Framework for Integration of Heterogeneous Mobile Devices

by

Francisco Cantú de la Garza

Monterrey, N.L., April 2012
FRAMEWORK FOR INTEGRATION OF HETEROGENEOUS MOBILE DEVICES

by

Francisco Cantú de la Garza

Thesis

Submitted to the School of Engineering and Information Technologies

at the

Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey

in partial fulfillment of the requirements for the degree of

Master of Science

in

Information Technology

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Monterrey, N.L., April 2012
Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey
School of Engineering and Information Technologies

Thesis committee members recommend the present thesis of Francisco Cantú de la Garza to be accepted in partial fulfillment of the requirements for the degree of Master of Science in Information Technology.

Thesis Committee

José Raúl Pérez Cázares
Ph.D.
Thesis advisor

Mario Isidro de la Fuente
Martínez, M.Sc.
Examiner

Raúl Valente Ramírez
Velarde, Ph.D.
Examiner

Lorena Guadalupe Gómez
Martínez, Ph.D.
Director of the Master of Science in Information Technology

April 2012
With the increasing number of mobile devices in the market and the use of wireless technologies, there has been a problem in the integration of all these devices to form collaborative networks in order to process and share information. This paper presents the design and implementation of PARMI, a socket-based remote method invocation system that enables heterogeneous systems integration. PARMI specification is aimed to provide guidance on the implementation of a distributed system on mobile devices; the problem faced by engineers working on a multi-platform environment is the lack of integration between different systems. Mobile devices are required to collect data in the field of work in different areas of industry and commerce. If we can implement a technology that enables integration of heterogeneous mobile devices, it may contribute to the formation of a "mobile distributed system". There are mobile computing systems supporting wireless media communication, but their networks are limited to specific hardware platforms. This investigation proposes a framework to enable the integration between mobile devices and desktop computers. A communication model is proposed, which can be adapted to different platforms, such as mobile phones based on Windows, Android, Symbian, or even any device that supports TCP/IP wireless communication. Multiple wireless technologies were evaluated during this research; leading to select one that best serves the requirements of our prototype. The PARMI based model is a framework independent of programming languages and OS, based on UDP sockets for fast integration of heterogeneous platforms.
# Table of Contents

Chapter 1. Introduction ........................................................................ 1
  1.1. Background information .................................................... 2
  1.2. Problem definition ............................................................ 6
  1.3. Purpose of the thesis project .............................................. 11
  1.4. Hypothesis of the thesis project ......................................... 12
  1.5. Scope of the thesis project ................................................ 12

Chapter 2. Theoretical basis ............................................................... 14
  2.1. Management of small and medium enterprises .................. 14
  2.2. Distributed systems .......................................................... 15
  2.3. Programming languages ................................................... 16
  2.4. Mobile devices ............................................................... 20
  2.5. Operating systems for mobile devices .............................. 24
  2.6. Conclusions .................................................................. 25

Chapter 3. Mobile ecosystem research .............................................. 29
  3.1. Technologies for high performance mobile communications ... 29
  3.2. Development frameworks ................................................ 40
  3.3. Mobile application development tools ............................... 50
  3.4. Conclusions .................................................................. 53

Chapter 4. Evaluation of middleware and existing distributed systems ... 56
  4.1. Remote Procedure Call Middleware ................................. 56
  4.2. Existing mobile distributed systems .................................... 66
  4.3. Conclusions .................................................................. 81

Chapter 5. PARMI specification ......................................................... 86
  5.1. PARMI overview ............................................................. 86
  5.2. Device model ................................................................. 87
  5.3. Mobile distributed system model ....................................... 88
  5.4. PARMI interpreter .......................................................... 96
  5.5. Conclusions ................................................................ 100

Chapter 6. System prototype ............................................................... 103
  6.1. PARMI library implementation ....................................... 103
  6.2. System implementation .................................................... 108
  6.3. Deployment of the application ......................................... 114
  6.4. Conclusions ................................................................ 122

Chapter 7. Bibliography .................................................................. 123
Table 1. Features and requirements for Java .......................................................... 17
Table 2. Features and requirements for C# ................................................................ 19
Table 3. Features and requirements for C++ .............................................................. 20
Table 4. Mobile OS processors .................................................................................. 25
Table 5. Features of programming languages ............................................................. 26
Table 6. Mobile device evaluation ............................................................................. 28
Table 7. Features for Android .................................................................................. 42
Table 8. Features for Microsoft Visual Studio ........................................................... 43
Table 9. Features for wxWidgets .............................................................................. 44
Table 10. Features for Mono .................................................................................... 45
Table 11. Features for PhoneGap .............................................................................. 47
Table 12. Features for J2ME MIPI ............................................................................ 48
Table 13. Features for Qt ......................................................................................... 49
Table 14. Comparison of different wireless technologies ............................................ 54
Table 15. Cross-platform development frameworks .................................................. 55
Table 16. Features supported by CORBA .................................................................. 58
Table 17. Features supported by COM .................................................................... 59
Table 18. Features supported by Java RMI ............................................................... 60
Table 19. Features supported by SOAP .................................................................... 62
Table 20. Features supported by Enterprise Java Beans ............................................. 64
Table 21. Features supported by Sockets .................................................................. 66
Table 22. Characteristics of the system .................................................................... 68
Table 23. Characteristics of the system .................................................................... 69
Table 24. Characteristics of the system .................................................................... 71
Table 25. Characteristics of the system .................................................................... 73
Table 26. Characteristics of the system .................................................................... 75
Table 27. Characteristics of the system .................................................................... 78
Table 28. Characteristics of the system .................................................................... 81
Table 29. Middleware technologies comparison ....................................................... 83
Table 30. Comparison of existing mobile distributed systems ................................... 84
Table 31. FunctionX() skeleton and stub. ................................................................. 98
Table 32. PARMI functions implemented in server .................................................... 110
Table 33. PARMI functions implemented in client .................................................... 110
List of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Business process</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Lack of information analysis</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Information analysis</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Distributed system at a truck company</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>Client offices</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>Gas station</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>Mobile distributed system</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>QT system over Wi-Fi</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>Java RMI system based on Bluetooth</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>System based on MS.Net</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>Challenging system integration</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>PARMI-based application stack</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>PARMI message data flow</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>PARMI framework development stages</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>Distributed system host</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>Java application stack</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>C# application stack</td>
<td>18</td>
</tr>
<tr>
<td>18</td>
<td>C++ application stack</td>
<td>19</td>
</tr>
<tr>
<td>19</td>
<td>Pantech P7000</td>
<td>21</td>
</tr>
<tr>
<td>20</td>
<td>Motorola WX290</td>
<td>22</td>
</tr>
<tr>
<td>21</td>
<td>Amazon Kindle Fire tablet</td>
<td>23</td>
</tr>
<tr>
<td>22</td>
<td>BIP-6000</td>
<td>23</td>
</tr>
<tr>
<td>23</td>
<td>Apple iPhone 4</td>
<td>24</td>
</tr>
<tr>
<td>24</td>
<td>HTC HD2</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>Features of programming languages</td>
<td>27</td>
</tr>
<tr>
<td>26</td>
<td>A GSM architecture</td>
<td>30</td>
</tr>
<tr>
<td>27</td>
<td>A GPRS architecture</td>
<td>32</td>
</tr>
<tr>
<td>28</td>
<td>3G network architecture model</td>
<td>35</td>
</tr>
<tr>
<td>29</td>
<td>Bluetooth stack</td>
<td>37</td>
</tr>
<tr>
<td>30</td>
<td>A Bluetooth network</td>
<td>37</td>
</tr>
<tr>
<td>31</td>
<td>A Wi-Fi architecture</td>
<td>39</td>
</tr>
<tr>
<td>32</td>
<td>Android SDK architecture</td>
<td>41</td>
</tr>
<tr>
<td>33</td>
<td>Visual Studio SDK architecture</td>
<td>42</td>
</tr>
<tr>
<td>34</td>
<td>wxWidgets SDK</td>
<td>44</td>
</tr>
<tr>
<td>35</td>
<td>Mono architecture</td>
<td>45</td>
</tr>
<tr>
<td>36</td>
<td>PhoneGap Architecture</td>
<td>46</td>
</tr>
<tr>
<td>37</td>
<td>J2ME-MIPI architecture</td>
<td>47</td>
</tr>
<tr>
<td>38</td>
<td>QT build process</td>
<td>48</td>
</tr>
<tr>
<td>39</td>
<td>QT Platform [41]</td>
<td>49</td>
</tr>
<tr>
<td>40</td>
<td>Android emulator</td>
<td>50</td>
</tr>
<tr>
<td>41</td>
<td>Windows Mobile/Phone emulator</td>
<td>52</td>
</tr>
<tr>
<td>42</td>
<td>Symbian device emulator</td>
<td>53</td>
</tr>
<tr>
<td>43</td>
<td>CORBA Communication Model</td>
<td>57</td>
</tr>
<tr>
<td>44</td>
<td>COM software architecture</td>
<td>58</td>
</tr>
</tbody>
</table>
Figure 91. Java PARMI library class diagram ................................................................. 104
Figure 92. C# PARMI library class diagram ................................................................. 105
Figure 93. PARMI interpreter modular view ................................................................. 106
Figure 94. PARMI function owned by the server .......................................................... 107
Figure 95. PARMI function owned by the client ......................................................... 107
Figure 96. Prototype system architecture ................................................................... 108
Figure 97. Prototype PARMI interface ....................................................................... 109
Figure 98. Classes implemented in the client ............................................................... 111
Figure 99. Client’s U.I. ............................................................................................... 112
Figure 100. Classes implemented in the server .......................................................... 113
Figure 101. Server’s U.I. .......................................................................................... 114
Figure 102. Client call stack for remote method request .............................................. 115
Figure 103. Server call stack for remote method request ............................................ 115
Figure 104. Server call stack for remote method callback .......................................... 115
Figure 105. Client call stack for remote method callback .......................................... 116
Figure 106. Kindle Fire Properties Hardware ............................................................. 117
Figure 107. Kindle Fire Properties Details .................................................................. 118
Figure 108. android_winusb.inf ............................................................................... 119
Figure 109. adb_usb.ini ......................................................................................... 119
Figure 110. Adb success confirmation ....................................................................... 119
Figure 111. Kindle Fire notification bar .................................................................... 120
Figure 112. Kindle Fire settings menu ....................................................................... 121
Figure 113. Kindle Fire allow installation of applications ........................................... 121
Figure 114. PARMI based distributed system for a transportation company. .............. 122
Acknowledgments

Thanks to my wife for her extraordinary support to complete this research.

Thanks to Professor José Raúl Pérez Cázares for guiding me to define my own thesis theme, giving me the opportunity to create my own vision of this research project, but at the same time complementing the gaps in it.
Chapter 1. Introduction

This research aims to provide a tool for small and medium businesses that need to collect field data and exchange, to be processed and stored in information systems. Small and medium enterprises lack of tools for managing the business efficiently. The problem posed by the lack of information for the owners of such businesses has a direct impact on business costs. Several case studies argue that technological modernization in enterprises aims to increase productivity and costs up to 50 percent. Medway Plastics case study is presented in [1]; it shows that the implementation of an Information Technology (IT) solution simplifies management causing money savings and increased productivity by 50 percent. Another case study is the company Peaks [2]; it demonstrates that online collaboration enabled the company reduce the costs of consultants also by 50 percent. Figure 1 shows the three main stages of the business process, for small and medium enterprises:

- Decision making: In this stage, the stakeholders define the rules and steps for the operation of the company.
- Operations: It is the execution of the business; which requires investment and supervision from the same stakeholders.
- Results: This is where the product or service is complete as a result of the execution stage. In this stage, the stakeholders expect to get earnings based on the investment done in the operation stage. The earnings are increased when the operation is executed proficiently, resulting in less investment required as well.

![Figure 1. Business process](image-url)
1.1. Background information

The context of this research is the lack of information on the state of the business; limiting companies to conduct an analysis of the results obtained from the operation. Figure 2 shows how the stakeholders are not able to make decisions when there is not a previous analysis on the state of the business.

![Figure 2. Lack of information analysis](image)

In order to get feedback on the current state of a business, information needs to be collected; such is generated both during the operation and results of operation. Information systems provide the ability to store and process the information generated in the company. In Figure 3 the business process, shown in Figure 1, is improved by adding 2 more phases:

- Information systems: During this phase, all information generated from the operations and results phases is stored in a system.
- Information analysis: The information stored in the system is used to generate reports and projections, useful for making decisions on how to manage the business.

In companies where operations are not controlled by computer, but physically, it is necessary to deploy a mobile information system, which allows employees to acquire and transmit information.
Development of mobile information systems is at an immature stage, compared to the development of desktop systems. This is because wireless communication technologies and mobile devices are at a stage of development and improvement; every day new tasks are given and invented for mobile devices. Mobile information systems are based on specific platforms and high-cost wireless technologies, systems based on technologies such as .Net Compact Framework (CF), Java ME (Micro Edition), supporting wireless technologies as 3G (third Generation) or SMS (Short Message Service). Using platform specific software makes difficult for the system to keep up with the increasing and rapid emergence of mobile devices with more processing power and new features.

The context of this project is a truck company with need for an integrated mobile distributed system. Trucks need to process and log information relevant to the operation and management of the company. Management of this company has identified 3 main places where information needs to be collected and exchanged. First place is the headquarters of the organization; this is the place where trucks start and end all trips. Figure 4 shows the layout of the distributed system implemented for trucks to synchronize with the management office. Trucks are equipped with a mobile device which is used for multiple tasks at the main office:

- Provide instructions to the operators when a new trip is programmed.
- Get previous trip logs, including:
  - Items loaded/unloaded at client’s stations.
  - Truck mileage.
  - Truck fuel level.
  - Fuel loaded at the gas station.
  - Operator identification.

Figure 5 shows the layout of the distributed system implemented at the clients’ quarters. Trucks use this system to get information on the items loaded and unloaded, as well as instructions on where each items needs to be delivered.
Figure 4. Distributed system at a truck company

Figure 5. Client offices
Figure 6 shows the layout of the distributed system implemented at the gas station where trucks load fuel. Trucks use this system to collect details on the amount of fuel loaded and its cost.

A distributed system model based on mobile devices, as shown in Figure 7, should support different models of mobile devices and must have the ability to support low-cost wireless technologies to achieve a successful implementation on Small and Medium Enterprises (SMEs).
Monitoring and collecting data directly from the operation of the business enables administration for decision making based on real, concrete information. It is necessary to define a mobile distributed system model that allows the use of different hardware platforms, facilitating the integration of multiple systems. It is also important to empower the mobile system to use a low cost communication media, which will facilitate the deployment in small and medium enterprises; companies whose operation does not require a long-range communication system, for data collection.

Mobile distributed systems have been developed to allow using different hardware platforms, but based on high-cost wireless technologies [3] or with high performance requirements [4]. DORMIA (Distributed Object Replicas for Mobile Information Access) system [5] proposes a solution to the diversity of mobile devices through the use of Java and CORBA (Common Object Request Broker Architecture) IDL (Interface Definition Language). The system described in [6] proposes a similar solution based on Java ME; Chen [7] proposes a model based on Java RMI (Remote Method Invocation) over Bluetooth, but the fact that the systems use RMI, limits its deployment to devices that support a Java Virtual Machine (JVM).

With the development of a system for collecting, storing and processing information that is generated from the business, and through new information technologies, companies will be enabled to keep track of their internal activities. Mobile devices must be synchronized with master databases, according to the needs of customers. It is necessary to consider a mobile distributed system model that is based on wireless communication technology, at low cost, fast processing time and allowing synchronization between a server and its clients.

Since clients in the system are based on mobile devices, it is necessary to provide a framework that allows clients to invoke methods or processes on the server. Given the diversity of mobile devices and technologies that each one supports, it is necessary to consider a communication model that is adaptable to different platforms, such as mobile phones based on Windows Mobile/Phone, Android, Symbian, or even any device that supports wireless communication, such as Bluetooth, Wi-Fi (Wireless Local Area Network) or 3G.

1.2. Problem definition

There is not an optimal model for designing and implementing a distributed system based on heterogeneous mobile devices for the administration of small and medium enterprises; the development of hardware and software for compact devices is not under a standard, compared with desktop systems; making system integration difficult. The main factors influencing this issue are described below.

Existing mobile distributed systems allow access to remote information from portable computers; systems that promise high availability, efficiency and support for heterogeneous environments, as [5] using CORBA and Java IDL to define interfaces and allow the same to be independent of platforms and operating systems, based on TCP (Transmission Control Protocol) / IP (Internet Protocol) as a communication medium. Distributed object-oriented development for wireless environments has become an important value in the context of distributed computing and collaboration systems. Java RMI [7] has been proposed as one of the key methods for developing distributed computing systems over heterogeneous wireless systems, specifically in the Bluetooth environment.
The use of Java RMI for implementation of distributed systems in heterogeneous wireless environments is promising; however it requires devices to support a JVM. In addition, deploying applications for mobile and desktop computers require different development frameworks. The development framework for mobile Java applications is based on Java ME; which cannot be used for desktop applications. There are solutions based on CORBA over Bluetooth; Mutlu [8] shows how Bluetooth can be used in distributed systems based in CORBA without the need for TCP / IP, presenting ideas for developing applications that can make extensive use of mature CORBA specifications.

With the introduction of web services on the desktop environment, the same technology has been extended to mobile software development frameworks as Java ME and .Net CF, enabling a wide range of development opportunities. The development of web service based systems is at an early stage, allowing new ideas to implement this technology in various sectors of distributed computing.

Current technologies that implement mobile distributed systems are tied to specific platforms, either via programming language, wireless technology or OS (Operating System). CORBA service invocations are based on a client-server paradigm and allow the interaction of heterogeneous environments, increasing scalability and portability of the system. However, the specification of CORBA interfaces is limited. There is no information on what is done when a method is invoked, pre- or post-conditions, or dependencies between the invocations, which is a disadvantage for this technology.

We have identified limitations in the development of distributed systems for mobile devices:
- Systems whose development frameworks allow them to be deployed on multiple operating systems, but the same are limited by the communication medium. Figure 8 shows a distributed system developed in QT and based on Wi-Fi communication; QT is a framework for developing cross-platform GUI (Graphical User Interface) applications. Figure 9 shows a system based on Java RMI over Bluetooth.

![Figure 8. QT system over Wi-Fi](image-url)
Systems designed to support scalability and portability, based on a single operating system or development framework (Figure 10).

Figure 9. Java RMI system based on Bluetooth
There is not a development framework that enables the implementation of a distributed system over real heterogeneous devices; which makes challenging system integration and portability (Figure 11). QT allows GUI applications to be deployed on heterogeneous platforms, but is its programming language is C++; while Java RMI allows remote procedure calls across heterogeneous platforms, but its deployment is limited to devices supporting a JVM. CORBA allows remote procedure calls across platforms and is supported by Java and C++, but its deployment on mobile operating systems is limited.
Figure 11. Challenging system integration
1.3. Purpose of the thesis project

Existing commercial systems are designed to work in a single administrative domain, usually based on a single development framework or operating system. The availability of links between many different devices will enable mobile users to access information and services anywhere [9].

The purpose of this project is to design a framework called PARMI (Platform Agnostic Remote Method Invocation) for development of mobile distributed systems based in a client-server paradigm with a simple communication protocol. The framework includes the specification of communication protocol, the application stack (Figure 12) for using this protocol to support remote method invocation or services through a wireless medium. Using Bluetooth as the communication channel is evaluated, because it supports point to point connections between different mobile devices; Wi-Fi is also considered for the development of the prototype. Other wireless technologies are evaluated for defining a list of media which will allow the creation of an optimal model.

![Figure 12. PARMI-based application stack](image)

The PARMI specification provides a generic design that allows deployment of a remote method invocation service in different platforms, with one constraint: these platforms should support wireless communication based on Wi-Fi or Bluetooth (Figure 13). The model is independent of programming languages and technologies; however, a prototype for a Pocket PC-style device will be developed.
The proposed framework allows improvements in the area of security and session management. The connection between devices and the server needs to be secured and session oriented, to ensure correct synchronization between devices.

1.4. Hypothesis of the thesis project

It is possible to design a framework for the development of mobile distributed systems, which facilitates remote method invocation across multiple platforms and hardware, using a technology that allows wireless communication and secure connections.

1.5. Scope of the thesis project

The process which will allow the development of the PARMI framework (Figure 14) is defined by these steps:

1. An investigation of existing wireless technologies/frameworks is needed, to be able to design the new framework.
2. Analyze the advantages and disadvantages of the wireless technologies, candidates for the framework. Different combinations of technologies should be evaluated.
3. Comparison between existing designs of distributed systems.
4. Evaluation of current development tools for mobile applications on different operating systems.
5. Define a model that provides the best quality of service.

![Diagram of PARMI framework development stages]

An interpreter for defining common interfaces between clients and a server is implemented. A XML (Extensible Markup Language)-type standard is proposed for describing interfaces that can be taken by the interpreter to generate a stub and a skeleton. The interpreter reduces the risk of inconsistencies while invoking remote methods.

The design of the framework for mobile distributed systems is validated by implementing a prototype system, showing the capabilities and advantages of the model.
Chapter 2. Theoretical basis

This chapter starts with a brief overview of the challenge present for small and medium enterprises while keeping track of their day to day operations, converting the information for decision making into an important resource. Information systems are presented as a tool to mitigate this challenge for small and medium enterprises.

Programming languages and the devices supporting them are studied. Three programming languages are evaluated, Java, Microsoft .Net C# and C++; analyzing their features in order to extract advantages and disadvantages comparing their features with each other.

At the end of the chapter, mobile operating systems and the platforms that support them are analyzed and compared.

2.1. Management of small and medium enterprises

SMEs have a great importance in developing countries. In Mexico, 98% of businesses are small or medium size. Although SMEs are able to provide a product or service with quality, they have a deficiency in the management of its resources. 35% of the problems in SMEs reside in the need for credit, but the other 65% is related to administration [10]. Traditionally, the resource management of an SME is seeking to control economic factors, material and human, but there is a new resource, information.

The growth of a company increases the complexity of its operations, making its control more difficult and converting the information for decision making into an important resource [11]; consequently taking relevant information technologies, will allow storing, processing and transforming the data generated from the operation of the business through computers. When the use of information technology is maximized, the accuracy of the information on the current state of business is increased and therefore decisions will be more assertive.

Investing money in computer equipment is a complex task for SMEs who are not experts in information systems; consequently this operation is performed rarely [11]. Providing the computer equipment with suitable software allows an effective use of information technology to control operations within the company, which translates to making decisions with less risk.

Decision Support Systems (DSS) are designed to manage and control the operations of the company, serving as a support tool for making different types of decisions (strategic, tactical and operational). The information they handle is:

- Access to sources and data stores.
- Financial projections.
- Decision alternatives.

The benefits in an SME due to the implementation of an information system are independent of the size of the organization; however, entrepreneurs of SMEs do not know this or believe that such tools are very expensive. Some of the benefits, according to a survey of more than 200 U.S. organizations are listed below [12]:

14
• Quality of decision-making process.
• Effective communication.
• Cost reduction.
• Reduction of time.
• Increased productivity.
• Increased customer and employee satisfaction.

2.2. Distributed systems

Management systems for companies are usually based on distributed systems, which use technology to communicate between multiple computers. A distributed system consists of components running on different computers to interact with each other through a network [13]. The decision to build distributed systems is based on user requirements, especially scalability, heterogeneity and fault tolerance. We say that a system evolves, when new components are added to it; when this happens, the oldest components may not be compatible with the new technology, programming language or operating system, so it is necessary to maintain a heterogeneous system.

![Distributed system host diagram]

Figure 15. Distributed system host

A distributed system consists of the following elements:

• Host: A computer running components that are part of a distributed system (Figure 15).
• Middleware: The layer between the operating system and application. It is the tool that solves the problems of heterogeneity and distribution.
• Network: A distributed system is a set of autonomous hosts that connect through a computer network.

An important property of a distributed system is the ability to hide the complexity of the system due to the distribution of components; this property is identified as "transparency." For example, access transparency, refers to the interfaces for local and remote communication being the same; the location transparency refers to a client not needing to know the physical location of the component to be used;
transparency of migration means that a component could be relocated without customers noticing this; and replication transparency is when users can access a service regardless of whether this is provided by the master component or a replica of it.

2.3. Programming languages

2.3.1. Java

Three main concepts should be defined when working with Java [14]:

- **Java programming language.** The language in which all Java the applications are written. Programs are converted to byte codes that can be executed on any platform running a JVM.
- **JVM.** It is implemented in the form of a software program which takes a Java application and executes it on the PC.
- **Java platform.** It is a set of classes installed with Java Framework. These classes are the core for the development of Java applications.

The programming language has syntax similar to that of C, with the intention of being powerful, but at the same time it avoids complex features as in other programming languages like C++.

The JVM is a powerful tool allowing a Java program to be executed on multiple platforms; the only limitation is that every system must have support for a JVM.

Java has been improved with the implementation of Just in Time (JIT), which allows Java applications to be converted to native platform machine language on the fly and improves execution speed.

2.3.1.1 Application stack

Figure 16 shows the stack of a Java application.

Figure 16. Java application stack
2.3.1.2 Features and requirements

Table 1 lists the features and requirements for Java.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating objects</td>
<td>Yes</td>
</tr>
<tr>
<td>Creating structs</td>
<td>No</td>
</tr>
<tr>
<td>Macro definition</td>
<td>No</td>
</tr>
<tr>
<td>Middleware required</td>
<td>JVM</td>
</tr>
<tr>
<td>Middleware supported</td>
<td>RMI, CORBA, Sockets, Web Services, SOAP (Simple Object Access Protocol), JDBC (Java Database Connectivity)</td>
</tr>
<tr>
<td>OS supported</td>
<td>Windows Mobile/Phone Android™, Brew® Mobile Platform, Symbian™</td>
</tr>
</tbody>
</table>

Table 1. Features and requirements for Java

2.3.2. Microsoft .Net (C#)

C# is a programming language which makes use of the Microsoft .Net Framework. .Net Framework is composed of a runtime environment Common Language Runtime (CLR)) and a set of class libraries. C# is a combination of C++ and Java languages, its key features include [15]:

- Component orientation. C# makes component building easy, including component-oriented attributes like events and constructs.
- One-stop coding. Declaration is done only once and there is no need to use header files or interfaces.
- Type safety. C# ensures a variable can be accessed only by its associated type, making an encapsulation.
- Unified type system. All types (primitive, structs, arrays, enums) share the same basic functionality, like its ability to be converted to string, serialized or being stored in a collection.
- Automatic memory management. This solves issues seen in other languages, like memory leaks and dangling pointers.
- Leveraging of CLR. Mechanisms like memory management model, exception handling and type system, fit with .Net CLR.

CLR improves runtime interactivity between programs, portability, and security, simplifying development due to its cross-language integration.
2.3.2.1 Application stack

Figure 17 shows the stack of a C# application.

![C# Application Stack Diagram]

Figure 17. C# application stack

2.3.2.2 Features and requirements

Table 2 lists the features and requirements for C#.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating objects</td>
<td>Yes</td>
</tr>
<tr>
<td>Creating structs</td>
<td>Yes</td>
</tr>
<tr>
<td>Macro definition</td>
<td>No</td>
</tr>
<tr>
<td>Middleware required</td>
<td>.Net Framework</td>
</tr>
<tr>
<td>Middleware supported</td>
<td>CORBA, Sockets, Web Services, SOAP, JDBC</td>
</tr>
<tr>
<td>OS supported</td>
<td>Windows Mobile/Phone</td>
</tr>
</tbody>
</table>
On contrary to C# and Java, which are languages that depend on frameworks and platforms, C++ is independent of any platform and framework, with the ability to be adapted and compiled to several different environments. C++ is based on C language, but with additional features like object oriented programming.

One of the relevant properties of C++ for our project is the ability of this language to be used in every platform. C++ programs are compiled and assembled to specific computer architectures, which might sound not too useful if we want to develop cross-platform applications; however, with the help of platform-specific libraries, C++ can be of great benefit. This concept will be explored in the next section.

### 2.3.3.1 Application stack

Figure 18 shows the stack of a C++ application.

![C++ application stack](image)

Figure 18. C++ application stack

### 2.3.3.2 Features and requirements

Table 3 lists the features and requirements for C++.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Supported</th>
</tr>
</thead>
</table>

19
Table 3. Features and requirements for C++

<table>
<thead>
<tr>
<th>Feature</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating objects</td>
<td>Yes</td>
</tr>
<tr>
<td>Creating structs</td>
<td>Yes</td>
</tr>
<tr>
<td>Macro definition</td>
<td>Yes</td>
</tr>
<tr>
<td>Middleware required</td>
<td>No</td>
</tr>
<tr>
<td>Middleware supported</td>
<td>CORBA, Sockets, Web Services, SOAP</td>
</tr>
<tr>
<td>OS supported</td>
<td>Windows Mobile/Phone Android™ Brew® Mobile Platform Symbian™</td>
</tr>
</tbody>
</table>

2.4. Mobile devices

Palm pilot is the first mobile device, released to market in 1996. As Palm devices showed great success, other manufacturers started to bid in the mobile world. The first mobile devices supported only simple applications as schedulers, calendars and to-do lists. As mobile technology evolved, devices started to handle more complex tasks. State of the art in mobile devices, its hardware and operating system, will be covered in this chapter.

As the cellular phones became popular, they became more complex in terms of hardware, operating system, applications and peripheral devices. In Europe, more than 75 percent of the population owns a cellular phone [16]. In this section we will present a classification of mobile devices according functionality to the end user. Mobile devices are categorized by [17] in three branches:

- Mobile phones
- PDAs
- Smartphones

Mobile devices are highly heterogeneous, there are nearly 10,000 different models, and the list increases every month. Development of powerful mobile devices makes it possible to use them as an all-in-one solution for private and business applications.

2.4.1. Mobile phones

Mobile phones are divided in 2 categories:

- Web-enabled phones
  - Advantage: Primary use of this type of devices is voice, however new devices include WAP (Wireless Application Protocol), text messaging and wireless technologies for enabling Internet access, which simplify the Internet applications deployment. WAP allows the user to use the modem for voice and data transfers. One more advantage of these devices is the power consumption capabilities, since it does not provide processing capabilities as Smartphone’s do, it makes easier to save power and have a battery last longer.
Limitation: Devices provide a small display (4 to 12 lines of text), 12-button keypad, which makes Internet surfing difficult. However, specific purpose applications can be an alternative for accessing data; in example, stock quotes or flight updates.

- Two-way pagers.
  - Advantage: Devices allow users to exchange short text messages. A special feature in these devices is their capability to be connected all the time to the network.
  - Limitation: Most of applications running in these devices are loaded by manufacturer and are not upgradeable. Devices are excellent for specific purpose “paging” applications, and do not support sophisticated systems.

Models

Pantech P7000 provides text and touch capabilities with a qwerty keypad with basic calling and messenger services (Figure 19). Specifications: Java, Bluetooth, camera, USB (Universal Serial Bus).

![Pantech P7000](image)

Figure 19. Pantech P7000

Motorola WX290 (Figure 20) features an mp3 player and FM (Frequency Modulation) radio with 2GB of internal memory. Specifications: Bluetooth, WAP 2.0, Camera, Radio.
2.4.2. PDAs

PDAs (Personal Digital Assistant) provide below characteristics:

- **Advantage:** Main features are its touch screen and wireless capabilities. These devices are also known as Palm or Pocket PC's. Target audience is corporate users, however as the devices become more popular and cheaper, their use is being shifted to personal purposes.
- **Limitation:** As companies deploy line-of-business applications, the memory and processor capabilities for these devices have become an issue. Applications deployed, demand requirements set by JVM or other support applications. Windows Mobile/Phone operating system requires a powerful device, in terms of memory, processing and power consumption capabilities. These devices are different from Smartphone's, mostly because they do not include a modem, for voice and data services.

**Models**

Kindle Fire has a 7" multi-touch display with IPS (In-Plane Switching) technology and anti-reflective treatment, 1024 x 600 pixel resolution at 169 ppi, 16 million colors; 8GB internal (approximately 6GB available for user content); cloud storage and cloud web surfing technologies; it supports Wi-Fi networks or hotspots that use 802.11b, 802.11g, 802.11n, or enterprise networks with support for WEP (Wired Equivalent Privacy), WPA (Wi-Fi Protected Access) and WPA2 security using password authentication; does not support connecting to ad-hoc (or peer-to-peer) Wi-Fi networks; It has a USB 2.0 micro-B connector and a 3.5 mm stereo audio jack, with top-mounted stereo speakers. This device runs Android operating system (Figure 21).
Bluebird Soft's BIP-6000 [18] offers 3.5-inch VGA (Video Graphics Array) touch screen, 3-megapixel camera, 1D/2D barcode scanner, RFID (Radio Frequency Identification) and IrDA (Infrared Data Association) readers, WiFi, GPS (Global Positioning System), and HDSPA (High Speed Downlink Packet Access). This device runs Windows Mobile/Phone operating system (Figure 22).

2.4.3. Smartphones

Smartphones provide below characteristics:

- **Advantage:** These devices are capable of providing calling services as well as running local applications. These devices provide multiple services in a single chip, as they evolve they are able to provide the same set of services than a personal computer. Modern Smartphone’s provide WVGA (Wide Video Graphics Array) displays with touch screen capabilities and 5 to 12 megapixel cameras.

- **Limitation:** Support of applications is limited to their storage, processing and display capabilities. Size of the device is set by user experience and battery requirements.

**Models**

iPhone 4 features USB, 5MP (Megapixel) camera, touch screen, Bluetooth, mp3 player, Wi-Fi and modem for voice and data. Based on iPhone OS 4.0 operating system (Figure 23).
HTC HD2 is a Windows Mobile/Phone 6.5 featuring Wi-Fi, GPS, One Exchange email service, 5MP camera, audio and video player, Bluetooth, G sensor, voice and data modem (Figure 24).

2.5. Operating systems for mobile devices

Operating systems or a software platform on the hardware device itself have become popular on mobile environments; in order to provide a shell to run different software applications and programs on the wireless device, the device should be controlled by an OS [19]. There are multiple mobile OSs running on multiple hardware platforms on the market; each hardware platform with different characteristics; all this mix of software and hardware makes the deployment of applications constrained to a specific set of devices.

Table 4 shows most common operating systems supported by each mobile processor currently in the market.
### Mobile OS Processors

<table>
<thead>
<tr>
<th>Processor Revision</th>
<th>OS Support</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>QUALCOMM</strong></td>
<td></td>
</tr>
<tr>
<td>QSD8672 (Dual-CPUs)</td>
<td>Android™, Brew® Mobile Platform, Windows Mobile/Phone</td>
</tr>
<tr>
<td>MSM8x60 (Dual-CPUs)</td>
<td>Android™, Brew® Mobile Platform, Windows Mobile/Phone</td>
</tr>
<tr>
<td>MSM8x55</td>
<td>Android™, Brew® Mobile Platform, Windows Mobile/Phone</td>
</tr>
<tr>
<td>MSM7x30</td>
<td>Android™, Brew® Mobile Platform, Windows Mobile/Phone</td>
</tr>
<tr>
<td>QSD8x50</td>
<td>Android™, Brew® Mobile Platform, Windows Mobile/Phone</td>
</tr>
<tr>
<td><strong>TEXAS INSTRUMENTS</strong></td>
<td></td>
</tr>
<tr>
<td>OMAP4</td>
<td>Symbian™, Windows Mobile/Phone, Linux® (Android, MiMo)</td>
</tr>
<tr>
<td>OMAP3</td>
<td>Linux, Windows Mobile/Phone, Open Handset Alliance, Android, Symbian™</td>
</tr>
<tr>
<td><strong>NVIDIA</strong></td>
<td></td>
</tr>
<tr>
<td>Tegra 650</td>
<td>Windows Mobile/Phone</td>
</tr>
<tr>
<td>Tegra 600</td>
<td>Windows Mobile/Phone</td>
</tr>
</tbody>
</table>

Table 4. Mobile OS processors

### 2.6. Conclusions

Table 5 shows the differences between the three programming languages. The three languages provide similar features in terms of object definition and library integration. However, every language is targeted for different platforms; Java and C# programs require specific middleware technologies in order
to execute the application on a platform; this is with the purpose of supporting multiple platforms; C++ language is not tied to a middleware and it is compiled for a specific platform.

RMI middleware is limited to Java language, however, web services, sockets, CORBA and SOAP are targeted for cross-platform environments; supporting different operating systems and hardware. Most of the mobile operating systems support JVM and some of them support .Net Framework, but mobile device manufacturers might decide not to include support for it in specific devices, making systems integration difficult.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Java</th>
<th>C#</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating objects</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Creating structs</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Macro definition</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Middleware required</td>
<td></td>
<td>JVM</td>
<td>.Net Framework</td>
</tr>
<tr>
<td>RMI</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JDBC</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOAP</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>CORBA</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Sockets</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Web Services</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Windows</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mobile/Phone</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Android™</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brew® Mobile Platform</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symbian™</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5. Features of programming languages

Figure 25 describes the application stack for each programming language. Understanding the stack, help us identify the gaps in the existing infrastructure, which prevent deployment of mobile distributed systems over heterogeneous devices. The gap identified in the application stacks in the figure below is the lack of a common layer, which controls the communication flow between these.
Table 6 shows a relation between existing mobile devices and the middleware and programming languages supported by each. The state of the art on mobile phones, limits the development of applications for these devices, making difficult deploying a distributed system on them. However, PDAs and smartphones meet all the requirements for implementation of mobile distributed systems. PDAs usually do not make use of a modem and are targeted to specific tasks. For this research, the prototype uses a PDA, specifically based on the characteristics of the Kindle Fire device.
<table>
<thead>
<tr>
<th>Device classification</th>
<th>Mobile phone</th>
<th>PDA</th>
<th>Smartphone</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pantech P7000</td>
<td>Motorola WX290</td>
<td>Kindle Fire</td>
</tr>
<tr>
<td>Operating system</td>
<td>Proprietary</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Android</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Windows Mobile/Phone</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Symbian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming language</td>
<td>Java</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>C++ / C objective</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>C#</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Wireless link</td>
<td>GSM</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>GPRS</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Bluetooth</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>WLAN</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>3G</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Power consumption</td>
<td>Active [hours]</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Idle [hours]</td>
<td>336</td>
<td>620</td>
</tr>
</tbody>
</table>

Table 6. Mobile device evaluation
This chapter starts with the evaluation of wireless technologies commonly supported by mobile devices, including GSM (Global System for Mobile Communications), GPRS (General Packet Radio Service), Bluetooth, Wi-Fi, and 3G. Main points covered for each technology include overview, performance, disadvantages, services offered and benefits. These points allow comparing the technologies with each other.

SDKs (Software Development Kit) for development of mobile applications are explored to identify the best candidate for the implementation of the prototype. Frameworks analyzed and compared include Android SDK, Microsoft Visual Studio, wxWidgets, Mono, PhoneGap, QT and J2ME (Java 2 Platform Micro Edition)-MIPI (Mobile Industry Processor Interface). Data points are covered for each SDK include an overview of the framework, and support for programming languages, operating systems, middleware and native applications.

At the end of the chapter, IDEs (Integrated Development Environment) are exercised to get a Hello World application running. The purpose of this exercise is to be able to determine how difficult is to create an application, adding libraries and using middleware from these operating systems:

- Android.
- Windows Mobile/Phone.
- Symbian.

3.1. Technologies for high performance mobile communications

In this section we look at the state of the art in wireless technologies supported by commercial mobile devices. Wireless communication technologies play an important part on the implementation of a mobile distributed system. The performance, scalability and portability of a system are led by the communication layer. Wireless technologies which have been integrated to mobile devices include:

- GSM.
- GPRS/EDGE (Enhanced Data rates for GSM Evolution).
- 3G/4G.
- Wi-Fi.
- Bluetooth.

Access to these technologies from a programming perspective is provided in the form of APIs (Application Programming Interface). The communication layer on a distributed system is shown in the form of an Operating System Adaptation Layer.

3.1.1. GSM

GSM is designed to provide a range of services and features not available on analogue cellular networks, allowing voice and data transfers. It [20] is based on a series of contiguous radio cells which provide complete coverage of the service area and allow the subscriber to operate anywhere within it,
making easier to handle calls from cell to cell. It also incorporates worldwide roaming in other GSM networks.

A unique feature of GSM compared to older analog systems is the SMS. SMS is a bidirectional service for sending short alphanumerical (up to 160 bytes) messages in a store-and-forward fashion.

GSM allows digital data to be transported, both synchronous and asynchronous, to or from an ISDN (Integrated Services Digital Network) terminal. It provides two services for data transfer:
- Transparent service: It has a fixed delay but no guarantee of data integrity.
- Non-transparent service: It guarantees data integrity through an Automatic Repeat request (ARQ) mechanism, but with a variable delay.

### 3.1.1.1 Architecture

The architecture of a GSM network (Figure 26) can be fragmented in these components [21]:
- Mobile station: It is the device carried by the subscriber.
- Base station subsystem: It controls the radio link with the mobile station.
- Network switching subsystem: It provides the core services, by executing the switching of calls between the mobile and other fixed or mobile network users.

![Figure 26. A GSM architecture](image-url)
3.1.1.2 Performance

A GSM network user is able to send data at speeds of up to 9.6 Kbps [22]. The data rates supported by GSM are 300 bps, 600 bps, 1200 bps, 2400 bps, and 9600 bps. Allowing for high audio quality and, uninterrupted phone calls while moving at high speeds, like from cars or trains.

Due to its chip-card, users are able to access networks from different providers [23]. Network access is secured by authentication via chip-card and PIN. GSM network handles localization of users.

3.1.1.3 Disadvantages

GSM networks do not support encryption of user data, allowing for possible abuse of private data. Usage of this technology generates electromagnetic radiation.

3.1.1.4 Services

GSM offers several types of connections [23]:
- Voice.
- Data.
- Short message service.
- Multi-service options (combination of basic services).

These connections are possible due to 3 network sub-domains:
- Bearer services: Allow transmission of appropriate signals between access points.
- Teleservices: Terminal equipment functions to communicate with other subscribers.
- Supplementary services: These services modify or supplement basic telecommunications services and are offered together or in association with basic telecommunications services.

GSM data applications include:
- Mobile Internet access.
- Remote database access.
- E-mail send and receive.
- File transfer and remote LAN (Local Area Network) access.

3.1.1.5 Benefits

The major benefit of GSM is that users can access it at any time from anywhere. There is no need to rely on accessing a POTs (Plain Old Telephone Service) line. GSM is a digital technology, and is inherently more secure. As part of the GSM standard, both, data and voice transmissions are encrypted for transmission across the network.

GSM interacts with the ISDN to speed up call setup time. ISDN uses a rate adaptation technique to convert the slower GSM data up to 64Kbps. The call set-up time in a GSM network ranges from 3 to 4 seconds, as compared with dialing into a V.34 modem, which ranges from 35 to 40 seconds.
3.1.2. GPRS

GPRS network is an evolution of GSM, emerged with the advent of Internet and WAP based services, as a packet data service. It provides efficient data access over the radio interface.

GPRS [24] divides each radio frequency into TDM (Time-Division Multiplexing) frames and each frame contains 8 TDMA (Time Division Multiple Access) time-slots. Radio frequency and time-slots are allocated dynamically by the network to allow data transfer to and from the mobile station. It allows addition of new networks without change to the radio interface.

GPRS is a packet oriented mobile data technology based on GSM, supporting PPP (Point to Point Protocol) and IP protocols [25]. The main achievement in GPRS networks involves transferring packet data over GSM networks, which has two advantages:
- Combination of user data and wireless network resources.
- Transparent IP protocol transmission.

3.1.2.1 Architecture

New modules need to be added to a GSM network (Figure 27) to convert it into GPRS:
- Gateway GPRS Service Node (GGSN).
- Serving GPRS Service Node (SGSN).
- Border Gateway (BG): a gateway to other PLMN.
- Intra-PLMN (Public Land Mobile Network) backbone: IP based network which links the GPRS elements.
- Charging Gateway (CG).
- Legal Interception Gateway (LIG).

Figure 27. A GPRS architecture
Additional services are required to support GPRS over GSM; these include:
- PCUs (Packet Control Unit), hosted in the BSS (Base Station Subsystem).
- Domain Name System (DNS).
- Mobility management to locate the GPRS mobile station.
- Air interface for packet traffic.
- Firewalls for external network access.

3.1.2.2 Performance

Data rates varying from 9 to 150 Kilobytes per second (Kbps) can be achieved per user. GPRS allows deployment of applications that were not available on GSM networks due to data transfer speed limitations and lack of access to Internet.

GPRS data transfer is based on overlaying a packet-based air interface on the existing circuit switched GSM network, giving the user the ability to use a packet-based data service. GPRS radio resources are used only when users are actually sending or receiving data. Making efficient use of the radio resources allows a large number of users to share the same bandwidth in a single cell.

GPRS is not only a service to be deployed on GSM networks; it also supports the IS-136 TDMA standard [26].

3.1.2.3 Disadvantages

Theoretical maximum speed is 171.2 Kbps, with GPRS using eight timeslots at the same time and no error protection. High data speeds to individual mobile users require implementation of EDGE or UMTS (Universal Mobile Telephone System). GPRS modulation technique is GMSK (Gaussian Minimum-Shift Keying). EDGE is based on 8 PSK (Eight-Phase-Shift Keying) modulation, allowing higher bit rate over the air interface.

GPRS packets can be routed through different paths to reach the same destination; these packets might get lost and never reach their destination. As a recovery mechanism, GPRS standard implements data integrity and retransmission techniques, which can lead to delays in delivery of the data packets.

There is no store and forward mechanism in GPRS standard, compared SMS that does implement such engine.

3.1.2.4 Services

GPRS is able to provide any service available on Internet, including:
- File Transfer Protocol (FTP).
- Web browsing.
- Chat.
- E-mail.
3.1.2.5 Benefits

GPRS provides the following advantages [27]:
- Users share GPRS channels, using them only when there is need to send or receive data; this allows billing based on the data flow.
- GPRS provides users with transparent IP and Internet access.
- GPRS supports data transmission while voice calls are being made.

3.1.3. 3G

IMT-2000 (International Mobile Telecommunications), called 3G, is a series of mobile standards defined by ITU (International Telecommunication Union), including:
- GSM.
- EDGE.
- UMTS.
- CDMA2000 (Code Division Multiple Access).
- DECT (Digital Enhanced Cordless Telecommunications).
- WiMAX (Worldwide Interoperability for Microwave Access).

3G is designed to create global infrastructure in order to carry existing and future services. It unifies existing cellular standards into a single global system that includes both ground and air components.

3.1.3.1 Architecture

3G networks (Figure 28) are composed of overlapping mobile base stations which allow users to switch between cells [28]. The base stations are connected through a backhaul network to the Public Switched Telecommunications Network (PSTN).
3.1.3.2 Performance

3G standards support data rates ranging from 300 Kbps up to 2.4 Mbps, with actual rates of around 100 Kbps in commercial networks. Speeds depend on the application, according to [29] the first version of CDMA2000 supports:

- Download speed: 3 Mbps.
- Upload speed: 1.8 Mbps.
- Combined: 2.4 Mbps.

3.1.3.3 Disadvantages

Updating existing cellular infrastructure to support 3G standards is costly. 3G equipment needs to be installed on each cellular base station. New frequencies are needed for 3G transmissions.

Mobile devices based on 3G required more power than 2G models. According to [30] when 3G devices are in idle state, the current consumed is approximately 4.3 mA, while 2G devices only consume 3.3 mA. In connected state, 2G devices consume an average of 438 mA at 3.3 V; while 3G devices consume at least 573 mA, depending on the band used. This extra power consumed, leads to larger battery requirements.

3.1.3.4 Services

Services provided by 3G networks include [31]:

- Audio and video streaming.
- High speed Internet.
• TV through Internet (IPTV).

3.1.3.5 Benefits

3G provide high bandwidth, compared to 2G; this provides the following benefits:
• Quick access to multimedia and Internet tools.
• The mobile device targeted to be always online, ready for on the click Internet access.
• Billing for the connection is done only while sending or receiving packet data.

3.1.4. Bluetooth

Bluetooth uses a radio technology called FHSS (Frequency Hopping Spread Spectrum). It began as an alliance between mobile communications and mobile computing companies to develop a short range communication standard for wireless data transfer within 10 meters of distance. It allows data exchange between mobile devices such as phones, computers and GPS systems. The group of companies formed the Bluetooth SIG (Secure Internet Gateway), in order to define and license specifications for this technology. Bluetooth specification is divided in two categories:
• Core specification: Describes how the technology works.
• Profile specification: Describes how to build interoperating devices using the core technologies.

3.1.4.1 Architecture

Bluetooth devices are able to connect to each other, creating a PAN (Personal Area Network) (Figure 30). The Bluetooth stack as shown in Figure 29 is composed of these layers:
• Radio Frequency: It is the air interface, consisting of an antenna with power ranging from 0dBm (1mW) up to 20dBm (100mW). The radio uses a frequency starting at 2.402 GHz and stopping at 2.480 GHz.
• Baseband: This layer takes care of synchronizing the radio channels between devices. A connection between two Bluetooth devices is called piconet.
• Link manager: Starts and manages physical communication links, as well as channel status and control commands.
• L2CA (Logical Link Control and Adaptation Protocol): Starts and manages logical channels.
• Communication protocols: Protocols as RF-COMM (Radio Frequency Communication), TCP/IP and HID (Human Interface Device) are part of the Bluetooth application stack.
3.1.4.2 Performance

Bluetooth operates at a radio frequency of 2.4 GHz with data transmission speeds between 720 Kbps and 1 Mbps [32]. A Bluetooth frame consists of a transmit packet followed by a receive packet. Bluetooth allows multi-slot frames, eliminating turn-around time between packets and reducing header overhead, which results in higher data rates.
3.1.4.3 Disadvantages

The maximum distance supported between two devices to communicate using Bluetooth is:
- Class 1 devices support a range of up to 100 meters, with an actual range of up to 30 meters.
- Class 2 devices support a range of up to 30 meters, with an actual range of up to 10 meters.

3.1.4.4 Services

These are common applications of Bluetooth:
- Audio transfer between a mobile phone and a headset.
- File transfer and data synchronization between devices.

3.1.4.5 Benefits

Bluetooth operating frequency is under the unlicensed spectrum area, which makes the connection between devices free. It allows simultaneous data and voice transfer.

One differentiating factor from Bluetooth is its ability for connecting to other devices, with the need for a network access point. It also facilitates discovery and configuration of services between devices.

3.1.5. Wi-Fi

Wi-Fi is a wireless LAN, based on the radio technology 802.11, operating at frequencies of 2.4 and 5 GHz. A Wi-Fi network allows devices to connect with each other, to Internet and to other networks as Ethernet.

3.1.5.1 Architecture

A Wi-Fi network (Figure 31) is based on access points, which connect the wireless devices with an Ethernet network.
3.1.5.2 Performance

Wi-Fi provides performance similar to wired networks with speeds of up to 54 Mbps covering an area with a radio of up to 92 meters from the access point.

3.1.5.3 Disadvantages

The range of 802.11g network is on the order of tens of meters, which might be sufficient for a home network, but insufficient for larger networks, such the ones in buildings; additional repeaters or access points might be needed to cover all the area.
Wi-Fi networks are subject to interference, as any radio technology, making its real speed slower than common wired networks speeds.

3.1.5.4 Services

Wi-Fi allows connections between PCs, terminals, and computing resources such as servers, storage devices, or printers. It is able to provide access to Internet, Voice and data transfers, as well as multimedia streaming.

3.1.5.5 Benefits

Wireless LAN is the most common network in businesses and homes, due to several reasons:

- **Cost efficiency**: The spectrum over which Wi-Fi operates is unlicensed.
- **Ease of integration**: with other networks and users.
- **Most of the mobile computers currently on the market come pre-equipped with Wi-Fi technology.**
- **Public wireless networks are based on Wi-Fi and allow users to access Internet from anywhere, at lower costs and faster speeds than 3G networks.**

3.2. Development frameworks

3.2.1. Android SDK

Android SDK provides a debugger, libraries, and a handset emulator [33]. Google deploys this SDK as a plugin for Eclipse, Netbeans and IntelliJ in order to develop Android powered applications. Android OS is based on Linux kernel and GNU (GNU's Not Unix, Linux OS) software. Android core libraries are based in C/C++.

3.2.1.1 Architecture

Figure 32 provides a description of the components in the Android application development framework. The lowest layers consist of the Linux Kernel services. The layer above the kernel provides the core libraries for the framework. The application framework layer exposes the APIs for the final application development.
3.2.1.1 Features

Table 7 shows the outlines of Android SDK: the programming languages supported for developing applications; the operating systems on which the applications can be deployed; middleware supported for remote procedure invocation; and it highlights the support for development of native applications.

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming languages</td>
<td>Java</td>
</tr>
<tr>
<td>Operating systems</td>
<td>Android</td>
</tr>
<tr>
<td>Middleware</td>
<td>RMI, Sockets, Web Services</td>
</tr>
<tr>
<td>Native application development</td>
<td>Not supported</td>
</tr>
</tbody>
</table>
Table 7. Features for Android

3.2.2. Microsoft Visual Studio

Visual Studio [34] is an IDE that provides tools for developing different types of applications, such as:
- ASP.NET.
- XML Web Services.
- Desktop applications.
- Mobile applications.

Visual Studio .NET supports application development for mobile devices, using .NET Compact Framework, a subset of the .NET Framework. .NET Framework supports development in multiple programming languages. It consists of three main parts:
- Common Language Runtime: Just in time compiler for managing the execution of .NET programs.
- Unified programming classes: Common set of APIs for all supported programming languages.

3.2.2.1 Architecture

Figure 33 provides an overview of the stack for applications developed in Visual Studio. Visual Studio provides support for native and managed applications. Native applications can be written in C/C++ accessing Win32 APIs; while managed applications can be programmed in three different languages Visual C++, C# and Basic. Managed applications make use of CLR, which provides an abstraction of the target device.
### 3.2.2.2 Features

Table 8 shows the outlines of Microsoft Visual Studio: the programming languages supported for developing applications; the operating systems on which the applications can be deployed; middleware supported for remote procedure invocation; and it highlights the support for development of native applications.

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming languages</td>
<td>Visual Basic, Visual C#, and Visual C++, XML</td>
</tr>
<tr>
<td>Operating systems</td>
<td>Windows</td>
</tr>
<tr>
<td>Middleware</td>
<td>COM, CORBA, Sockets, Web Services</td>
</tr>
<tr>
<td>Native application development</td>
<td>Win32</td>
</tr>
</tbody>
</table>

Table 8. Features for Microsoft Visual Studio

### 3.2.3. wxWidgets

wxWidgets is a set of C++ libraries which allows development of common code in applications for Windows, OS X, Linux and UNIX on 32-bit and 64-bit architectures, as well as several mobile platforms including Windows Mobile/Phone, iPhone SDK and embedded GTK+ (Graphics Tool Kit). WxWidgets provides applications with access to platform’s native APIs using common code across multiple operating systems [35].

The main feature of wxWidgets is its ability to provide a native GUI appearance on the major platforms. It also provides classes for multiple types of applications [36], including:

- GUI.
- Files and streams.
- Multiple threads.
- Inter process communication.
- Online help.
- Database access.

### 3.2.3.3 Architecture

Figure 34 provides a description of the components in wxWidgets framework.
3.2.3.4 Features

Table 9 shows the outlines of wxWidgets: the programming languages supported for developing applications; the operating systems on which the applications can be deployed; middleware supported for remote procedure invocation; and it highlights the support for development of native applications.

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming languages</td>
<td>C++</td>
</tr>
<tr>
<td>Operating systems</td>
<td>Windows, Unix, Mac OS X, Windows Mobile/Phone, iPhone</td>
</tr>
<tr>
<td>Middleware</td>
<td>OS native middleware</td>
</tr>
<tr>
<td>Native application development</td>
<td>C++</td>
</tr>
</tbody>
</table>

Table 9. Features for wxWidgets

3.2.4. Mono

Mono is an open source project for development of Linux applications, which enables developers to build and deploy cross-platform Microsoft .NET applications [37]. Mono uses Common Language Runtime to execute applications from different programming languages [38]. The CLR code is the same in all platforms and applications do not need to be ported. This means porting an application to a new platform requires only the runtime tool to be ported.

A CLR compiler is provided by Mono for C# language. Source code from Mono applications is used to generate Microsoft Intermediate Language (MSIL), which is similar to assembly code; it is then taken by the JIT compiler to generate platform-dependent instructions.

3.2.4.1 Architecture

Mono framework (Figure 35) is composed of the following components:

- Common Language Infrastructure (CLI): A virtual machine providing a class loader, just-in-time compiler, and a garbage collecting runtime.
- Library of classes: Common libraries supported by CLR; including .NET compatible and Mono-provided class libraries.
- C# language compiler: Compiler compatible with CLR.

![Mono Architecture Diagram]

**Figure 35. Mono architecture**

### 3.2.4.2 Features

Table 10 shows the outlines of Mono SDK: the programming languages supported for developing applications; the operating systems on which the applications can be deployed; middleware supported for remote procedure invocation; and it highlights the support for development of native applications.

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming languages</td>
<td>C#</td>
</tr>
<tr>
<td>Operating systems</td>
<td>Linux, Mac OS X, Windows, BSD (Berkeley Software Distribution)</td>
</tr>
<tr>
<td>Middleware</td>
<td>COM, CORBA, Sockets, Web Services</td>
</tr>
<tr>
<td>Native application development</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

Table 10. Features for Mono

### 3.2.5. PhoneGap

PhoneGap is a framework for building cross-platform web applications [39]. It is an open source library based in JavaScript, which provides access to services and hardware on the target platforms. Mobile applications are developed in HTML (HyperText Markup Language) and JavaScript, accessing core features of the target device using PhoneGap APIs. PhoneGap based application source code can be used on multiple mobile platforms. Hardware features exposed by the PhoneGap API include:
- Geo-location.
- Vibration.
- Accelerometer.
- Audio.

### 3.2.5.1 Architecture

Figure 36 shows the architecture of PhoneGap:

![PhoneGap Architecture Diagram](image)

**Figure 36. PhoneGap Architecture**

### 3.2.5.2 Features

Table 11 shows the outlines of PhoneGap: the programming languages supported for developing applications; the operating systems on which the applications can be deployed; middleware supported for remote procedure invocation; and it highlights the support for development of native applications.

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming languages</td>
<td>HTML, JavaScript</td>
</tr>
<tr>
<td>Operating systems</td>
<td>iPhone, Android, BlackBerry, Palm and Symbian</td>
</tr>
<tr>
<td>Middleware</td>
<td>Web services, SOAP</td>
</tr>
</tbody>
</table>
Native application development | Supported
---|---

Table 11. Features for PhoneGap

3.2.6. J2ME-MIPI

J2ME is a tool for developing mobile applications, called Mobile Information Device Profile (MIDP); J2ME is a framework used for developing cross-platform applications for modern mobile phones [40].

3.2.6.1 Architecture

Figure 37 shows the architecture of J2ME:
- JVM: Virtual machine for the host operating system of a specific device.
- Configuration layer: Defines the functions of JVM and the smallest set of Java class library available. J2ME has two configuration specifications:
  - CLDC (Connected Limited Device Configuration)
  - CDC (Connected Device Configuration)
- Profile layer: Defines the smallest set of API available.
- MIDP: Java APIs for developing several aspects of an application, such as user interface, persistent storage and networking.

![J2ME-MIPI Architecture](image.png)

Figure 37. J2ME-MIPI architecture

3.2.6.2 Features

Table 12 shows the outlines of J2ME: the programming languages supported for developing applications; the operating systems on which the applications can be deployed; middleware supported for remote procedure invocation; and it highlights the support for development of native applications.

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming languages</td>
<td>Java</td>
</tr>
</tbody>
</table>
Operating systems | Any OS running JVM
---|---
Middleware | RMI, Web services
Native application development | Not supported

Table 12. Features for J2ME MIPI

3.2.7. Qt

Qt is a cross-platform application framework, which enables web and native application development. One of the main features of this framework is its support for desktop as well as mobile platforms. QT is not a runtime tool, therefore there are not performance constraints. QT Framework is developed in C++, and it provides libraries for each operating system. Building a QT application makes use of the platform specific pre-built QT libraries, making easy to integrate custom components. Applications can be written once and deployed across multiple operating systems, using the same source code.

Three steps are required for building and deploying an application in QT environment (Figure 38):

a) Generate the project by issuing a command in the directory where the source code resides: `qmake --project`.

b) Generate platform specific make files using “qmake” command, which uses the project files.

c) Generate an application for a specific platform using “make” command, which uses the make files.

![Figure 38. QT build process](image)

3.2.7.1 Architecture

Figure 39 shows QT architecture and its integration to the operating system. Qt includes a cross-platform class library, integrated development tools and a cross-platform IDE.
3.2.7.2 Features

Table 13 shows the outlines of Qt: the programming languages supported for developing applications; the operating systems on which the applications can be deployed; middleware supported for remote procedure invocation; and it highlights the support for development of native applications.

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming languages</td>
<td>C++, Javascript</td>
</tr>
<tr>
<td>Operating systems</td>
<td>Windows, Mac OS, Linux / X11, Embedded Linux, Windows Mobile/Phone, Maemo, Symbian (S60)</td>
</tr>
<tr>
<td>Middleware</td>
<td>RMI, CORBA, Web Services, Sockets</td>
</tr>
<tr>
<td>Native application development</td>
<td>Supported</td>
</tr>
</tbody>
</table>

Table 13. Features for Qt
3.3. Mobile application development tools

In this section we will analyze emulators and other tools used for developing mobile applications on several operating systems, including Android, Windows Mobile/Phone and Symbian.

3.3.1. Android Emulator

The Android SDK provides a mobile device emulator, a virtual mobile device that runs on a PC. The emulator enables developers to prototype, develop, and test Android applications without using a physical device.

![Android emulator](image)

Figure 40. Android emulator

The Android emulator [42], as shown in Figure 40, mimics all of the hardware and software features of a mobile device, except for phone calls. It supports Android Virtual Device (AVD), which is a configuration for specifying the hardware capabilities of the Android platform model to be simulated. An application running on the emulator can use the services of the Android platform to invoke other applications, access the network, play audio and video, store and retrieve data, notify the user, and render graphical transitions and themes.

The emulator provides debugging capabilities, such as:

- Log console.
- Interrupts simulation (arriving SMS messages or phone calls).
- Latency effects and dropouts simulations on the data channel.
The emulator provides complex networking capabilities to set up modeling and testing environments. Each instance of the emulator is assigned a virtual router isolated from the development machine's network interfaces. The emulator is able to forward voice calls and SMS messages from one instance to another.

Android Development Tools (ADT) plug-in for Eclipse allows creating and debugging Android applications. ADT plugin provides below features:

- Access to Android development tools from inside the Eclipse IDE, such as the DDMS (Dalvik Debug Monitor Server) tool, which allows:
  - Getting screenshots
  - Manage port-forwarding
  - Set breakpoints
  - View thread and process information
- Project wizard, to create and set up a new Android application.
- Code editor, for configuring Android manifest and resource files in XML format.
3.3.2. Windows Mobile/Phone emulator

In order to develop applications for Windows® phones, Visual Studio 2005 Standard Edition or above or Visual Studio 2008 Professional Edition or above is required. SDKs are required for each family of Windows phones. The SDKs include emulators (Figure 41) for a variety of Windows Mobile/Phone devices. However, a real device can be used, in which applications can be downloaded and debugged directly on the target hardware.

![Windows Mobile/Phone emulator](image)

Figure 41. Windows Mobile/Phone emulator

Windows Mobile/Phone SDKs allow the development of applications that make use of Microsoft .NET CF. Windows Mobile/Phone device emulators are installed as part of the SDKs, but also rely on components included in Visual Studio. The device emulator supports the following features:

- Configurable screen resolution.
- Flexible display orientation.
- Host-key combinations that support special functionality.
- Serial port mappings.
- Storage card emulation.
- Networking support.

In addition, the emulator supports multiple development environments, such as Visual Studio 2005, Visual Studio .NET 2003, and Windows Mobile/Phone 5.0 and later.

Features that are new in version 2 of the emulator include:

- Improvements to execution and debug/deploy performance.
- Accelerated video support for Direct3D Mobile.
- Backlight support.
- Improved folder sharing.
- Emulation of battery and notification LEDs.
- Coordinated NAND Flash.
- USB host mode.
3.3.3. Symbian emulator

Symbian SDK provides an emulator (Figure 42) for debugging applications. The emulator allows execution and debugging of native, as well as Qt-based applications. The features of this emulator are listed below:

- Configuration Tool, which allows setting the capabilities of the emulated Symbian device.
- TCP/IP support using Winsock.
- Applications in emulator:
  - Web browser
  - RealPlayer
- Emulated events, such as incoming call and SMS.

![Figure 42. Symbian device emulator](image)

3.4. Conclusions

Table 14 compares the different wireless technologies explored in this chapter. GSM, GPRS and 3G networks cover a large area with a low data transfer rate, consume more power and are costly to use. Bluetooth and Wi-Fi cover a short area consuming less power; Wi-Fi provides higher data transfer rate. The purpose of this research is to select the wireless technology that best fits the PARM1-based distributed system model, which is intended to be implemented on small and medium enterprises; these enterprises require a low cost mobile distributed system; this requirement contains two factors:

- Low cost network: A network which is free for access and easy to deploy in a medium size office.
- Low cost mobile devices: Devices common in the market; with different OS and different hardware capabilities, but a common transport layer support.

<table>
<thead>
<tr>
<th>Features</th>
<th>GSM</th>
<th>GPRS</th>
<th>3G</th>
<th>Bluetooth</th>
<th>WiFi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>900 to 1800 MHz</td>
<td>900 to 1800 MHz</td>
<td>900 to 1800 MHz</td>
<td>2.4 GHz and 5 GHZ</td>
<td></td>
</tr>
<tr>
<td>Coverage area</td>
<td>35 km</td>
<td>35 km</td>
<td>35 km</td>
<td>10m</td>
<td>92 m</td>
</tr>
<tr>
<td>Power Consumption</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Active</td>
<td>1700mW</td>
<td>1445.4mW</td>
<td>1758.9mW</td>
<td>500mW</td>
<td>425mW</td>
</tr>
<tr>
<td>Idle</td>
<td>650mW</td>
<td>10.89mW</td>
<td>14.19mW</td>
<td>285mW</td>
<td>285mW</td>
</tr>
<tr>
<td>SMS</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Voice</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Email</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multimedia</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>P2P connection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Data transfer rates</td>
<td>9.6 Kbps</td>
<td>150 Kbps</td>
<td>2.4 Mbps</td>
<td>720 Kbps</td>
<td>54 Mbps</td>
</tr>
<tr>
<td>Cost</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 14. Comparison of different wireless technologies

Different cross-platform development frameworks and mobile SDKs were studied in the second half of this chapter. According to the features of each, one of them will be selected and used for developing the prototype of the project. Table 15 shows a raw comparison between the frameworks analyzed in this chapter.
<table>
<thead>
<tr>
<th>OS</th>
<th>Windows</th>
<th>Mac OS</th>
<th>Linux/ X11</th>
<th>Embedded Linux</th>
<th>Windows Mobile/Phone</th>
<th>Symbian</th>
<th>Android</th>
<th>Iphone</th>
<th>BlackBerry</th>
<th>Unix</th>
<th>Palm</th>
<th>BSD</th>
<th>Maemo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Middleware for RPC</th>
<th>RMI</th>
<th>CORBA</th>
<th>Web Services</th>
<th>Sockets</th>
<th>SOAP</th>
<th>COM</th>
<th>Win32</th>
<th>C++</th>
<th>x</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Native application development | | |
|-------------------------------|---|---|---|---|---|---|
|                               | x | x | x | x | x | x |

Table 15. Cross-platform development frameworks

From the table above, we are able to identify that Web Services and Sockets are middleware supported by most of the cross-platform development frameworks. This is taken into consideration for the PARM framework design, as it tries to integrate multiple devices and environments using a common layer, and making use of the existing infrastructure. Sockets are a good candidate for the development of a common transport layer.
Chapter 4. Evaluation of middleware and existing distributed systems

Strategies and previous work related to this project is covered in this chapter. An introduction to middleware for distributed systems is provided, as well as technologies for their development. At the latter part of the chapter, existing mobile distributed systems are studied and compared in order to identify the areas of improvement, which can be covered by the PARM framework.

This chapter provides a comparison of remote procedure call middleware, including CORBA, COM, RMI, Web Services, SOAP, Sockets (BSD Socket API). Features being analyzed for each middleware include operating system and programming language support, architecture and services provided.

In the second half of the chapter existing mobile distributed systems are presented. These systems are based on either hardware or software to differentiate from each other. Characteristics being analyzed include network infrastructure, physical communication technology, operating systems and programming languages supported, development tools, middleware, services provided, software architecture, advantages and disadvantages.

4.1. Remote Procedure Call Middleware

RPC (Remote Procedure Call) middleware is the technology that enables implementation of distributed systems, and is one of the leading factors towards the definition of a distributed system model. Some of the most outstanding middleware technologies for remote communication are presented in this section.

4.1.1. CORBA

CORBA was created in 1989 by the Object Management Group (OMG), to enable remote invocation from object-oriented applications. The key element of CORBA standard is ORB, which handles the requests of the operations between distributed objects. These objects can be heterogeneous in various dimensions:

- Development platforms.
- Operating systems.
- Network interfaces.
- Programming languages.
- Communication protocols required for the middleware to run.

CORBA programming is based on the definition of object interfaces, using IDL [43]. IDL allows binding objects created in different programming languages.

4.1.1.1 Software architecture

Figure 43 shows the various components in the CORBA architecture:

- Client: It performs application tasks by obtaining references to remote objects and invoking operations on them.
- Servant: Supports operations as defined in the IDL, by creating objects in an implementation language such as C++ or Java among many others.
- ORB Core: It delivers the request for an operation on a remote object and returns the response.
- General Inter-ORB Protocol (GIOP): It is the layer that enables ORB Core accessing remote objects.
- Internet Inter-ORB Protocol (IIOP): This layer allows access to TCP transport protocol.
- ORB Interface: These are the APIs exposed to the application for abstracting an ORB.
- OMG IDL Stubs and Skeletons: Stubs provide a Static Invocation Interface (SII) which marshals application parameters into a common data-level representation. Skeletons de-marshal the data-level representation back into typed parameters according to the servant.

![CORBA Communication Model](image)

**Figure 43. CORBA Communication Model**

### 4.1.1.2 Features

Table 16 shows the outlines of CORBA: the programming languages supported; the operating systems on which this middleware can be deployed; services provided; and the communication protocols that the middleware can run on [44]:

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating systems</td>
<td>Windows, Unix, Windows Mobile/Phone, Symbian, VxWorks</td>
</tr>
<tr>
<td>Programming languages</td>
<td>C, C++, Java, ADA, COBOL (Common Business-Oriented Language), Lisp, Python, Smalltalk</td>
</tr>
<tr>
<td>Services provided</td>
<td>Remote object operations</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Communication protocols</td>
<td>TCP/IP, IPX (Internetwork Packet Exchange)/SPX (Sequenced Packet Exchange), FDDI (Fiber Distributed Data Interface), ATM (Asynchronous Transfer Mode), Ethernet, Fast Ethernet, embedded system backplanes, and shared memory</td>
</tr>
</tbody>
</table>

Table 16. Features supported by CORBA

4.1.2. COM

COM (Component Object Model) framework, created by Microsoft, allows communication between applications written in different programming languages; this is possible via COM objects. COM objects exist in a process, another process or in remote computers. The only requirements for the programming language in which a COM object is created, is the ability to handle pointers to functions and structures of pointers.

A software object is composed by a set of data and functions to manipulate the data. Similarly, a COM object contains data that is exclusively accessed through functions; these functions are called methods and defined in interfaces. COM allows access to the methods of an interface through a pointer to the interface.

In languages like C++ and Java, a COM object is defined by an IDL file which is processed by a MIDL (Microsoft Interface Definition Language) compiler to create a type library or header and proxy files. A type library is a binary file describing the COM interfaces.

4.1.2.1 Software architecture

Figure 44 shows the various components in the COM architecture:
- **Object**: It is a software representation of a data set, with functions to manipulate it.
- **Interfaces**: Definition of methods to access data of an object.

![Figure 44. COM software architecture](image-url)
4.1.2.2 Features

Table 17 shows the outlines of COM: the programming languages supported; the operating systems on which this middleware can be deployed; services provided; and the communication protocols that the middleware can run on:

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating systems</td>
<td>Windows</td>
</tr>
<tr>
<td>Programming languages</td>
<td>Visual Basic, C, C++, VBScript and Java</td>
</tr>
<tr>
<td>Services provided</td>
<td>Remote method invocation</td>
</tr>
<tr>
<td>Communication protocols</td>
<td>TPC/IP, HTTP (Hypertext Transfer Protocol)</td>
</tr>
</tbody>
</table>

Table 17. Features supported by COM

4.1.3. Java RMI

RMI allows an object running on a JVM to invoke a method on a remote object that resides in another JVM. RMI is part of the core Java API, and is critical for implementation of distributed objects in distributed application development [45].

RMI provides basic object services. These services are:
- Naming/Registry Service: Allows a server process to register RMI-enabled objects with its local RMI registry, using a name that clients can refer to.
- Distributed garbage collection: This is an automatic process.
- Object activation service: Enables a server object to be activated automatically when a client requests it.

4.1.3.1 Software architecture

Figure 45 shows the various components in the RMI architecture. It is composed of three layers [45]:
- Stub-Skeleton layer: Provides the interface that client and server application objects use to interact with each other.
- Remote reference layer: Middleware between the stub-skeleton layer and the underlying transport protocol.
- Transport protocol layer: Binary data protocol that sends remote object requests over the network.

A stub is used to make method invocations, by marshaling the invocation and sending it to the remote RMI server. All RMI servers register with the registry server of their hosting remote objects. RMI clients look up stubs of a remote object on the registry server.
Figure 45. Java/RMI software architecture

4.1.3.2 Features

Table 18 shows the outlines of Java RMI: the programming languages supported; the operating systems on which this middleware can be deployed; services provided; and the communication protocols that the middleware can run on:

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating systems</td>
<td>Any OS supporting a JVM</td>
</tr>
<tr>
<td>Programming languages</td>
<td>Java</td>
</tr>
<tr>
<td>Services provided</td>
<td>Remote method invocation, naming registry, distributed garbage collection, object activation.</td>
</tr>
<tr>
<td>Communication protocols</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

Table 18. Features supported by Java RMI

4.1.4. SOAP

SOAP [46] is an XML-based protocol for exchanging data over HTTP TCP, UDP (User Datagram Protocol), BEEP (Blocks Extensible Exchange Protocol) or SMTP (Simple Mail Transfer Protocol) [47]. It allows message exchange between applications, by defining the structure of an XML document to
represent the request and the response of a service. Applications written in different languages and deployed on different platforms can communicate with each other over the network.

Web services use SOAP to send messages between a service and its clients. These support for asynchronous and synchronous interactions, as well as the use of Internet protocols such as HTTP, SMTP, FTP and MIME (Multipurpose Internet Mail Extensions), which make the connection.

A Web service is defined by two core XML-based components [48]:

- WSDL (Web Services Description Language): Describes the service location on the Web and the functionality the service provides. Web services are self-describing; they can describe themselves, notify what operations it supports and how to invoke them.
- UDDI (Universal Description, Discovery and Integration). Information about the service may then be entered in a UDDI registry, which allows Web service consumers to search for and locate the services they need.

### 4.1.4.1 Software architecture

SOAP consists of three parts:

- Envelope: A framework for expressing what is in a message.
- Encoding rules: A serialization mechanism that can be used to exchange instances of application-defined data types.
- RPC representation: A convention to represent remote procedure calls and responses.

Figure 46 shows the various components in the SOAP architecture:

![Figure 46. SOAP client-server communication](image-url)
Figure 47 shows the SOAP framework which allows clients to publish, find and request services.

Figure 47. SOAP software architecture

4.1.4.2 Features

Table 19 shows the outlines of SOAP: the programming languages supported; the operating systems on which this middleware can be deployed; services provided; and the communication protocols that the middleware can run on:

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating systems</td>
<td>Any OS</td>
</tr>
<tr>
<td>Programming languages</td>
<td>Any Language</td>
</tr>
<tr>
<td>Services provided</td>
<td>Service discovery, data and functions access</td>
</tr>
<tr>
<td>Communication protocols</td>
<td>HTTP, TCP, UDP, BEEP or SMTP</td>
</tr>
</tbody>
</table>

Table 19. Features supported by SOAP

4.1.5. Enterprise Java Beans

Enterprise JavaBeans (EJB) is a technology specific to Java and allows distributed component applications. It is based in an architecture where objects live on a server. The server provides interfaces for finding, creating and destroying objects. The objects are instances of a bean component; the component provides interfaces to functions. There are three types of java beans [49]:

- Session: Beans whose instances are dedicated to a single client. These are divides in two categories:
  - Stateless: No state is maintained between calls to its methods.
  - Stateful: State is maintained across the life of the object.
- Entity: Allows representation of an entity in a database; it exposes a row in the database.
- Message-driven: These run asynchronously in response to messages received from the Java Messaging Service (JMS) provider. A message posted by a client to a JMS queue or destination, is then forwarded to the beans subscribed to the same destination.

An EJB container is the virtual environment that hosts and manages EJB objects. It provides services to the bean object, such as transaction management, security, persistence, remote access, lifecycle management (find, create, destroy) and database access.

4.1.5.3 Software architecture

Figure 48 shows the various components in the Enterprise Java Beans architecture. An EJB server provides a standard set of services:
- Distributed transaction management service: It allows transactions between clients and JavaBean objects.
- EJB container: It manages and controls services for JavaBean objects, such as the EJB Home interface, which provides access to the bean’s lifecycle services, like finding, creating and destroying an object.
- Java Naming and Directory Interface (JNDI): It allows a client locating the EJB Home interface to:
  - Create a new EJB object.
  - Find an existing entity bean instance.
4.1.5.4 Features

Table 20 shows the outlines of EJB: the programming languages supported; the operating systems on which this middleware can be deployed; services provided; and the communication protocols that the middleware can run on:

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating systems</td>
<td>Any OS supporting a JVM</td>
</tr>
<tr>
<td>Programming languages</td>
<td>Java</td>
</tr>
<tr>
<td>Services provided</td>
<td>Service discovery, transactions and persistence</td>
</tr>
<tr>
<td>Communication protocols</td>
<td>TCP/IP</td>
</tr>
</tbody>
</table>

Table 20. Features supported by Enterprise Java Beans
4.1.6. Sockets

A socket is a connection between two pieces of software. Applications can communicate using standard mechanisms built into network hardware and operating systems using sockets. Socket-based communication is independent of the programming language in which it is implemented. A socket program written in Java language can communicate to a program written in C or C++ socket program.

Socket interfaces are divided in three categories:
- **Stream socket**: Connection-oriented mechanism which requires both communicating entities first establish a socket connection.
- **Datagram socket**: Connection-less mechanism. A client sends datagrams as needed and waits for the other to respond; messages can be lost or received out of order.
- **Raw socket**: Bypasses the built-in support for standard protocols like TCP and UDP. Raw sockets are used for custom low-level protocol development.

4.1.6.5 Software architecture

Figure 49 shows the various components in the Sockets architecture. TCP sockets are packets of data encapsulated by each layer in the network communication stack.

![Figure 49. Sockets software architecture](image)

4.1.6.6 Features

Table 21 shows the outlines of Sockets: the programming languages supported; the operating systems on which this middleware can be deployed; services provided; and the communication protocols that the middleware can run on:

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating systems</td>
<td>Any OS</td>
</tr>
</tbody>
</table>
Programming languages  Any Language
Services provided  Session, data exchange
Communication protocols  TCP, Bluetooth

Table 21. Features supported by Sockets

4.2. Existing mobile distributed systems

4.2.1. Web services-based system for ad-hoc mobile application integration

Ad-hoc networks are the ones formed temporarily when 2 devices connect to each other; ad-hoc mobile application integration is the temporary communication of software modules running on 2 different devices connected via ad-hoc network.

The system in [50] allows information services to be accessed as a user enters an area within range wireless access points and for these services to be available to heterogeneous mobile devices. When a Bluetooth or Wi-Fi enabled device enters an area of service, it makes a request to a known standard bootstrap service at the access point to get a summary of the services available.

Communication between the client devices and the server is via web services. The layers of the system are composed of standards such as SOAP, WSDL, UDDI and BPEL4WS (Business Process Execution Language for Web Services). However, the main contribution of this system is a set of standards for discovering and accessing local services at different locations.

4.2.1.1 Network infrastructure

Figure 50 shows how the system can be deployed among mobile devices. A device is able to gather information about its surrounds by querying a server on the access point.

Figure 50. Mobile devices communicate with server at wireless access point when in range
4.2.1.2 Software architecture

Figure 51 shows the data flow between a device and a server in the system. The first call starts on the mobile device and is a query for the list of available services. Then the server responds with the information requested. Additionally, the user can start requests for specific services.

Figure 51. Standard XML web service requests to server

Local resources register with the server to advertise their services, as shown in Figure 52.

Figure 52. Web service registration with server

4.2.1.3 Features

Table 22 shows the characteristics of the system: the wireless technology used for communication between the entities; the programming languages, tools and RPC middleware used for developing the system; the operating systems on which the system can be deployed; and the services provided.
### Features Supported

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless technology</td>
<td>Bluetooth, Wi-Fi</td>
</tr>
<tr>
<td>Operating system</td>
<td>Any OS</td>
</tr>
<tr>
<td>Programming language</td>
<td>XML</td>
</tr>
<tr>
<td>Development tools</td>
<td>None specific</td>
</tr>
<tr>
<td>RPC Middleware</td>
<td>Web Services</td>
</tr>
<tr>
<td>Services provided</td>
<td>Retrieval of information in response to a user request; and autonomous dialogue between the mobile device and the local services.</td>
</tr>
</tbody>
</table>

#### 4.2.2. Location based system for mobile devices

The system in [51] is based on a mobile device with a RFID reader attached to it, which is used for identifying objects around a building. The objects are assigned RFID tags, which are used by the mobile device to provide location information to the server using a wireless network.

#### 4.2.2.1 Network infrastructure

Figure 53 shows the network infrastructure of the system. The mobile device provides 2 interfaces:
- RFID is used for identifying objects.
- Wireless access to a server, through an access point.

![Figure 53. High level Network model of overall system](image-url)
4.2.2.2 Software architecture

Figure 54 shows the software architecture of the system, which is based on a client and a server; the client application running on the mobile device and collecting information via RFID; the server application servicing the clients and interfacing with a database.

![Software Architecture Diagram](image)

Figure 54. The communication model of the system

4.2.2.3 Features

Table 23 shows the characteristics of the system: the wireless technology used for communication between the entities; the programming languages, tools and RPC middleware used for developing the system; the operating systems on which the system can be deployed; and the services provided.

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless technology</td>
<td>RFID, WiFi</td>
</tr>
<tr>
<td>Operating system</td>
<td>Any OS supporting J2ME</td>
</tr>
<tr>
<td>Programming language</td>
<td>J2ME</td>
</tr>
<tr>
<td>Development tools</td>
<td>None</td>
</tr>
<tr>
<td>RPC Middleware</td>
<td>HTTP</td>
</tr>
<tr>
<td>Services provided</td>
<td>Positioning system</td>
</tr>
</tbody>
</table>

Table 23. Characteristics of the system
4.2.3. SMS-based web search

A system based on SMS communication is a system where a user would send a query to a server using a text message and the server responds to the query with another text message.

SMS-Find [3] is a search system based on SMS messages that enables users to obtain search responses. It uses a conventional search engine as a back-end. The engine extracts results from downloaded pages and returns them to the SMS gateway.

4.2.3.1 Network infrastructure

Figure 55 shows the network infrastructure of the system:
- SMSC: It is composed by a front-end to send and receive SMS messages.
- SMSFind server: It is the interface between the search engine and the SMSC.
- Search engine: It is the core of the system, taking care of generating responses to the users.
- Database: Information to collect responses to the uses.

![Figure 55. System Architecture](image)

4.2.3.2 Software architecture

The architecture of system (Figure 56) consists of a client application querying the server; and the query takes care of generating a response to the clients.

![Figure 56. SMS search system architecture](image)
4.2.3.3 Features

Table 24 shows the characteristics of the system: the wireless technology used for communication between the entities; the programming languages, tools and RPC middleware used for developing the system; the operating systems on which the system can be deployed; and the services provided.

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless technology</td>
<td>GSM</td>
</tr>
<tr>
<td>Operating system</td>
<td>Any supporting SMS</td>
</tr>
<tr>
<td>Programming language</td>
<td>Python</td>
</tr>
<tr>
<td>Development tools</td>
<td>None</td>
</tr>
<tr>
<td>RPC Middleware</td>
<td>SMS</td>
</tr>
<tr>
<td>Services provided</td>
<td>Query information</td>
</tr>
</tbody>
</table>

Table 24. Characteristics of the system

4.2.4. Java RMI over Bluetooth

The distributed system in [52] enables Java RMI over Bluetooth protocol stacks, adding a set of layers to the stack, called JavaBT. It also enables offloading OS tasks to remote site servers with RMIs. Access to the Bluetooth hardware capabilities is possible via the JavaBT layer. The prototype is tested with a pair of Ericsson Bluetooth Development Kits (EBDKs) connected to a personal computer. JavaBT is implemented over JDK 1.1.8 with JavaCOMM API 2.0 for Microsoft Windows.

The system implements a custom Java RMI library, to send and receive packets via Bluetooth L2CAP (Logical Link Control and Adaptation Protocol) layer, as a replacement for TCP/IP sockets. Figure 57 shows the call sequences between the server and the client using Bluetooth sockets. The sequence is similar to TCP/IP. The components involved in this type of connection are:

- BluetoothServerSocket: Listens for connection requests using the L2CAP service.
- BluetoothSocket: It is tied to a L2CAP Channel Identifier (CID) to maintain a connection.
4.2.4.1 Network infrastructure

Figure 58 shows the purpose of the system; which is supporting Java RMI across multiple communication technologies.

4.2.4.2 Software architecture

Figure 59 (a) shows the stack of an application using RMI over the standard TCP/IP protocol via Bluetooth. Figure 59 (b) shows the improvement made by the system, which is implementing RMI directly over Bluetooth layer.
4.2.4.3 **Features**

Table 25 shows the characteristics of the system: the wireless technology used for communication between the entities; the programming languages, tools and RPC middleware used for developing the system; the operating systems on which the system can be deployed; and the services provided.

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless technology</td>
<td>Bluetooth</td>
</tr>
<tr>
<td>Operating system</td>
<td>Windows Desktop</td>
</tr>
<tr>
<td>Programming language</td>
<td>Java</td>
</tr>
<tr>
<td>Development tools</td>
<td>None</td>
</tr>
<tr>
<td>RPC Middleware</td>
<td>RMI</td>
</tr>
<tr>
<td>Services provided</td>
<td>Packet send and receive</td>
</tr>
</tbody>
</table>

Table 25. Characteristics of the system

4.2.5. **Microsoft’s .NET**

In [53] a web-based system for data acquisition is presented as an environmental experiment system. The purpose of the system is to control data acquisition devices remotely. A server provides access to the instruments, communicating with the devices via GPIB (General Purpose Interface Bus) or RS-232 physical links. Access to the measurement instruments is granted by its own driver which is wrapped by a custom library, called AMS COM. This library exposes the features of each instrument, allowing a web server to access them. The interface on the web server is called AMS module. The server enables remote users to control the instruments locally.
4.2.5.1 Network infrastructure

Figure 60 describes the system architecture; a core server controls three data acquisition devices, two environmental testing chambers, and one multimeter. The core server also hosts a web server, and manages a database.

![AMS hardware architecture diagram]

Figure 60. AMS hardware architecture

4.2.5.2 Software architecture

Figure 61 shows the software architecture of the system; it is composed of an http client and a web server.
4.2.5.3 Features

Table 26 shows the characteristics of the system: the wireless technology used for communication between the entities; the programming languages, tools and RPC middleware used for developing the system; the operating systems on which the system can be deployed; and the services provided.

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless technology</td>
<td>Wi-Fi</td>
</tr>
<tr>
<td>Operating system</td>
<td>Windows Desktop</td>
</tr>
<tr>
<td>Programming language</td>
<td>C++</td>
</tr>
<tr>
<td>Development tools</td>
<td>Visual Studio</td>
</tr>
<tr>
<td>RPC Middleware</td>
<td>COM</td>
</tr>
<tr>
<td>Services provided</td>
<td>Data acquisition</td>
</tr>
</tbody>
</table>

Table 26. Characteristics of the system

4.2.6. P2P networks for mobile devices

There are three main architectures for mobile P2P (Peer to Peer) networks [54]:

- Centralized: Composed by a set of peers and a dedicated server or several central index servers to control the network and keep track of the connected peers and their resources.
- Decentralized: A network in which meta-information is maintained by peers, peer discovery and query delivery functions are distributed among peers.
- Hybrid: It is a combination of centralized and decentralized architectures, combining the efficiency and resilience of both architectures.
The system in [54] enables mobile P2P file sharing over a GSM mobile network. In this model, a mobile phone connects to a database server using GPRS connection, the server then replies with a list of available peers sharing files and details to connect to them. Then the mobile phone sends a SMS connection to one of the peers with a request for getting a file; the peer sends the requested file via MMS (Multimedia Messaging Service).

The proposed model is based on the hybrid architecture, with the database server acting as a link to the mobile devices in the network.

4.2.6.1 Network infrastructure

As shown in Figure 62, the phones in the network form a peer to peer file sharing network. However, the steps to form this network are defined in Figure 63; the server starts the link with the rest of the peers.

Figure 62. A virtual mobile P2P network
The server hosts the Database of Meta Information. The Meta Information includes file names, file type, file size and the details of phone that keeps the file.

1. Mobile Peer A sends out a Request for Data on a particular file and where it could be found.
2. Response from the Web server about the mobile phone that has the content that has been searched for.
3. Peer A sends out request for content from Peer B using the Short Messaging Service (SMS).
4. Peer B replies Peer A with the request File using Multimedia Messaging Service (MMS).

Figure 63. The mobile P2P process

4.2.6.2 Software architecture

The model for this system is shown in Figure 64. Main components in the system are:

- Database server: It keeps track of files and phones in the network.
- File sharing protocol: Short and multimedia messaging services.
- Phone application: It initiates a search for a file on the mobile P2P network.
4.2.6.3 Features

Table 27 shows the characteristics of the system: the wireless technology used for communication between the entities; the programming languages, tools and RPC middleware used for developing the system; the operating systems on which the system can be deployed; and the services provided.

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless technology</td>
<td>GSM</td>
</tr>
<tr>
<td>Operating system</td>
<td>Windows Mobile/Phone, Symbian</td>
</tr>
<tr>
<td>Programming language</td>
<td>Java</td>
</tr>
<tr>
<td>Development tools</td>
<td>Sun wireless mobile toolkit phone emulator</td>
</tr>
<tr>
<td>RPC Middleware</td>
<td>SMS</td>
</tr>
<tr>
<td>Services provided</td>
<td>File sharing</td>
</tr>
</tbody>
</table>

Table 27. Characteristics of the system
4.2.7. SOWCAS system architecture

SOWCAS (Service Oriented Wireless Context-Aware System) [4] is a mobile distributed system that integrates existing standards from mobile devices and enterprise systems.

SOWCAS architecture is based on multiple middleware technologies for remote procedure invocation and data transfer, such as TCP/UDP socket connections, RMI and SOA (Service-Oriented Architecture). This system also integrates an indoor and outdoor positioning system, which keeps track of any user having an IEE 802.11, Bluetooth and GPS in the Faith University campus.

4.2.7.1 Network infrastructure

The physical layer of the architecture is shown in Figure 65. SOWCAS proxy software runs on computers located in classes, laboratories and special locations with access to wireless access points; these computers are used to locate and track the positions of mobile users in the university.

![Figure 65. SOWCAS system architecture [55]](image-url)
4.2.7.2 Software architecture

Client and server architectures component descriptions are shown in Figure 66 and Figure 67. Access to the wireless technologies is provided via C/C++ APIs system specific. SOWCAS middleware consists of multiple layers such as a JVM, distribution middleware, such as RMI and SOAP, common middleware services, and domain specific middleware services.

Figure 66. SOWCAS client [55]

Figure 67. SOWCAS server [55]
### 4.2.7.3 Features

Table 28 shows the characteristics of the system: the wireless technology used for communication between the entities; the programming languages, tools and RPC middleware used for developing the system; the operating systems on which the system can be deployed; and the services provided.

<table>
<thead>
<tr>
<th>Features</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless technology</td>
<td>WLAN (Wireless Local Area Network), Bluetooth</td>
</tr>
<tr>
<td>Operating system</td>
<td>Any OS supporting Java or .Net CF</td>
</tr>
<tr>
<td>Programming language</td>
<td>Java, C, C++</td>
</tr>
<tr>
<td>Development tools</td>
<td>Sybase's and Pocket Builder</td>
</tr>
<tr>
<td>RPC Middleware</td>
<td>TCP sockets, RMI and Web Services</td>
</tr>
<tr>
<td>Services provided</td>
<td>Positioning engine, database access</td>
</tr>
</tbody>
</table>

Table 28. Characteristics of the system

### 4.3. Conclusions

Middleware technologies enable development of distributed systems; some of them allow interaction between different programming languages or operating systems. SOAP is the main contributor to integration of heterogeneous devices, since the interaction between each entity is based in XML. A major disadvantage of SOAP is the performance; since is not as efficient as distributed architectures that use binary protocols for communication. Web services use plain text protocols to identify data; this means that Web service requests are larger than requests encoded with a binary protocol. SOAP encourages code reuse among different applications [56], but the constraint is that Web services only allow for some very basic forms of service invocation.

EJB enables development of distributed, transactional, secure and portable applications out of reusable components [57]. Server-centric applications can use existing EJB components. EJB architecture provides enterprise-level system services, leaving only the business logic on the client side. The constraint for EJB is the programming language, which is only Java.

COM objects are a good solution for distributing objects and defining interfaces; however they are limited to Windows operating system. On the other side, TCP sockets are a good protocol for exchanging messages between remote computers; sockets are independent of programming languages and operating systems, but there is really no support for defining interfaces to access remote objects. Table 29 provides the features supported by each of the middleware technologies described in the first section of this chapter.
<table>
<thead>
<tr>
<th>Programming Languages</th>
<th>Mac OS</th>
<th>Linux/ XII</th>
<th>Embedded Linux</th>
<th>Windows Mobile/Phone</th>
<th>Symbian</th>
<th>Android</th>
<th>Iphone</th>
<th>Blackberry</th>
<th>Palm</th>
<th>VxWorks</th>
<th>Unix</th>
<th>Brew Mobile Plattform</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>C++</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>C#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Java</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Visual Basic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>XML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>VBScript</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Java Script</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Services</td>
<td>Service discovery</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Remote object operations</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Data transfer</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Remote method invocation</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Session</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Naming registry</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Distributed garbage collection</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Object activation</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Data and functions access</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Transactions and persistence</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Communication Protocols</td>
<td>TCP/IP</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>TCP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>HTTP</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>UDP</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>BEEP</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Technology</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
<td>S5</td>
<td>S6</td>
<td>S7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>----</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>SMTP</td>
<td>x</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>IPX/SPX</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FDDI</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATM</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fast Ethernet</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Embedded system</td>
<td></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>backplanes</td>
<td></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>Shared memory</td>
<td></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>Bluetooth</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 29. Middleware technologies comparison

Table 30 shows a comparison of the mobile distributed systems presented in this chapter; the names of the systems are abbreviated and can be interpreted as:

- S1: Web services-based system for ad-hoc mobile application integration.
- S2: Location based system for mobile devices.
- S3: SMS-based web search.
- S4: Java RMI.
- S5: Microsoft’s .NET.
- S7: SOWCAS system architecture.
<table>
<thead>
<tr>
<th>Features</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wireless technology</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WiFi</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RFID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>GSM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Bluetooth</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Operating system</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SunOs</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows Desktop</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows CE</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symbian</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iPhone</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Android</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td><strong>Programming language</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>J2ME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Phyton</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Java</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C++</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Development tools</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visual Studio</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Sun Wireless</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Sybase’s Pocket Builder</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td><strong>RPC Middleware</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Services</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>TCP Sockets</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>RMI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>SMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>CORBA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>COM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>HTTP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td><strong>Services provided</strong></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Positioning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data transfer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data base access</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 30. Comparison of existing mobile distributed systems.

S1 is based on web services and allows any combination of mobile device and access point platforms. Server and clients can be running on any operating system. The low memory and processing capacity of mobile devices reduces the ability of the same for storing and querying every single resource.
on a new network; having information and services on a server is efficient for obtaining specific information. Service discovery is implemented by making use of Universal, Description, Discovery and Integration (UDDI); providing an intelligent link between the server and the clients. One of the disadvantages of this system is imposed by the use of XML, which requires processing the strings on every message.

S2 has a deployment limitation; the client application can only run on mobile devices supporting J2ME. However, its main contribution is supporting RFID to identify objects via wireless, which implies low power requirements for operation.

S3 implements a SMS-based search, which is very different from conventional mobile search via XHTML/WAP. An attractive aspect of this system is its accessibility, due to the use of low-end phones and extensive availability of SMS mobile devices.

The purpose of S4 is to support Java RMI across multiple communication technologies, including Bluetooth, Wi-Fi and Ethernet. Enabling Java RMI to directly access the Bluetooth L2CAP layer to send and receive packets improves the time for processing each message, by removing additional marshaling requirements imposed by using TCP/IP protocol.

S5 wraps data acquisition device drivers under a COM-based library allowing the system designer to focus on developing higher layers of the server to access the measurement instruments, without being familiar with the technologies of instrument control; such as GPIB, which implements device-specific commands.

S6 uses a MySQL database in a central server to keep track of the information shared by the mobile devices connected to the system. The centralized server prevents every peer in the network to send extra SMS queries to every peer; which is time consuming and costly for every peer.

The purpose of S7 is to demonstrate the performance for each of the middleware technologies; showing that Web Services implementations perform slower than RMI and TCP socket implementations. XML metadata in SOAP messages for Web Services requires higher network bandwidth and CPU times than RMI. XML documents are bigger than binary data traffic of RMI or TCP sockets; which requires more data to be exchanged over the network. While web services provide low performance, they allow for simpler interfaces in heterogeneous devices.

The framework proposed in this research takes into account the disadvantages of the existing systems and improves the areas of support for heterogeneity, cost, performance and interface definition. S1 and S7 support heterogeneous devices by implementing web services, but the performance is proven to be lower than binary protocols, as the ones implemented in S5 or S6. However the systems based in binary protocols are limited in the deployment to specific operating systems, programming language or physical communication channel. On the cost-benefit aspect, SMS-based systems, as S3 and S6, can be deployed on heterogeneous devices, but the cost of operation is high compared to systems based in Wi-Fi or Bluetooth. The framework proposed and defined in the next chapter makes a combination of the technologies used in the existing systems to improve the deployment of a distributed system on heterogeneous devices; it simplifies interface definition and uses a binary communication protocol.
Chapter 5. PARMI specification

Mobile devices introduce limited capabilities due to hardware restrictions and network access cost, which must be considered during the application development. The design of a framework for mobile distributed systems must also consider these limitations and create an optimal solution.

The proposed framework uses sockets to send messages between clients and the server, and it uses XML to define the interfaces for invoking remote procedures. Since TCP sockets are supported by most OS’s and mobile devices, the higher layers of the application can be adapted to use a communication protocol independent of programming languages or additional frameworks.

The first section of the chapter provides an overview of the challenge this project tries to resolve. The second section defines a mobile device model; this model provides a description of the basic features and restrictions of the same device. The proposed mobile device is then used in the third section of this chapter, as a baseline for defining a mobile distributed system model. The forth section of this chapter provides the interface specification details of the PARMI framework.

5.1. PARMI overview

As described at the beginning of this document, the challenge for this project is defining a framework which enables integration of heterogeneous mobile devices (Figure 68). The purpose of this framework is to communicate any platform, with any OS and any programming language using a common layer across different platforms with low data transfer required.

![Figure 68. Challenge for the proposed framework](image)
The proposed framework (Figure 69) consists in implementing a tool for interpreting an XML-based interface and adding two more layers to the application stack, in order to standardize access to the transport layer in each platform and enable remote procedure invocations:

- Programming language-specific APIs; this layer implements the interface defined in the XML file and uses a generic API framework to access platform specific APIs.
- The second layer implements generic APIs that provide access to the OS-specific APIs.

![Diagram](image)

Figure 69. PARMI overview

5.2. Device model

A new class of computing device is going to be considered for our model. This new device presents limitations in power management and software resources, as described below [58]:

- Limited Energy: A portable device has to run on batteries; however the smaller the device is, the lower will be its power requirements, but wireless communication is energy-intensive.
- Resource constraints: Mobile devices export limited computing capabilities in terms of processor speed, storage capacity and network bandwidth.
- Sensors and actuators: A great variety of sensors and actuators are used in different mobile devices.

The devices considered in this model must support wireless connectivity, such as Bluetooth or Wi-Fi. Several factors should be taken into account for supporting and have a quality of service for runtime state of connection:
• Disconnection: The model considers mobile devices, moving out of the coverage range, which will impact on the state of the connection, and therefore data transfer.
• Variable bandwidth: Variable bandwidth is influenced by the same factors which provoke a device disconnection.

5.3. Mobile distributed system model

One goal of this distributed system development framework is to provide access to services residing on a remote computer (server) across multiple platforms, mixing OSs, programming languages and hardware platforms. This can be achieved by exposing a single set of APIs using an operating system abstraction layer and a communication protocol common between the server and the client. The distributed application stack for the proposed framework is described in Figure 70.

![Figure 70. Distributed application architecture](image)

In the proposed model the server consists of a service based on UDP sockets which receives packets and process them using a protocol which is named “Platform Agnostic Remote Method Invocation” (PARMI). The PARMI protocol is defined in detail in the next sections, by a set of rules for composing a raw data packet which can be encoded and decoded by the server and the client in the distributed system.

5.3.1. Functional/logic view

PARMI is based on sockets for transmitting encoded messages between a mobile system and a desktop server. The specification is based on encoded packets, which can be processed and generated in any platform since they are based on a common interface definition.
Figure 71 shows the layers involved with the PARMI framework. The logic layer interacts with the PARMI interface generated by the interpreter. The interface makes use of the services provided by the PARMI library. The PARMI library is an interface to the OS services for accessing the transport layer, which is implemented using UDP sockets. Both stacks (client and server) implement a PARMI library which provides support for accessing the socket APIs specific for each OS. The PARMI library needs to be developed once for each OS and programming language.

![Diagram of PARMI application stack]

The client application is the main interface with the user; it handles 2 tasks:
- Update the U.I. as response to user input; and when required, it starts a query to the server. The server response is received asynchronously.
- Continuously wait for server callback responses or requests.

5.3.2. Code/module view

Communication between the client and the server is based in sockets (Figure 72); the server can accept multiple connections from different clients, but the client can only connect to a main server. The logic layer calls the procedures in the PARMI functions layer and it also implements the callbacks for each of the PARMI functions; either as a client or as a server. The PARMI library needs to be implemented for each platform and programming language and it exposes a set of generic APIs, which are used by the PARMI functions and the logic layer.
PARMI library initialization routines:
- `SetParmiCallback(ParmiCallback call)`: Initializes the pointer to the callback implemented in the PARMI interface and auto-generated by the interpreter. The callback is executed when a message is received which invokes the right PARMI function callback in the Logic layer.
- `StartListening(int port)`: Initializes the local port and starts a thread to listen for datagrams.
- `StopListening()`: Stops the listener thread.

PARMI library client (mobile device) specific routines:
- `SetupRemoteSettings(String iPAddr, int port)`: Configures IP address and listening port of the remote server.
- `SendUDPMessage(byte[] message)`: Sends a PARMI datagram to the server. Used by the PARMI interface, generated by the interpreter.

PARMI library server (base station) specific routines:
- `SendUDPMessage(byte[] message, String remoteAddress, int remotePort)`: Sends a packet to a remote client. Executed by the server to contact each of the clients. Used in the PARMI interface, generated by the interpreter.

5.3.3. Development/structural view

The client and server applications can be implemented using different programming languages, socket APIs and operating systems, as shown in Figure 73. Where programming language X can be the same or different than programming language P; socket APIs can be accessed through a managed
library or using native services on each entity of the system; and OS Z can be the same or different than OS R.

![Development system view](image_url)

**Figure 73. Development system view**

### 5.3.4. Concurrency/process/runtime view

A simple protocol is followed between the client and the server to execute remote procedures; the sequence is started by the client and executed via PARMI messages (Figure 74). If the Logic layer in the client requires executing the remote procedure `FunctionX`, it would execute below sequence:

- **PARMIFunctions.FunctionX**: The sequence is started by the Logic layer, when invoking a PARMI remote method called `FunctionX`.
- **PARMILibrary.SendUDPMessage**: The PARMIFunctions interface marshals the procedure id and the input parameters and calls `SendUDPMessage()`, implemented by the PARMI library.
- **Server execution**: The server receives the datagram and process the request. It then sends a datagram back to the client.
- **PARMILibrary.ReceiveUDPMessage**: The client's listening thread, which receives a datagram and invokes the PARMIFunctions callback.
- **PARMIFunctions.ParmiCallback**: It checks for the procedure ID, parses the data accordingly and then it calls the `FunctionXCallback` in the Logic layer.
The server processes the call to `FunctionX` (Figure 75), executing below sequence:

- `PARMILibrary.ReceiveUDPMessage`: The server’s listening thread receives a datagram and invokes the `PARMIFunctions` callback.
- `PARMIFunctions.ParmiCallback`: It checks for the procedure ID, parses the data accordingly and then it calls `FunctionX()` in the Logic layer.
- `Logic.FunctionX`: This is the actual function execution, which ends by calling `FuncXReturn()`, with the return parameters.
- `PARMIFunctionsFUNCTIONXReturn`: Marshals the procedure id and the return parameters.
- `PARMILibrary.SendUDPMessage`: Sends a datagram back to the client.
Figure 76 describes the sequence for initializing the PARMi library and the PARMiFunctions:

- PARMiFunctions need to call SetParmiCallback() in the PARMi library, to expose the callback function that gets invoked when a new datagram arrives.
- The logic layer calls SetupRemoteSettings() in the PARMi library to set the address of the server and the listening port.
- When the logic layer is ready to process incoming messages, it calls StartListening(). It can call StopListening() when it needs to stop processing incoming messages.
The PARMI packets are built-up with the information below:

- Function call datagram (Figure 77)

<table>
<thead>
<tr>
<th>Function Call Datagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source IP address</td>
</tr>
<tr>
<td>Procedure ID</td>
</tr>
<tr>
<td>Input Parameters</td>
</tr>
</tbody>
</table>

Figure 77. Function call datagram

- Function callback datagram (Figure 78)

<table>
<thead>
<tr>
<th>Function Callback Datagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server IP address</td>
</tr>
<tr>
<td>Procedure ID</td>
</tr>
<tr>
<td>Return Parameters</td>
</tr>
</tbody>
</table>

Figure 78. Function callback datagram
5.3.5. Thread view

The server application consists of 2 base threads:
- Main thread: This is the application handler and interface with the server administrator.
- Listening thread: Handles all connections started by the clients; it creates a thread for every message received from a client. It accepts and processes every request by creating a socket connection to the client’s listening port. The client threads timeout after X amount of inactivity.

Figure 79 represents the thread view of the server side:

![Server Threads Diagram](image)

**Figure 79. Server threads view**

The client is capable of owning and servicing PARMi procedures, acting as a mini-server for the main server. Both the client and the server listen for datagrams in port 5001. Figure 80 represents the thread view of the client side:

![Client Threads Diagram](image)

**Figure 80. Client threads view**

The client application consists of 2 threads:
- User interface thread: Handles user actions and triggers queries to the server according to the information required by the application.
- Listening thread: Receives responses provided by the server and updates the User Interface (U.I.) accordingly.

5.3.6. Physical/deployment/install view

The server is based on a fixed location with network access to a WLAN. The client is a mobile platform with wireless access to a WLAN. The client and server use the TCP/IP protocol for transmitting PARMi based UDP packets between each other (Figure 81).

![Figure 81. Physical system view](image1)

5.3.7. Data view/data model

The user of the system can be both, source and requestor of data. The data is processed and fetched between the mobile device and the server. The mobile device is only responsible to send PARMi packets to the server and the server takes care of storing, processing or querying data to the storage server (Figure 82).

![Figure 82. Data model system view](image2)

5.4. PARMi interpreter

Figure 83 describes the process for defining the interfaces between the client and the server. An interpreter is implemented, in order to generate stubs and skeletons in multiple programming languages. A single XML-type document is written to generate interfaces for multiple environments.
Figure 83. PARMi Interface definition

The interpreter takes a XML file and generates pseudo code, which is then used to generate code specific to a programming language (Figure 84). The final output is a set of programming language specific API's which can be used to call and response to remote methods.

Figure 84. Interpreter stages

The interpreter takes the XML file and parses it, looking for procedure declarations; for each procedure it generates its identification number, and describes the parameters it receives.

5.4.1. PARMi interface XML-DTD

The format of the XML-based PARMi interface definition is defined in DTD (Document Type Definition). The first version of the PARMi DTD (Figure 85) only supports definition of procedures; a PARMi interface xml file is expected to have a PARMi API element, which supports one or more procedures. Each procedure is expected to have at least one parameter and one return value. A parameter encloses character data, which is the name of the parameter. A return value encloses character data, which is the name of the value. The procedure element has two attributes, its name and the owner. The owner attribute is used by the PARMi interpreter to identify where the function is executed; either in the client or the server and create the skeleton and stub functions in the corresponding interfaces (Table 31). A parameter has 2 attributes, type, which can be one of the following: int, boolean,
string, bytes, void. The second attribute of a parameter is the length, which is expected in bytes. The return parameter has the same attributes as the input parameter, but the type adds one more type: void; which implies, there is nothing to return.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!ELEMENT ParmiApi (Procedure+)>
<!ELEMENT Procedure (Parameter+, ReturnValue+)>
<!ELEMENT Parameter (#PCDATA)>
<!ELEMENT ReturnValue (#PCDATA)>

<!ATTLIST Procedure name CDATA #REQUIRED>
<!ATTLIST Procedure owner (Server | Client) #REQUIRED>
<!ATTLIST Procedure type (int | boolean | string | bytes) #REQUIRED>
<!ATTLIST Parameter length CDATA #REQUIRED>
<!ATTLIST ReturnValue type (int | boolean | string | bytes | void) #REQUIRED>
<!ATTLIST ReturnValue length CDATA #REQUIRED>
```

**Figure 85. parmi1.0.dtd**

Figure 86 shows a template for implementing a PARMI 1.0 interface.

```xml
<?xml version="1.0"?>
<!DOCTYPE ParmiSystem "parmi1.0.dtd">
<ParmiApi>
  <Procedure name="" owner="">
    <Parameter type="" length="">
      ...
    </Parameter>
    <ReturnValue type="" length="">
      ...
    </ReturnValue>
  </Procedure>
  ...
</ParmiApi>
```

**Figure 86. PARMI 1.0 XML template**

<table>
<thead>
<tr>
<th>FunctionX() owned by the client</th>
<th>FunctionX() owned by the server</th>
</tr>
</thead>
</table>

**Table 31. FunctionX() skeleton and stub.**
During the implementation of the model, a new interface was defined, which improves message exchange by encouraging the definition of enums. The second version of the PARMI DTD (Figure 87) adds support for defining enums. An enum element encloses at least one item. Each item element is expected to enclose character data, which corresponds to its identifier. The enum element has the name attribute. The item element has the value attribute, which is expected to define its ordinal number in the list.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!ELEMENT ParmiApi (Procedure+, Enum*)>
<!ELEMENT Procedure (Parameter+, ReturnValues+)>  
<!ELEMENT Parameter (#PCDATA)>  
<!ELEMENT ReturnValues (#PCDATA)>  
<!ELEMENT Enum (Item+)>
<!ELEMENT Item (#PCDATA)>

<!ATTLIST Procedure name CDATA #REQUIRED>
<!ATTLIST Procedure owner (Server | Client) #REQUIRED>
<!ATTLIST Parameter type 
  (int | boolean | string | bytes | enum) #REQUIRED>
<!ATTLIST Parameter length CDATA #REQUIRED>
<!ATTLIST ReturnValues type 
  (int | boolean | string | bytes | enum | void) #REQUIRED>
<!ATTLIST ReturnValues length CDATA #REQUIRED>
<!ATTLIST Enum name CDATA #REQUIRED>
<!ATTLIST Item value CDATA #REQUIRED>
```

Figure 87. parmi2.0.dtd

Figure 88 shows a template for implementing a PARMI 2.0 interface.
5.5. Conclusions

5.5.1. PARMI design constraints

The design of the framework had to use UDP sockets as transport, because the wireless clients are not expected to guarantee they will be in the coverage area for as long as required.

The original design did not consider an efficient way of keeping track of a client’s IP address. The PARMI library needs to keep track of it internally.

Synchronous remote procedure calls in the PARMI framework is not supported since it requires more infrastructures in the PARMI library to handle the synchronization by itself.

The main limitation in terms of deployment of the system is that a PARMI library and the interpreter need to be enhanced for each new system that needs to be integrated; when the programming language is new and not supported by the interpreter or when the operating system is new and the APIs for accessing the transport layer are different.

5.5.2. PARMI versus Web Services

By defining the PARMI interface in XML and having an interpreter generate the stub and skeletons for each of the procedures at compile time, the framework avoids having to translate the XML interface at runtime, improving the system performance. Plus helps eliminating the overhead caused by exchanging XML code with the clients, the datagrams exchanged between the client and the server include only input and return parameters. Using enums allows exchange of numbers only for pre-defined values; there is no need to use strings as parameters.
5.5.3. Future work

5.5.3.1 PARMI interface 3.0

A third version of the PARMI interface is proposed for future work, which will require changes in the design of the PARMI library and the PARMI interpreter. The third version of the PARMI DTD (Figure 89) adds support for defining objects.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!ELEMENT ParmiApi (Enum*, Object*)>
<!ELEMENT Object (Procedure*)>
<!ELEMENT Procedure (Parameter+, ReturnValue*)>
<!ELEMENT Parameter (#PCDATA)>
<!ELEMENT ReturnValue (#PCDATA)>
<!ELEMENT Enum (Item*)>
<!ELEMENT Item (#PCDATA)>

<!ATTLIST Object name CDATA #REQUIRED>
<!ATTLIST ParmiApi owner (Server | Client) #REQUIRED>
<!ATTLIST Procedure name CDATA #REQUIRED>
<!ATTLIST Parameter type
   (int | boolean | string | bytes | enum | objectpointer) #REQUIRED>
<!ATTLIST Parameter length CDATA #REQUIRED>
<!ATTLIST ReturnValue type
   (int | boolean | string | bytes | enum | objectpointer | void) #REQUIRED>
<!ATTLIST ReturnValue length CDATA #REQUIRED>
<!ATTLIST Enum name CDATA #REQUIRED>
<!ATTLIST Item value CDATA #REQUIRED>
```

Figure 89.parmi3.0.dtd

Figure 90 shows a template for implementing a PARMI 3.0 interface.
5.5.3.2 Synchronous calls

Implementing synchronous calls will be one of the major challenges. Two approaches are proposed:
- Modify the interpreter to support the synchronization in the PARMIFunctions layer.
- Modify the PARMI library to handle all the synchronization in the same library.

The second approach requires more work, since it requires supporting interfaces with different input parameters and return values.

For both cases, the interface will need to change to support only one return value, which will slow the communication if there is no support for structures or objects as parameters.

5.5.3.3 Quality of Service

Session: Support for session is implemented implicitly. A client’s session is valid as long as the client keeps responding. The session expires after certain amount of inactivity.

Transport layer access: The framework allows expansion of the PARMI library to support other transport layers, as Bluetooth or 3G.

The framework allows improvements in the quality of service in higher layers of the stack.
Chapter 6. System prototype

As presented in the previous chapters, the development of distributed systems for mobile devices is constrained by the limitations in the hardware of each device; these limitations include, processing and memory capabilities, interfaces for wireless communication and heterogeneous software.

The simplest form of a distributed system is the client-server architecture. In case of the mobile environments, this approach for designing systems is an excellent choice, since it allows for mobile devices to offload their tasks to high performance servers. This chapter describes the implementation of a mobile distributed system based on the PARMI specification.

6.1. PARMI library implementation

A mobile distributed system based on PARMI requires two extra layers in the application stack, one is the PARMI library, which enables access to the platform specific resources and APIs; the second layer is the PARMI interface definition layer, which is generated by the PARMI interpreter based on the standard XML PARMI specification. This section exposes the implementation of the PARMI library for Java and C#; it also describes the implementation of the PARMI interpreter and how to use it.

6.1.1. Client PARMI library implementation

The client prototype was designed to run on a Kindle Fire mobile device using Java and XML for the development of the application. However, the PARMI library must expose the same API’s on every platform and every language.

Figure 91 shows the class diagram for the Java implementation of the PARMI library. It is implemented as a package and defines one class and one interface. The class, named “ParmiLib” exposes the common APIs for accessing the transport layer from the PARMI interface layer:

- Transport layer access methods: ParmiStartListening, ParmiStopListening and ParmiSendUdpMessage.
The server prototype was designed to run on Windows 7 OS using C# and .Net Framework for the development of the application. This PARMi library exposes the same API’s as the client’s library, freeing the PARMi interface layer from platform specific implementations.

Figure 92 shows the class diagram for the C# implementation of the PARMi library. It is implemented as a dynamic link library (Dll) and exposes one class and one interface. The class, named “ParmiLib” exposes the common APIs for accessing the transport layer from the PARMi interface layer:

- Transport layer access methods: ParmiStartListening, ParmiStopListening and ParmiSendUdpMessage.
6.1.3. PARMI interpreter implementation

The interface definition language for PARMI-based distributed systems is expressed using standard XML. The XML interface is a contract between the client and the server and therefore it has to be interpreted by a single entity to generate language-specific interfaces, which can be used by the system developers to integrate heterogeneous platforms.

The interpreter was designed to run on Windows 7 OS and implemented in C# and .Net Framework. It takes an XML file and generates a language-specific code to integrate the logic layer of an application with the PARMI library and enable communication with remote devices.
Figure 93. PARI interpreter modular view

Figure 93 describes the implementation using a modular view of the PARI interpreter. The Program class is the main method, which starts and ends the command line application; it creates and Interpreter object to handle the logic of the application. The interpreter class creates a PARMIApi, two PARMIFunctionsBuilder and two PARMInterfaceBuilder objects to do the conversion from XML to code. The PARMIApi class takes the XML file and represents the interface mapping the XML elements to objects, such as procedures, enums, parameters, return values and enum elements (items). The PARMInterfaceBuilder class takes a PARMIApi object as an input and generates a string with the programming language-specific code to create the PARMICallbacks interface for both, the client and the server. The PARMIFunctionsBuilder class takes a PARMIApi object as an input and generates a string with the programming language-specific code to create the PARMIFunctions layer for both, the client and the server.

6.1.3.1 Sample XML to PARI function owned by the server

According to the definition of procedure GiveMeTrip in Figure 94, the client application implements these functions in each of its layers:

- Logic layer:
  - GiveMeTripCallback(int StartPoint, int EndPoint): This is the routine called with the return parameters for the GiveMeTrip() procedure and gets invoked asynchronously from the server.
- PARMIFunctions layer (generated by the PARI interpreter):
  - GiveMeTrip(int TruckId, int OperatorId): This is the routine called by the client's logic layer when the user requires it.
According to the definition of procedure *GiveMeTrip* in Figure 94, the server application implements these functions in each of its layers:

- **Logic layer:**
  - *GiveMeTrip(int TruckId, int OperatorId):* This is the routine called with the input parameters for the *GiveMeTrip* procedure and gets invoked asynchronously from the client.

- **PARMIFunctions layer (generated by the PARMI interpreter):**
  - *GiveMeTripReturn(int StartPoint, int EndPoint):* This is the routine called by the server’s logic layer in response to calling *GiveMeTrip*. This routine sends the return parameters to the client.

### 6.1.3.2 Sample XML to PARMI function owned by the client

According to the definition of procedure *QueryMileage()* in Figure 95, the client application implements these functions in each of its layers:

- **Logic layer:**
  - *QueryMileage(int TruckId):* This is the routine called with the input parameters for the *GiveMeTrip* procedure and gets invoked asynchronously from the server.

- **PARMIFunctions layer (generated by the PARMI interpreter):**
  - *QueryMileageReturn(int TruckId, int Miles):* This is the routine called by the client’s logic layer in response to calling *QueryMileage*. This routine sends the return parameters to the server.

According to the definition of procedure *QueryMileage()* in Figure 95, the server application implements these functions in each of its layers:

- **Logic layer:**
- QueryMileageCallback(int TruckId, int Miles): This is the routine called with the return parameters for the QueryMileage() procedure and gets invoked asynchronously from the client.

- PARMIFunctions layer (generated by the PARMi interpreter):
  - QueryMileage(int TruckId): This is the routine called by the server's logic layer when the user requires it.

6.2. System implementation

Our prototype uses one Windows XP operating system for the desktop server and an Android emulator for the mobile device. Desktop application will be the Server, and takes care of the computing intensive tasks as well as to provide access to databases.

6.2.1. PARMi interface definition

The procedures implemented in the prototype distributed system are defined in the XML PARMi interface (Figure 97). It uses PARMi interface version 2.0.
PARMI interpreter takes this standard XML file and generates 2 interfaces:

- **Server**: An interface based in C# programming language, implementing (Table 32):
  - Code to invoke callback functions implemented in the server's logic layer.
    - Skeleton functions: Routines owned by the same server.
    - Callback functions: Return functions for routines owned by the client.
  - Stub functions to invoke routines implemented in the client:
    - Skeleton functions: Routines owned by the client.
    - Callback functions: Return functions for routines owned by the server.

- **Client**: An interface based in Java programming language, implementing (Table 33):
  - Code to invoke callback functions implemented in the client's logic layer.
    - Skeleton functions: Routines owned by the same client.
    - Callback functions: Return functions for routines owned by the server.
  - Stub functions to invoke routines implemented in the server:
    - Skeleton functions: Routines owned by the server.
- **Callback functions**: Return functions for routines owned by the client.

### Table 32. PARG functions implemented in server

<table>
<thead>
<tr>
<th>Logic</th>
<th>PARMIFunctions</th>
</tr>
</thead>
<tbody>
<tr>
<td>GiveMeTrip (TruckId, OperatorId)</td>
<td>GiveMeTripReturn (StartPoint, EndPoint)</td>
</tr>
<tr>
<td>NotifyArrival (TruckId, OperatorId)</td>
<td>NotifyArrivalReturn (ParkingSpot)</td>
</tr>
<tr>
<td>NotifyDeparture (TruckId, OperatorId)</td>
<td>NotifyDepartureReturn (TripInstructions)</td>
</tr>
<tr>
<td>QueryMileageCallback (TruckId, Miles)</td>
<td>QueryMileage (TruckId)</td>
</tr>
</tbody>
</table>

### Table 33. PARG functions implemented in client

<table>
<thead>
<tr>
<th>PARMIFunctions</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>GiveMeTrip (TruckId, OperatorId)</td>
<td>GiveMeTripCallback (StartPoint, EndPoint)</td>
</tr>
<tr>
<td>NotifyArrival (TruckId, OperatorId)</td>
<td>NotifyArrivalCallback (ParkingSpot)</td>
</tr>
<tr>
<td>NotifyDeparture (TruckId, OperatorId)</td>
<td>NotifyDepartureCallback (TripInstructions)</td>
</tr>
<tr>
<td>QueryMileageReturn (TruckId, Miles)</td>
<td>QueryMileage (TruckId)</td>
</tr>
</tbody>
</table>

#### 6.2.2. Client application

Figure 98 shows the list of classes implemented in the client application. It also includes the classes implemented in the PARG Library for Android system. The classes of the client application map to the layers proposed in the PARG framework:

- **UI layer**: In this prototype the client’s UI layer is called Prot01Activity and implements all the related methods for updating and responding to and user event.
- **Logic layer**: The client’s logic layer is called Prot01Logic, which handles the core of the application and uses the PARG interface to communicate with the remote server.
- **PARG Functions layer**: This prototype implements a class with the same name; this is the class generated by the PARG XML interpreter.
- **PARG callbacks interface**: This interface is also generated by the interpreter and needs to be implemented by the client’s logic layer.
Figure 98. Classes implemented in the client

Figure 99 shows the U.I. in the client’s application. It simulates a simple mobile device application to be deployed in the truck company case exposed in chapter 1. It provides the user with 3 buttons to query the remote server:

- **Notify Arrival**: It sends the server a PARMI message to notify that the user has arrived to the base station. The server then responds to this message with a callback providing the slot number where the vehicle should park.
- **Give Me Trip**: It sends the server a PARMI message to query for a new trip. The server then responds to this message with a callback providing start and end locations for the new trip.
- **Notify Departure**: It sends the server a PARMI message to notify that the vehicle is leaving the main office. The server responds to this message with a callback providing instructions for the trip.
The application is deployed on a Kindle Fire device from Amazon Company. Its features are listed here:

- 7" multi-touch display.
- 8 GB internal storage device.
- Wi-Fi connectivity.
- Android OS.

6.2.3. Server application

Figure 100 shows the list of classes implemented in the server application. It also includes the classes implemented in the PARMI Library for Windows system. The classes of the server application map to the layers proposed in the PARMI framework:

- UI layer: In this prototype the server’s UI layer is called ParmiServerUI and implements all the related methods for updating and responding to and user event.
- Logic layer: The server’s logic layer is called ServerLogic, which handles the core of the application and uses the PARMI interface to communicate with the remote server.
- PARMI Functions layer: This prototype implements a class with the same name; this is the class generated by the PARMI XML interpreter.
- PARMI callbacks interface: This interface is also generated by the interpreter and needs to be implemented by the client’s logic layer.
Figure 100. Classes implemented in the server

Figure 101 shows the U.I. in the server’s application. It simulates a simple server application to be deployed in the truck company case exposed in chapter 1. It provides the user with 4 buttons to query the remote server:

- **Welcome**: It sends the client a PARMI message to respond to *NotifyArrival* call, providing the slot number where the vehicle should park.
- **New trip**: It sends the client a PARMI message to respond to *GiveMeTrip* call, providing start and end locations for the new trip.
- **Goodbye**: It sends the client a PARMI message to respond to *NotifyDeparture* call, providing instructions for the trip.
• Query mileage: This invokes a method implemented in the client, which returns the current mileage of the truck.

![Figure 101. Server's U.I.](image)

6.3. Deployment of the application

6.3.1. Client-server interaction

A server-owned procedure call is started with the client generating a PARMI message to the server. The logic layer of the client calls a function in the PARMIFunctions layer, which then encodes the message and sends it to the server (Figure 102) using the PARMI library. The server receives the message and passes it to the logic layer of the application (Figure 103).
Figure 102. Client call stack for remote method request.

![Client call stack](image)

**Figure 102. Client call stack for remote method request.**

A server-owned procedure response is started by the UI layer of the server; it calls a “return” function in the `PARMIFunctions` layer and then sends the message to the client using the PARMI library (Figure 104). Then, the client receives the message and passes it up to the callback routine implemented in the Logic layer (Figure 105).

![Server call stack](image)

**Figure 103. Server call stack for remote method request.**

**Figure 104. Server call stack for remote method callback.**

**Figure 105. Logic layer call stack for remote method callback.**

**Figure 106. Callback routine call stack for remote method callback.**
6.3.2. Setting up the emulator

The following explains how to configure an Android AVD to communicate with the host machine. The Android AVD sits behind a virtual router with an IP of 10.0.2.1. To communicate with the host machine, the Android application needs to talk to 10.0.2.2. This IP is mapped to the localhost loopback IP 127.0.0.1. For example, you have a server application running on the host listening to port 7777. The application running on the AVD will need to connect using 10.0.2.2 port 7777.

To pass data from a desktop client to an AVD, it is necessary to set up a port redirect on the AVD router. To do this, connect to the AVD using telnet, the first emulator port is 5554 (telnet localhost 5554). Then use the redir add command to set up the redirect. For example, to redirect incoming TCP/IP connections from the localhost to the AVD on port 5000, use the following redir add tcp:5000:5000. Any data passed to port 5000 on the local host (127.0.0.1:5000) will be redirected to port 5000 on the AVD (10.0.2.1:5000).

6.3.3. Setting up the hardware device

Amazon Kindle Fire can be connected to a PC as a USB mass storage drive with no issues. Windows will install the correct USB driver for it. However, Kindle Fire does not provide USB Driver for Android Debug Bridge (adb) usage. Kindle Fire adb USB driver needs to be installed in order to talk to Kindle Fire from the PC. Android SDK development environment needs to be setup first. Step by Step Guide:

a) Connect USB cable to Kindle Fire.
b) Go to Devices and Printers.
c) Double click on Kindle.
d) Go to Hardware tab (Figure 106).
e) Select Kindle with yellow exclamation mark, and then click on Properties.
f) Go to Details tab (Figure 107).
g) At Property, choose Hardware Ids.
h) That is Kindle Fire USB hardware Id that will be used later.
i) Leave the property window open.
j) Go to My Computer.
k) Then go to C:\Program Files (x86)\Android\android-sdk\extras\google\usb_driver folder.
l) Open android_winusb.inf with notepad (Figure 108).
m) Insert these 3 lines at [Google.NTx86] and [Google.NTamd64].

;Kindle Fire
%SINGLEADBINTERFACE% = USB\Install, USB\VID_1949&PID_0006
%COMPOSITEADBINTERFACE% = USB\Install, USB\VID_1949&PID_0006&MI_01

Figure 107. Kindle Fire Properties Details
n) Save it.
o) Go to C:\Users\YourUserName\.android folder.
p) Open adb_usb.ini with notepad (Figure 109).
q) Key in 0x1949.

```
# ANDROID 3RD PARTY USB VENDOR ID LIST -- DO NOT EDIT.
# USE 'android update adb' TO GENERATE.
# 1 USB VENDOR ID PER LINE.
0x1949
```

Figure 109.adb_usb.ini

r) Save it.
s) Go back to Kindle property window.
t) Go to General tab.
u) Select Update Driver.
v) Select Browse my computer for driver software. Locate and install driver software manually.
w) Select Browse, and select C:\Program Files (x86)\Android\android-sdk\extras\google\usb_driver folder.
x) When Windows Security prompt, select Install this driver software anyway

y) Wait for the success notification to appear (Figure 110).

Figure 110. Adb success confirmation
6.3.3.3 Testing Kindle Fire adb USB driver

These are the steps to test the connection with Kindle Fire adb USB driver.

a) Bring up command prompt (CMD).

b) Type “adb kill-server”, then enter.

c) Type “adb devices” then enter.

d) Kindle Fire should be listed as one of the devices.

6.3.3.1 Setting up a Device for Development

To debug an Android application in real hardware it is necessary to declare the application as debug-capable in its manifest file [59]. When using Eclipse, this step can be skipped, because running the app directly from the Eclipse IDE automatically enables debugging. In the AndroidManifest.xml file, add android:debuggable="true" to the <application> element.

USB debugging needs to be enabled in the device. And the debug host system needs to be configured to detect the device. If developing on Windows, a USB driver for adb needs to be installed.

6.3.3.2 How to enable side loading on the Kindle Fire

Kindle Fire has an option in its settings menu to enable installation of apps from unknown sources. This is known, as side loading. Tap the top-right settings area of the notification bar on the Kindle Fire. Next, tap the (+) More button on the drop-down menu (Figure 111).

![Kindle Fire notification bar](Figure 111. Kindle Fire notification bar)

Tap the Device category in the Settings menu (Figure 112).
Scroll down the screen and find "Allow Installation of Applications From Unknown Sources" tab. Tap the ON button (Figure 113).
6.4. Conclusions

The development of the client prototype was challenging; integrating the UI with the PARMI library required an intermediate layer for handling the logic and interfacing with the PARMI library. Originally, the application only implemented a

With the implementation of the prototype we prove that the PARMI framework defined in this document is valid and supports integration of heterogeneous mobile devices. Enabling the development of mobile distributed systems, such as the one described in chapter 1, for the transportation company (Figure 114).

![Diagram of PARMI based distributed system for a transportation company.](image)

The prototype can be further improved, in several aspects, but it serves the purpose of demonstrating that the design of the PARMI framework is valid and enables the development of a mobile distributed system. The PARMI framework enables the portability of the system to heterogeneous devices by using the standard XML interface definition and the interpreter, which serves a contract between each device in the same system. It also provides a lightweight communication protocol, since it transmits only binary data; this lightweight communication protocol serves the purpose of the model mobile device, which is limited in processing performance and battery life.
Chapter 7. Bibliography


[Accessed 11 September 2010].


