// To enter a value you must type a keyword to indicate the parameter to pass to the application
// followed by a SPACE then immediately enter the value to be passed.
// IF YOU OMIT THE SPACE THE VALUE WILL NOT BE LOADED.
// ************** RUNNING MULTIPLE OCCURRENCES OF THE SAME TEST FILE**************
// You can run the same test file more than once or more than one type of test to do this
// just enter a new line with the parameter test with a file name.
// i.e. to run the test file Test01.tst you would enter on a new line test Test01.tst
// *************** SETTING THE RUNNING ORDER OF THE TESTS **********************
// The tests are run in order as they appear in this file. To change the order in which they run
```

```plaintext
// Config file: Start up parameters for system.
// All lines starting with "//" will be ignored when loading the file.
// *************** HOW TO ENTER DATA **********************
// To enter a value you must type a keyword to indicate the parameter to pass to the application
// followed by a SPACE then immediately enter the value to be passed.
// IF YOU OMIT THE SPACE THE VALUE WILL NOT BE LOADED.
// ************** RUNNING MULTIPLE OCCURRENCES OF THE SAME TEST FILE**************
// You can run the same test file more than once or more than one type of test to do this
// just enter a new line with the parameter test with a file name.
// i.e. to run the test file Test01.tst you would enter on a new line test Test01.tst
```

```plaintext
// Config file: Start up parameters for system.
// All lines starting with "//" will be ignored when loading the file.
// *************** HOW TO ENTER DATA **********************
// To enter a value you must type a keyword to indicate the parameter to pass to the application
// followed by a SPACE then immediately enter the value to be passed.
// IF YOU OMIT THE SPACE THE VALUE WILL NOT BE LOADED.
// ************** RUNNING MULTIPLE OCCURRENCES OF THE SAME TEST FILE**************
// You can run the same test file more than once or more than one type of test to do this
// just enter a new line with the parameter test with a file name.
// i.e. to run the test file Test01.tst you would enter on a new line test Test01.tst
```

```plaintext
// Config file: Start up parameters for system.
// All lines starting with "//" will be ignored when loading the file.
// *************** HOW TO ENTER DATA **********************
// To enter a value you must type a keyword to indicate the parameter to pass to the application
// followed by a SPACE then immediately enter the value to be passed.
// IF YOU OMIT THE SPACE THE VALUE WILL NOT BE LOADED.
// ************** RUNNING MULTIPLE OCCURRENCES OF THE SAME TEST FILE**************
// You can run the same test file more than once or more than one type of test to do this
// just enter a new line with the parameter test with a file name.
// i.e. to run the test file Test01.tst you would enter on a new line test Test01.tst
```
// all you need to do is change the order in which they appear in this file.
//
// The comments above each value indicate what the test does and the type of value to enter.
//
// All lines starting with "//" are comments and will be ignored by the FileTrans loader class.
//
// ***************************************** OMITTING DATA ******************************************
//
// If you wish to ignore a line you can enter a HASH after the value for example if you want to
// Leave an entry for loading a test file for later use you would enter a the parameter then
// leave a space then enter the test file to run, (see example below):
//
// Example: test level0Test.tst
//
// ***************************************** LOADING ADDITIONAL PARAMETERS AND VALUES **********
//
// If you need to load additional values into the application you will need to create a parameter
// in the class BSSLoad that will store the value entered in this file. You would also need to
// add another if else{ statement in the function setConfigValues to capture the keyword entered
// in this file.
//
// ***************************************** PLACE ALL PARAMETERS TO LOAD AND VALUES BELOW ****
//
// Mode of operation 1 = menu mode, 0 = continuous test mode
mode 1
//
// The test file to load: Note you can enter more than one test file, each will be loaded
// in turn. To ignore the test place a HASH # after the file name. You can also run the same test
// file multiple times by adding the same line as many times as you wish the test file to load.
//
// test level2Test.tst
//
// Enter the system CPU speed in MHz: It is planned to retrieve this value automatically in future
cpu 850
//
// Enter the system RAM: It is planned to retrieve this value automatically in future
ram 512

8.8 Sample test file

File level0Test.tst, ZERO level test. #
//
// This file is a level 0 test file for the BSS application; a level 0 test is a standard test
// with no background TEST processes or continuous tests running.
//
// All tests are suitable for running on all platforms and operating systems.
// The file will be parsed and the values required to run the test will be extracted.
//
// ***************************************** HOW TO ENTER TEST DATA *****************************************
// To enter a value you must type a test keyword to indicate the test followed by a SPACE then
// immediately enter the value to be passed to the test.
//
// * IF YOU OMIT THE SPACE THE VALUE WILL NOT BE LOADED AND THE TEST WILL NOT RUN *
//
// A value of HASH # indicates that the test has not to be run, see example below:
//
// To set the t_bubble test with 10000 integers you would enter t_bubble 10000 on a new line.
//
// If a test has not to be run you must place a HASH # after i.e. t_bubble 10000#
//
// NOTE: Default values are entered with a # at the end of the line. Remove the # to run the
// you may than change the default values as required.
//
//RUNNING MULTIPLE OCCURRENCES OF THE SAME TEST WITH DIFFERENT OR IDENTICAL DATA
// You can run the same test as many times as you wish and with different values; to do this
// just enter a new line with the test name and value.
//
// **** SETTING THE RUNNING ORDER OF THE TESTS **************************
//
// The tests are run in order as they appear in this file. To change the order in which they run
// all you need to do is change the order in which they appear in this file.
//
// The comments above each value indicate what the test does and the type of value to enter.
//
// All lines starting with "//" are comments and will be ignored by the FileTrans loader class.
//
// ******** PLACE ALL TEST DATA AND VALUES BELOW ****************************
//
// DESCRIPTION
// Test ID: This is the ID of the test, each test file type will have
// an ID to identify the type of test, this could be used by the java
// class to run test in a particular way

**testID 1**

//
// The number of Integers to run the 1 dimensional bubble sort on. bubble
// Example: t_bubble 10

<table>
<thead>
<tr>
<th>t_bubble 30000#</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_bubble 10#</td>
</tr>
</tbody>
</table>

//
// The number of Integers to run the 2 dimensional bubble sort on. BSSBubble2D
// Example: t_bubble2D 100

<table>
<thead>
<tr>
<th>t_bubble2D 173#</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_bubble2D 100#</td>
</tr>
</tbody>
</table>

//
// The number of Integers to run the 3 dimensional bubble sort on. BSSBubble3D
// Example: t_bubble3D 12

<table>
<thead>
<tr>
<th>t_bubbleTD 31#</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_bubble3D 12#</td>
</tr>
</tbody>
</table>

//
// The number of Integers to run the 4 dimensional bubble sort on. BSSBubble4D
// Example: t_bubble4D 15

<table>
<thead>
<tr>
<th>t_bubble4D 13#</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_bubble4D 15#</td>
</tr>
</tbody>
</table>

//
// The number of Integers to run the merge sort on. mergeSort
// Example: t_merge 1000

<table>
<thead>
<tr>
<th>t_merge 1600000#</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_merge 1000#</td>
</tr>
</tbody>
</table>

//
// The size of files in KB to create in the file creation test. BSSKernelTestP1
// Example: t_Kernel1 100

<table>
<thead>
<tr>
<th>t_Kernel1 16#</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_Kernel1 100#</td>
</tr>
</tbody>
</table>

//
// The number of files to create in the file creation test. BSSKernelTestP2
// Example: t_Kernel2 100

<table>
<thead>
<tr>
<th>t_Kernel2 2#</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_Kernel2 100#</td>
</tr>
</tbody>
</table>

//
// The String URL and filename to download for network connectivity test
// SSConnectivityCheck
// The String MUST contain the full address including the file to download
// Example: t_url http://www.aib.gov.uk/index.html

<table>
<thead>
<tr>
<th>t_url <a href="http://www.aib.gov.uk/index.html#">http://www.aib.gov.uk/index.html#</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>t_url <a href="http://www.soc.napier.ac.uk/index.html#">http://www.soc.napier.ac.uk/index.html#</a></td>
</tr>
<tr>
<td>t_url <a href="http://www.ebay.co.uk/index.html#">http://www.ebay.co.uk/index.html#</a></td>
</tr>
<tr>
<td>t_url <a href="http://uk.weather.com/weather/local/index.html#">http://uk.weather.com/weather/local/index.html#</a></td>
</tr>
<tr>
<td>t_url <a href="http://java.sun.com/docs/books/tutorial/essential/TOC.html#">http://java.sun.com/docs/books/tutorial/essential/TOC.html#</a></td>
</tr>
<tr>
<td>t_url <a href="http://www.ncl.ac.uk/index.html#">http://www.ncl.ac.uk/index.html#</a></td>
</tr>
<tr>
<td>t_url <a href="http://buchananweb.co.uk/cnds.timetable.html#">http://buchananweb.co.uk/cnds.timetable.html#</a></td>
</tr>
<tr>
<td>t_url <a href="http://www.ed.ac.uk/index.html#">http://www.ed.ac.uk/index.html#</a></td>
</tr>
</tbody>
</table>

//
// Distance between Client and Server on the client server test. BSSServer, BSSClient
// Enter a distance of ZERO 0 to run Local Client server.

//
// The comments above each value indicate what the test does and the type of value to enter.
//
// All lines starting with "//" are comments and will be ignored by the FileTrans loader class.
//
// ******** PLACE ALL TEST DATA AND VALUES BELOW ****************************
To run a remote client server you must also enter an IP address into the _RemoteIP_ value below. See example below of running Remote Client Server.

Example of values to set:
- t_Server 20
- _remoteIP_ 192.166.0.1

**t_Server 1**

- IP address of remote host for client to connect to in Client-Server BSSClient test. i.e. 192.168.0.12
- _remoteIP_ 127.0.0.1#

---

### 8.9 Sample Results file from a test-run

**----- BSS Auto test started: Thu Nov 20 00:54:15 GMT 2003 -----**

**Test file running:** level0Test.tst  
**Operating system:** Windows CE

Battery Life at beginning of test:  
Memory Battery LastUpdated 39 100  
20/11/03 00:54:06

Tests loaded from file:  
TestID 1  
*************** NEW TEST ***************

Test: t_bubble4D scheduled with value of 13

Test: t_bubble4D Starting at Thu Nov 20 00:54:17 GMT 2003  
Battery at Start of test:  
Memory Battery LastUpdated 39 100  
20/11/03 00:54:06

Battery at End of test:  
Memory Battery LastUpdated 47 95  
20/11/03 01:00:08

Test: t_bubble4D Completed at Thu Nov 20 01:00:14 GMT 2003

Type Of Sort  
Value Type  
Quantity  
Time Taken

| Bubble 4D | Integer | 13 | 356042 |

*************** END OF TEST ***************

Time taken to run all tests: 361777ms; in secs = 361

**** Battery and Memory Levels at the end of test ****

Memory Battery LastUpdated 47 95  
20/11/03 01:00:08

**** Battery Monitor Statistics gathered during test ****

Starting Test level0Test.tst at: Thu Nov 20 00:54:16 GMT 2003  
99 20/11/03 00:54:19  
98 20/11/03 00:55:47  
97  
20/11/03 00:56:57
8.10 Email response on JDBC status by Sun Microsystems Inc

From: Jonathan Bruce <Jonathan.Bruce@Sun.COM>
Date: 27 October 2003 03:49
To: Sinclair, Graham 9905883@napier.ac.uk
Subject: Re: JSR 169 JDBC Optional Package for CDC/Foundation Profile

Graham,

We are working towards making the API binaries available in as short a time as possible. At this time, we are preparations to file for our final approval ballot, the second to last stage in the Java Community Process. I anticipate the Final Release, and associated binaries will be available shortly after this, pending a successful ballot.

If you have any further questions, please do not hesitate to contact me on the alias above, or directly by email.

Jonathan Bruce
JDBC Architect & Spec. Lead

Sinclair, Graham wrote:
Hi Jonathan,

My name is Graham Sinclair and I am a final year Honours student at Napier University Computing section. I am doing a project that uses PersonalJava and J2ME to create mobile agents that gather system and network statistical information and store it in a MS Access database, these agents will be running on all Windows and Sun Platforms as well as Pocket PC 2002, (Windows CE.).

I have been looking everywhere for the API source code download for the JDBC Optional Package for CDC/Foundation Profile. It listed as being available on Sun's Website at: http://java.sun.com/products/jdbc/download.html#cdcfp but it is only the specification that is available.

I was reading the Java Community Process JSR 169: JDBC Optional Package for the CDC/Foundation Profile and it has your email address listed as a contact. Could you please let me know if this download is available and if it is I would be grateful if you could give me the details of how to get it.

Any assistance in this matter would be greatly appreciated.

Regards
Graham Sinclair

8.11 Code extracts from scenario in section 4.4

Function to insert the Client-Server test data into the database

```java
public synchronized void updateClientServerTestInfo(int testNum, String[][] strArray) {
    this.getConnection();
    String updateCS = "";
    String tempString = "";
    String s1 = this.getDateTime();
    String deleteOld = "DELETE * FROM clientServer WHERE testID = " + testNum; // clear old test data
    try {
        // stmt.executeUpdate(deleteOld); // uncomment this line if you wish to purge table before insert
        for (int y = 0; y < strArray.length; y++) {
            updateCS = "insert into clientServer " + "(testID,serverPort,clientPort,clientIP,remoteHostName,bytes,timeTaken,distancebetweenHosts,lastUpdated) values(" + "'" + testNum + "','" + Integer.valueOf(strArray[y][0]) + "','" + Integer.valueOf(strArray[y][1]) + "','" + strArray[y][2] + "','" + strArray[y][3] + "','" + strArray[y][4] + "','" + Integer.valueOf(strArray[y][5]) + "','" + Integer.valueOf(strArray[y][6]) + "');";
            stmt.executeUpdate(updateCS);
        }
    } // end for
    this.cleanup();
```
catch(SQLException ex) {
    System.out.println("- Exception caught -
" + ex.getMessage());
}
} // end

Function to get data, then pass to display function to display results of the database to screen
public synchronized void displayData(String query) {
    // Generic class to retrieve and display records from the database
    // select statement is passed in as a parameter
    this.getConnection();
    System.out.println("Database Query in progress............\n");
    try {
        ResultSet rs = stmt.executeQuery(query);
        ResultSetMetaData rsmd = rs.getMetaData();
        this.display(rs, rsmd);
        this.cleanUp();
    } catch(SQLException ex) {
        System.out.println("- Exception caught in getClientServerTest --
" + ex.getMessage());
    }
} // end display

private synchronized void display(ResultSet rs, ResultSetMetaData rsmd){
    // Display data retrieved from database
    try {
        int numberOfColumns = rsmd.getColumnCount();
        while (rs.next()) {
            for (int i = 1; i <=
numberOfColumns; i++) {
                String colName = rsmd.getColumnName(i);
                System.out.print(colName + " Value :");
                String s = rs.getString(i);
                System.out.print(s +"\n");
            }
        }
    } catch(SQLException ex) {
        System.out.println("- Exception caught -
" + ex.getMessage());
    }
} // end display

Extract from purge Database Function, (full code has option for purging each table).
public synchronized void purgeDatabase(String option){
    // Purge the database by clearing all tables, this will not clear the config
    System.out.println("Purging database tables in progress............\n");
    this.getConnection();
    try {
        if(option.equals("4")){
            stmt.executeUpdate(purgeClientServer);                           // Purge Client server
            System.out.println("Purging .....
purgeClientServer\n" + System.out.println("Purging database tables in progress............\n");
            this.getConnection();
            try {
                if(option.equals("all")){
                    stmt.executeUpdate(purgeAlgorithm);
                    stmt.executeUpdate(purgeGroupInfo);
                    stmt.executeUpdate(purgeKernel);
                    stmt.executeUpdate(purgeClientServer);
                    stmt.executeUpdate(purgeAppThread);
                    stmt.executeUpdate(purgeAppProcess);
                    stmt.executeUpdate(purgeNetState);
                    stmt.executeUpdate(purgeThreadInfo);
                    stmt.executeUpdate(purgetaskMngr);
                    stmt.executeUpdate(purgeThreadTotals);
                    System.out.println("Purging ALL tables ................\n");
                } else {
                    System.out.println("Cannot purge data - Invalid option entered\n");
                }
            } catch(SQLException ex) {
                System.out.println("- Exception caught -
" + ex.getMessage());
            }
        this.cleanUp();
    } // close connection
    } catch(SQLException ex) {
        System.out.println("- Exception caught -
" + ex.getMessage());
    }
} // end purgeDatabase
8.12 Database tables
Appendix B  - User Instructions

9.1 CD Contents
The CD has the following directory structure.

- BSS
  - All java source code and compiled class files for the Benchmark Software System.
  - Microsoft Access 2000® database, BSS.mdb
  - Sample .tst files used to automate tests i.e. level0Test.tst
  - RES folder: Containing all java source code as well as DLL files jvmmonitor.dll and ps.dll. These files are required to access native code via the JNI.

- CE
  - Containing all files for the PDA, includes subfolder BSS
  - Templates folder; This includes templates to allow for automated tests to be run without having to connect to a PC to download new files.
  - File BSS.lnk, which is the shortcut that will run the code on the PDA.

- Csource
  - All C and C++ source code used to compile the DLL’s mentioned above. For reference only, not used in system.

- DOCS
  - Copy of completed dissertation document; Projectv1.4.doc.
  - Excel document with charts used in report; Charts.xls.

- Java
  - Contains the Java v1.4.1 runtime environment (JRE) and Software Development Kit (SDK) set-up files; j2re-1_4_1_01-windows-i586-i.exe and j2sdk-1_4_1_01-windows-i586.exe.
  - Also contains a batch file that will set up the java environment; this may need to be edited to reflect the settings of your system

- TESTS
  - All test data files created during the test phase of the project.
  - Templates folder: Containing templates that can be used to automate tests.

- VB
  - APPLICATION folder; containing the set-up program for the Visual Basic battery monitor for the PDA.
  - Source Code folder: containing the Visual Basic source code.

9.2 Installation of the Benchmark System Software (BSS)
The BSS software will run on any operating system that has a suitable Java runtime environment installed. For a PC based system this is any version of java from v1.2.1 onwards. For the PDA, which is running Pocket PC 2002, (subset of Windows CE®) the Jeode® java runtime environment must be installed, (equivalent to java v1.3.1).
The java code supplied on the CD has been compiled using java v1.2.1; this is to ensure that it will run on all systems. If you already have any version of java from v1.2.1 upwards installed you will not have to install java.
Java
If you do not have Java installed, you will find Java v1.4.1 Software Development Kit and Java Runtime Environment for Windows on the CD in the java folder. To install these, double click on the required file, and then follow the install instructions.

Set Java Environment
You will need to set the class PATH for Java environment to point to the version of java you wish to use. A sample batch file, BSS Java Version 1.4.1.bat for use on Windows systems is provided, in the Java folder on the CD. If you install the Java v1.4.1 software also provided and except the default install directory, you will not have to set this class PATH. You will however have to change the directory settings in the file, unless you copy the BSS folder on the CD to the root of your C:\ drive. Instructions are provided in the batch file.

Install BSS software
1. Copy the complete BSS folder from the CD onto your PC. As mentioned above, if you install it on the root of your C:\ drive and you have installed the java version provided, you will not have to change the batch file.
2. Copy the batch file to your PC; this can go anywhere as long as it points to the correct version of java and also the directory you copied the BSS folder to.

Set up database
To use the database functionality, you must set up an ODBC connection as follows, (Windows-based systems):

1. Browse to the control panel in Windows, and then choose Administrative tools.
2. Choose Data Sources (ODBC); the screen below will appear.
3. Choose either the system or the user tab.

4. Select Add; you will then see the screen below.
5. Select Driver do Microsoft Access (*.mbd), then select Finish.
6. You will now see the screen below.
7. Enter a Data Source Name, ("BSS" has been inserted in the example below)
8. Enter a Description, (optional).
9. Choose the Select button.
10. Browse to the location of the database **BSS.mdb**, select the database, and then choose OK.
11. Select the advanced button.
12. Enter **"static"** in the Login name: field.
13. Enter **"password"** in the password: field.
14. Select OK, you will be taken back the screen below.
15. Press OK, and then OK again to complete the installation.
16. You will now be able to connect to the database when running the BSS on a PC based system, (not on the PDA).

**Note:** You can now run the system if wish to run the system in menu, (see section 9.4).
If you wish to run the system in automatic mode you will have to follow the instructions in section 9.3.
9.2.2 PDA Installation

To install the BSS on the PDA you will need to connect it to your PC via a USB connection and also have Microsoft ActiveSync and the Jeode® java runtime environment installed.

BSS Software

1. Double click the CE folder on the CD.
2. Select all files in the folder, including the BSS and Templates subfolders.
3. Select Edit > Copy, to copy all files.
4. Use Windows explorer to browse to the mobile device, then choose My Pocket PC folder.
5. Paste the files in the My Pocket PC folder on the PDA.
6. This will copy all files required to run the BSS system.

Visual Basic Battery Monitor Software

1. Connect the PDA to the PC that you have the CD in.
2. Browse to the VB\Application\CD1 folder.
3. Double click the setup.exe file, and then follow the instructions
4. Once installed, Select Start > Programs > BSSBatteryMonitor to start the application.
5. See Figure 20 in section 4.2.5.1 for a sample of the application running. Click here if you are reading the instructions online.

See section 9.4.2 if you wish to run the system in menu mode, or section 9.3 if you wish to configure the system to run automated tests.

9.3 Configuring the software

Once you have completed the relevant section, 9.2.1 for a PC and 9.2.2 for a PDA, you are now ready to choose how to run the software.

9.3.1 Set run mode

As described during the report, the software can run in one of two modes: automatic mode or menu mode. Changing the value assigned to the mode section in the BSSConfig.ini file sets the start-up mode of the system. The mode value is set to 1 as default. This will start the system in menu mode. If you wish to run the system in test mode you will need to change the mode value in the BSSConfig.ini file to 0.

Section 8.7 shows a sample BSSConfig.ini file. If you are reading this document online, click here, to go to the line in the sample file that you need to change. The code extract from the file below shows which line you need to change.

```csharp
// Mode of operation 1 = menu mode, 0 = continuous \ test mode
mode 1
```

Once you have installed the BSS, the BSSConfig.ini file can be found in the BSS folder on the PC systems and in the My Pocket PC folder in the PDA. This file must be in this location for the system to load the values from it.
If you have chosen to run the system in menu mode, (this is set as default) you can run the system. If you have chosen to run the system in automatic test mode, you will now need to configure a test file to load the tests. See section 9.3.2 below.

9.3.2 Load and configure .tst file to automate test run

When the system starts, the first thing that it does is check the BSSConfig.ini file. At present, this file has two main purposes, these are: To set the mode of start-up, (See section 9.3.1 above) and to inform the system what test file to load, if the mode is set to automatic. The instructions below show how to configure the test files. The following test files have been created:

- level0Test.tst
- level0Test1.tst
- level1Test.tst
- level2Test.tst
- level2Test2.tst

An example of each of these files can be found in the BSS folder on the PC and the My Pocket PC on the PDA.

Each test file is basically a template that can be changed to allow for the automation of the tests, which can be set within the file. The reason for different test files is due to the fact that certain JNI tests will not work on the PDA, the level1Test.tst file and also the file level2Test.tst contain entries for JNI tests. You can however use any file to load the tests on any system as long as you choose not to load an unsuitable test for the system you are running, from the file. The procedure for loading the test files is explained below:

1. Decide what tests you wish to run, and then choose the appropriate .tst file
2. Open the BSSConfig.ini file and enter the name of the test file you will be using, the code extract from the BSSConfig.ini file shows an example of configuring the system to load the test level2Test.tst file. To run a test, remove the HASH # entered anywhere on the same line as the filename. To stop the test from running place a HASH # anywhere on the same line as the filename.

```
// The test file to load: Note you can enter more than one test file, each will be loaded
// in turn. To ignore the test place a HASH # after the file name. You can also run the same test
// file multiple times by adding the same line as many times as you wish the test file to load.
//
// test level2Test.tst
```

3. You will now need to open the test file you are loading; the file level2Test.tst is shown in this example.
4. Find the entry for the test you wish to run, leave a space after the test name then enter the value to be passed into the test.
5. If there is a HASH # entered on same line of the test, remove it. The code extract from the file level2Test.tst shows how to run the t_bubble2D test with a parameter of 173.
Note: Full instructions on how to use these files are included in the file.

9.3.4 Using templates to automate test run

Template folders have been to aid in the frequent running of different tests, and also for running multiple tests. These template folders can be found on the CD in the \TESTS\Templates folder. These template folders each contain a BSSConfig.ini file and the corresponding .tst files; these files are already configured to run a standard set of tests. The name of the folder holding the files describes the purpose of the template. Figure 38 shows a list of available templates

To run a test using a template, copy and paste the BSSConfig.ini and the .tst files into the BSS folder on a PC or the My Pocket PC folder on the PDA.

Figure 38 – List of Templates for automated tests
9.4 Running the BSS System

Before running the application, you need to decide what mode you wish to start it up in, and if it is automatic mode you will need to configure a test file, (section 9.3).

9.4.1 Running on PC Based system

Follow the instructions below to run the system:

1. If you have configured it, double click the BSS Java Version 1.4.1.bat file to start the command line console. Otherwise, you will need to start a command prompt window, and set the java environment.
2. At the command prompt, type “java BSSLoad”, and then press return.
   a. If you do not wish to use the database, type any parameter after BSSLoad; this will run the system in CE mode.
3. If you chose to run in menu mode, you will see the log in prompt, (see below).
4. If you chose to run in automatic mode the system will run the tests then close.

5. Enter a valid username and press return. A valid name is “bill” the valid usernames and passwords are held in the database table users.
6. Enter a valid a password and press return. A valid password is “password”.
7. The system will now authenticate your username and password and if valid will take you to the Main menu, (see below), if not the system will automatically close.

8. You can now run tests, view results and update the database by browsing through the menu system. This is an intuitive, menu driven process.
9. If in the database mode, the results of each test will be stored in the database.
10. If you chose automatic test mode the complete set of the results will be stored in a text file in the format “BSS-level0Test201103-00723.txt”. Section 4.3 describes the format of the results files produced by the system.

9.4.2 Running on PDA

If you wish to use the menu system you can start the system immediately after the installation process, otherwise you will have to follow the instructions in section 9.3. The instructions below show how to run the system.

1. Choose Start > Programs > File Explorer > My Device
2. Select the BSS shortcut that was copied from the CD. This will start the BSS using the Jeode environment.
3. If you are running in menu mode, the same menu system as the PC, without the login and database functions will be displayed.
4. Or if you chose to run in automatic test mode, the test will run to completion then save the results to a results file. See Section 4.3 for a description of the contents of this results file.

Note: All test results files created from running the tests will be created in the MY Pocket PC folder, when viewing it from a connected PC. This is the same folder as the My Device folder mentioned in step 1 above.

9.5 Example of an automated remote client server test

In this test one machine will act as the Sever and the other the client. The instructions below assume that you have already set up and configured the BSS on each of the machines. The location the templates are stored in, and the location to place the files to run are different for the PC and the PDA. However all file settings are identical.

9.5.1 Server set-up

1. If the server is a PC, browse to the “\TESTS\Templates\cs remote template server” folder on the CD.
2. Or if it is a PDA browse to “\TESTS\Templates\CE single test Templates\csR\server”.
3. Copy and paste the files (BSSConfig.ini and level0Test.tst) from this folder to the BSS folder on your PC Or the My Pocket PC folder on the PDA.
4. You will now need to set the approx distance between the two machines in the level0Test.tst file. This is set as default to 10, representing 10 metres. The code extract below, from level0Test.tst shows the setting, (comments are in green)

```csharp
// Distance between Client and Server on the client server test. BSSServer, BSSClient
// Enter a distance of ZERO 0 to run Local Client server.
// To run a remote client server you must also enter an IP address into
// The t_RemoteIP value below. See example below of running Remote Client Server
// Example of values to set:
// t_Server 20
// t_remoteIP 192.166.0.1

t_Server 10
```

5. For each time you want to run the test, you can do one of two things.

```csharp
// Distance between Client and Server on the client server test. BSSServer, BSSClient
// Enter a distance of ZERO 0 to run Local Client server.
// To run a remote client server you must also enter an IP address into
// The t_RemoteIP value below. See example below of running Remote Client Server
// Example of values to set:
// t_Server 20
// t_remoteIP 192.166.0.1

t_Server 10
```
a. Place an entry as described above, in the level0Test.tst, for each instance of the test to run
b. Place an entry of level0Test.tst in the BSSConfig.ini for each time the system has to load the .tst file. See step 6.

6. The code extract below is from the BSSConfig.ini file. This shows six entries for the test file level0Test.tst. This will run the test six times.

7. To run a test you must type the word test, leave a space, the type the name of the .tst file to load, (see below)

```
// The test file to load:  Note you can enter more than one test file, each will be loaded
// in turn.  To ignore the test place a HASH # after the file name. You can also run the same test
// file multiple times by adding the same line as many times as you wish the test file to load.
//
//
test level0Test.tst
//
test level0Test.tst
//
test level0Test.tst
//
test level0Test.tst
//
test level0Test.tst
```

9.5.2 Client set-up

1. If the Client is a PC, browse to the “\TESTS\Templates\cs remote template client” folder on the CD.
2. Or if it is a PDA browse to “\TESTS\Templates\CE single test Templates\csR\client”.
3. Copy and paste the files (BSSConfig.ini and level0Test.tst) from this folder to the BSS folder on your PC and the My Pocket PC folder on the PDA.
4. As this is a client, you will need to specify the Server address in the level0Test.tst file. The code extract below, from the level0Test.tst file will cause the client to connect to a server at IP address 192.168.0.4.
5. You will need to change this to the IP address of your PC running the Server test. Ensure that you leave a space between the test name t_remoteIP and the IP address.

```
//
// IP address of remote host for client to connect to in Client-Server BSSClient
t// test.  i.e. 192.168.0.12
#t_remoteIP 192.168.0.4
```

6. You will have to configure the BSSConfig.ini file to run exactly the same way as in step 7 of section 9.5.1. Ensure that the number times to run the test are the same for client and server systems
7. If you have configured the server, you are now ready to run the test. See Section 9.5.3
9.5.3 Running the test

Before you run the test you will need to ensure that both the client and server are able to communicate with each other; by wireless on fixed network.

To run the test:

1. Start BSS system on the machine running the server
   a. The Server will listen on port 1111
2. Start the BSS system on the machine running the client
   a. The client will connect to the machine running the server
3. The Server and Client instances will now run the number of times you entered the test in the .tst file. They will synchronize, ensuring that the Server instance starts first and listens for the client.
4. Once the test is complete, the results containing the data and the start and end times of the tests, will be on the machine running the server instance. The results will be in the form of a results file as described in section 4.3.
5. The client instance will also have a results file, this will only have the start and end times of the tests. This will also be in a results file as described in section 4.3.