Surname: Chin

Other Names: Jeannette S

Qualification Sought: BSc Internet Computing

Title of the Project: TESA—Towards **E**mbedded-Internet **S**ystem **A**pplications

Supervisor: Dr Victor Callaghan

Date: 1st March 2003

Abstract:

This thesis explores the feasibility of applying emerging low-cost embedded Internet devices in support of *pervasive computing;* a new vision whereby domestic appliances are provided with Internet connections enabling them to be accessed and controlled by any web based interface.

The approach used was to design and build a simple embedded-internet botanical appliance labeled TESA (Towards Embedded-Internet System Applications). The server side of TESA was implemented using the embedded-Internet TINI [25]. The client side of TESA was implemented for PCs, PDAs and mobiles phones based on Web and WAP interfaces with communication mediums that include wired-IP, WiFi and Bluetooth.

The principal computer science challenges were the design of the appliance computing architecture, the development of the multi-mode interactive interfaces and the management of multi-interface consistency. The main outcomes were: the design of an embedded internet appliance, a set of supporting JAVA programs, a solution for multi-mode interface inter-operation, a wireless architecture, and a development methodology for internet-appliances.

A working system was produced and the overall project conclusion was that it is indeed feasible to use low-cost embedded-internet devices, with minimal development systems, to “quickly” create useful applications for pervasive computing products.

This project report is in accordance with Examination Regulations 6.12 and 6.13.

To Ivan, who inspires me, always.

Acknowledgements:

I have many people to thank for their part in making my achievements possible. I would like to start by saying how much I owe to my most extraordinary kind, supportive supervisor and advisor, Dr. Victor Callaghan, who has given me constant encouragement, advice and support throughout my three years at the Essex University. It was him who made me started in believing myself, after my long 13 years away from study, with family responsibilities, difficulties and numerous set backs as I strived to fulfil my "dream". I am extremely lucky that I was under his supervision. I have learnt much and gained a lot of wonderful experiences while we were working together. Moreover, he has opened my eyes from a "square window" view of the world to a socially enhanced technological dreamland!

I would like to offer my most sincere thanks to Dr. Graham Clarke, who, with his most cheerful smiles and wonderful sense of humour, has never hesitated for a single moment to give me advice and support whenever I need it. I would like to thank him for his exceptional kindness and support.

I would like to thank the members of the IIEG group, for providing me this opportunity and the resources that I needed to accomplish my project. In particular, I would like to thank Hakan Duman, who was always so patient in answering my never ending questions and for his helpful comments and ideas concerning my work! Thanks are also due to Dr. Martin Colley and Dr. Hani Hagras, particularly for their support in enabling me to participate in their Korean research work over the summer. I am also very grateful to Malcolm Lear, for his high quality technical support and for responding so promptly to my technical needs, especially in the very early stages of my project. Also I am pleased to acknowledge Anthony Pounds-Cornish who introduced me to the TINI system and Arran Holmes both of whom provided me with a lot of wide-ranging technical advice and support.

I would like to thank my fellow students Elias Tawil, Adam King, Faiyaz Doctor and Gustavo De Souza for making life in the "Sunlab" generally nicer. I would especially like to thank my friend Alvin Khaw, for his sincere encouragement and support, Yin_Young Sik and Hin_Hang Sik for their kind blessings, and a very special thanks to Sue Sharples, for her tender care and support.

Finally, I reserve the biggest and most wholehearted thanks to my family, my parents, brothers and sisters for their extraordinary patience and understanding, not to mention for their constant, immeasurable love and support.

1. Introduction

1.1 Project Aims

The main aim of this project is to evaluate the feasibility of using current of-the-shelf technology to build embedded Internet appliances. The project aims to accomplish this by building a demonstrator based around a botanical application.

1.2 Background Overview

There are few individuals or organizations in the developed world that do not make extensive use of the Internet. Take Britain for example, it is estimated that they are already 10 million homes in the UK have Internet access [Web16]. A recent report from Telewest Broadband estimated that new PCs were bought for the first time by one in nine homes during the first quarter of 2001 and we Britons are now spending as much as 25% of our time "switched on"[SmartVo1]!

It is not just the Internet has become all pervasive; the growth of mobile wireless technology is equally breathtaking. Statistics from a market research company reported that, in the period 2000 to 2001 (inc), worldwide sales for Internet-enabled mobile phones grew from 51 million to 327 million [Garber01]. The BBC recently reported that, in the UK, over 50% of the population now own a mobile phone [Web17]. In part the rapid market growth is fuelled by the descending mobile phone size, weight, and cost; more than 20% per year over the past 15 years! [Harte02].

Whilst these individual markets are impressive in their own right, when these two technologies are brought together there potential is even greater. Wireless (mobile phone) extends the functionality of the Internet by freeing it from wires allowing services such as emailing, web surfing and E-commerce making it an anytime, anywhere technology.

Just as these markets have begun to establish themselves, a new technology and vision have arrived, the embedded-internet. In this vision everyday items such as home appliances contain embedded networked computers that allow them to be connected to the Internet and controlled anytime, from anywhere by their owners.

My project addresses part of this vision by investigating the use of WAP mobile phones, wireless PDAs and Desktop PCs to interface with embedded Internet appliances allowing people to connect to, configure and control their environments. The project is not a complete solution but rather some initial steps *Toward Emended-Internet Systems Applications – TESA.*

1.3 Exemplary Scenario

Tessa, a busy airhostess working for Singapore Airlines, is very much inspired by the Internet technology. She lives on her own in a rented flat in London where she is based. Tessa enjoys her job, because it takes her to many exciting places and sees lots of different things. Tessa loves plants, and her hobby (some of her friends say, obsession!) is keeping plants that are rare and difficult to maintain.

Being alone, and traveling frequently would normally present difficulties for Tessa as there would be nobody at home to look after her plants and flowers while she is away. In addition, as she is living far away from her friends and family, she lacks friends or people who share her interests. As anyone could imagine, Tessa felt heartbroken to see her beloved plants and flowers grew unhealthily (and some even die) when her job takes her away (the nature of the job meaning that sometimes, and unpredictably, she is away for much longer than she expects!).

As Tessa is living in the "Internet Age" and inspired by the technology, she came across a circle of like-minded people when she was surfing the World Wide Web one day. She found the "TESA circle " on the Internet that seemed to be an e-commerce company selling stylish Internet-based systems for caring plants. The advertisements stated that from any web access point, or her mobile phone, could "see" and care for her beloved plants while she was away traveling. Moreover, the circle seemed to operate like a group of friends exchanging ideas on how they keep their rare plants. Tessa was thrilled and in no time, she made friends with similar minded people and bought herself a "*TESA*" the Embedded-Internet plant care system. Having bought the system, TESA also found that the "TESA circle" allowed her to see other people's plants, comparing notes with them and establishing many lasting friendships as well.

Some time later, Tessa was on another trip abroad. As she stepped out of a taxi in Singapore airport the following fortnight, and watered her beloved plants back in London from her mobile phone, she wondered to herself "what must life have been like in the dark ages, before the embedded-Internet?"

1.4 Project Overview

As explained in the above, the goal of my project is to develop a *simple demonstrator* to illustrate how technologies such as Web, mobile phones & wireless PDA can be developed to produce and interact with an embedded Internet product. An exemplary application, a botanical system called TESA, (Towards Embedded-Internet System Applications), based on a development of the department's mDorm (an embedded internet development environment). The intention was to develop a novel internet-appliance in keeping with the new vision being offered in this project.

The diagram below shows the main component of my project:

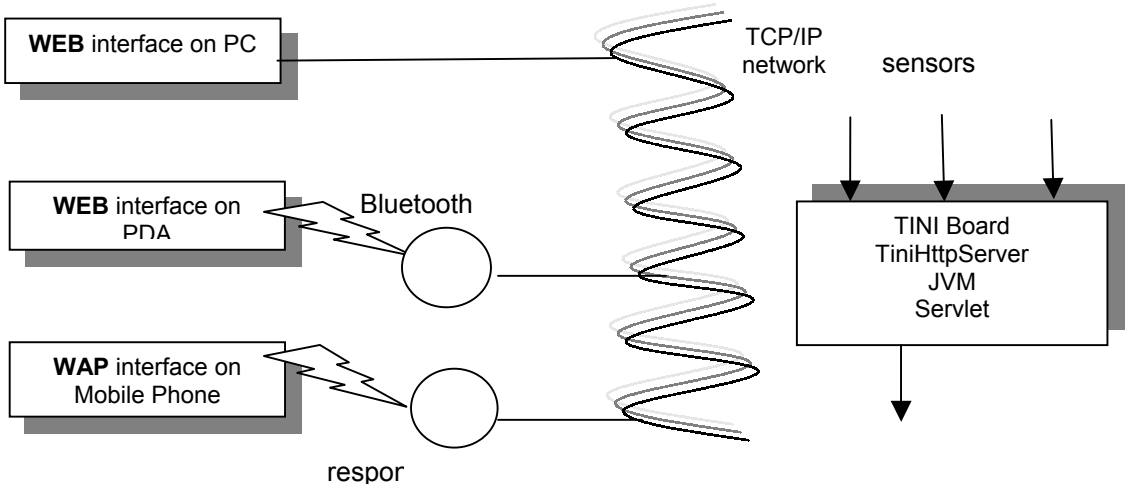

Figure 1. Overall View of the "TESA" architecture

The above system was developed in a stepwise way. The following is a brief overview of work that was carried out in this project.

Firstly, for the development platform I used one of the department's mDorms (ID: borg8of9) minimally modified to make it mimic a botanical system. Only a moisture sensor was being added to the standard parts, as it was not intended to be a commercial product, rather an evaluator. I then wrote the Web server components to enable the devices, which mostly based on field-bus standard called 1-wire, to be controlled from an external client.

Secondly, I investigated how a client Web interface could be created for a PC, PDA and mobile phone. These devices are very different in nature both in terms of size, graphic capability and language they use (eg HTML and WML). In addition, the underlying networks were different being Internet Protocol (IP) for the PC, Bluetooth for the PDA and General Packet Radio Service (GPRS) for the mobile phone. The different Web protocols and network standards presented demanded that the server should respond in a different way depending on the client platform.

The remainder of the thesis will elaborate on the background, specification, deign, implementation and evaluation of the above system,

1.5 Organisation of thesis

The thesis is organised as follow:

- Chapter 1 presents the project vision, objective and views with an exemplary scenario.
- Chapter 2 describes other projects and related work.
- Chapter 3 provides an in-depth description of the technologies used.
- Chapter 4 provides the TESA user requirement and specification.
- Chapter 5 describes the TESA system design.
- Chapter 6 explains the software implementation.
- Chapter 7 presents the software testing & system Evaluation.
- Chapter 8 summarises the main project achievements

2. Background

The general area addressed by this project has been described by a number of national and international research programs such as the EU's Disappearing Computer programme [Web33], the DTI's New Wave Technology [Web34] and the EPSRC Equator [Web35] program. Many large industrial companies such as Orange [Web10] and Philips [Web36] are investing large sums of money into research in this area, and have built test beds that are similar to the Essex iDorm [Web37].

Concerning technology, the project is based around a commercial embedded-Internet device. There are numerous such devices available [Web47], [Web48], the best example and the cheapest being the Tiny Internet Interface (TINI) [Web19], a platform developed by Dallas Semiconductor [Holmes02], [Cornish02], [Vial01].

2.1 The Pervasive Internet

The Internet can be viewed as a gigantic global network; a composite of a massive number of computers inter-connected together in a loosely structured way using a common communication language protocol called TCP/IP.

Having started life as a military network, the Internet found its way to wider public use in the late 80s or early 90s, being used for email and information serving. Information from the organisations such as the academia, government, commercial companies or even private homepages is sitting on various computers, waiting for someone's retrieval. However, few things in this life remain unchanged and the Internet of today is a very much different story!

Within the home it is now commonplace for appliances to include embedded-computers. It is thus a small step for such computer based appliances to include a network connection; a direction being promoted by numerous companies (eg Echelon [Web51], Siemens [Web52], CISCO [Web53] etc) and organizations (EHSA - European Home Systems Association [Web54], BACnet [Web55] etc). With the arrival of the embedded-Internet, Internet accessibility had been liberalised from solely PCs to *any* form of appliance. In addition Internet accessibility is no longer restricted to wired networks, but now includes wireless technology enabling the Internet to be accessed *remotely, "on the move"*, from anywhere in the world.

As the Internet has begun to embrace domestic appliances, a new vision is emerging whereby such technology can be used to allow people to connect to, configure and control the environments they inhabit! It is envisaged that such environment would be constructed from a new generation of Internet appliances [Web5] examples of which are [Web40], [Web9], iPot [Web58] and WAP enabled phones [Web10], [Web11]. Innovation is one of the aspects that is fuelling this market and new Internet appliances are being developed continually. A good example being a company called LG that has transformed a normal domestic fridge into an Internet fridge [Web5] that, amongst other possibilities, would allow a user to log on to the Internet and chat with a friend while preparing a meal. A further stimulant to the Internet market is the arrival of broadband communications ,which brings with it better performance. To give an indication of the size of these markets at the beginning of 2001, more than one out of 10 people in the world (over 680 million customers) had a mobile phone [Web12] and in 1999 there were 200 million world wide Internet users, compared to 3 million in 1993 [Web56]

2.2 Ubiquitous Computing

Some ten years ago, Mark Weiser introduced the term *ubiquitous computing* describing it in the following statement: "*For thirty years most interface design, and most computer design, has headed down the path of the "dramatic" machine. Its highest ideal is to make a computer so exciting, so wonderful, so interesting that we never want to be without it. A less traveled road I call the "invisible"; its highest ideal is to make a computer so embedded, so fitting, so natural, that we use it without even thinking about it.*" [Weiser88].

Essentially this described a vision for living environments populated with "computerized" objects where the emphasis was on greater user-friendliness, more efficient service support, user-empowerment, and support for human interactions [Weiser 91]. In many respects it seems that this vision well on the way to becoming a reality as networked devices such as mobile phones, PDAs net-Cameras, iFridges, iPots, iMicrowaves etc proliferate. More radically, research projects such as the EU FiCom see this vision spreading into hitherto "silicon free" items (e.g. garments, chairs, cups etc). Some evidence for the view that the pervasive computing era is closer than people may suspect can be found in a statement by Robert Metcalfe, Ethernet inventor and 3Com Corp founder speaking at the 2001 ACM1 conference in the USA who is reported as having stated: "*8 billion microprocessors will be produced this year (2001), but just 2% of them will go into PCs. Most will end up as part of that all pervasive fabric of computing that's*

being woven around and through our lives via a wide range of devices, some of which we don't even recognize as computers." [Metcalfe 01]

Thus the belief of those working in Ubiquitous computing is that computers will gradually find their way out of the existing computing infrastructure and blend in to our society. Sensors and effectors will be embedded into our environments, together with mobile computing and communication technology. Users will no longer be forced to interact with the technology but rather the technology will work seamlessly with users' and their social interactions without it being realised! Ubiquitous Computing works can be found at [Web49], [Web57].

2.3 Related Work

Numerous organizations are engaged in research in this area. In many respects, this project can be regarded as a further development of the Internet-appliance paradigm the best example being LG, a Korean manufacturer of domestic appliances, mentioned above. It is also linked to communication as that is a critical part of mobile interfaces. Most mobile phone companies are operating projects in this area. For instance, Vodafone [Cameron02] is planning to deploy services such as "Find-a-friend", "Find-a-date" and "Find-a-service" shortly. Similar projects such as finding the nearest Pizza delivery shop can already be seen at [Web38]. Another mobile phone company Orange is investigating using wireless phone to control the devices at home [Web6]. Likewise Nokia [Web32] and Ericsson [Web39] are working on similar technology.

The following is a sample of related projects:

- **Internet alarm clock** - checks the conditions of the road traffic over the Internet in the early hours of the morning using this information to modify the time of the "wake up" alarm signal [Web40].

- **Communicating Thermostat** - a telephone remote home control system that provides current temperature readings from anywhere in the world via touchtone phone [Web26].

- **Touchtone Controller** – an X-10, Telephone Responder which provide the capability to control devices such as lights, appliances and thermostats in an environment from anywhere in the world via telephony system [Web27].

- **All-in-One Controller**, a Compaq iPAQ that, via IR, controls a myriad of appliance appliances. The prototype handheld has so far been used to control two lamps, a fan and a stereo with a five-CD changer. This is a collaborative research project between Maya Designs Inc.and Carnegie Mellon University [Web41].

- **The** *intelligent aquarium* **-** called "Octopus" developed by the Sussex-based company called Casco [SmartVo2]. The aquarium is capable of automatically control and monitors its own environment, and can be accessed for checking the system status from anywhere in the world with an Internet access. Although it doesn't have an Internet connection, in all other respects it mirrors much of the vision of the botanical care system at the heart of this project. Therefore it is described in more detail below.

2.3.1 Brief overview of the "Octopus":

The intelligent aquarium system controller performs "fish-care" for the owner by carrying out some predefined tasks such as changing the water automatically, controlling the wave cycle, lighting and monitoring the status of the water. It is also designed to solve any minor problems, such as correcting any imbalance in the water states (for example PH level) without involving the owner. However, should the problem persist after the correction, the system will inform the owner know by paging him immediately.

Figure 2. The Octopus

Whilst the aim of the appliance is similar to this work (ie harnessing cutting edge technology to care for living entities, being eye catching and novel), it differs significantly from TESA in that it doesn't provide any Internet connection.

2.4 Summary

There is widespread evidence from international research programs and companies that the embedded-Internet is envisaged as being a potentially massive market. Already, there are several commercial applications being piloted, even though the underlying technology is still not fully developed. Currently there is much research being conducted into the technology that will enable this vision to be realised. This project aims to form part of the initial steps that might one day lead to this vision being commercially realised.

3. Technology Used In Project

The project involved a number of technologies, from the underlying hardware, through communication, to the high level software technologies. These technologies are described below.

3.1 TINI Technology

Tiny InterNet Interface (TINI) is a small single board computer developed by Dallas Semiconductor [Web19]. It facilitates both local and remote control (the latter via an in-built IP network interface).

The TINI platform is a combination of a small chipset and a Java™-programmable runtime environment. This chipset provides the TINI processing control, device-level communication and networking capabilities. The features underlying the TINI hardware are a set of Java application programming interfaces [23] that allow the high level software application coordinate with the low level hardware devices. TINI is also refers to both the TINI chipset and the TINI board.

The TINI CPU is not from the dominant PC companies such as the Intel or AMD; but rather from a small semiconductor manufacturer in the USA called Dallas Semiconductor. The TINI chipset is 1.25” x 4.0” Single Inline Memory Module (SIMM), small enough to fit almost anywhere. The processor chip is referred by way of a plain number, DS80C390, rather than a name [Web18].

Figure 3 The TINI

3.1.1 TINI Concept

The TINI's concept is to integrate, processing, networking and input/output (I/O into a small, low-cost system thereby enabling it to be used to connect everyday embedded computers system to the Internet. The cost of a full TINI development system is less than $50 with chips for production products being available for just a few dollars (the exact price being dependent on quantity).

TINI system provides a parallel I/O bus (expandable), serial-port (RS232), field-bus (CAN, 1-wire), TCP/IP network protocol stack and an object-oriented programming environment (Java). Its network connectivity enables the interaction with clients such as remote systems or users through a web browser. The TINI concept is illustrated in the following diagram.

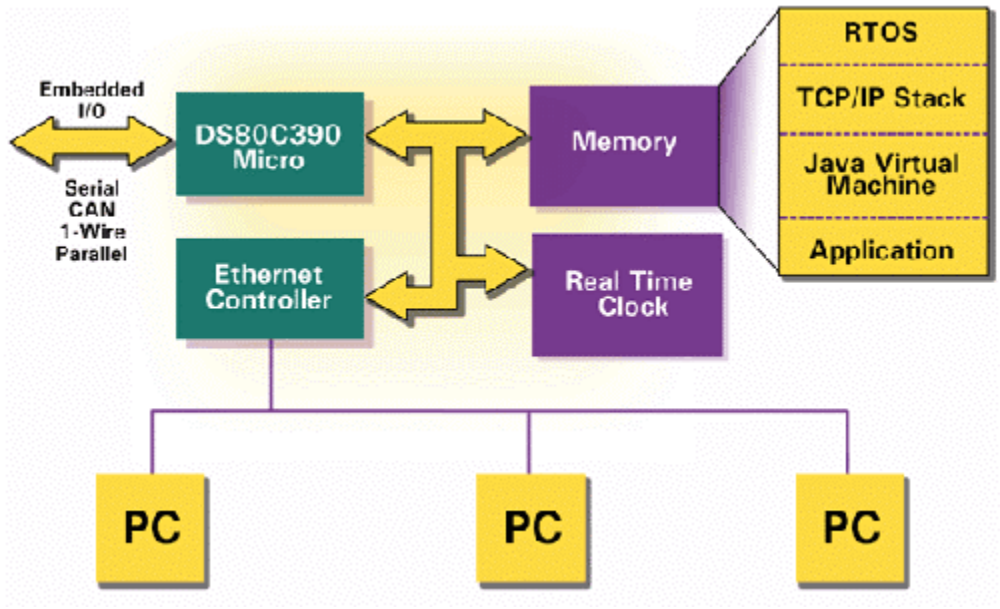

Figure 4. The TINI concept [Web19]

3.1.2 TINI Interface

The TINI system is essentially contained and sold as a single SIMM device. Apart from a single edge connector that connects the device with a SIMM interface, it has no other built-in sockets and therefore cannot be interfaced with any external devices without additional hardware. For this reason, a special dedicated socket board is required for the TINI to interface with external hardware devices. There are a number of such TINI socket board options around such as STEP socket [Web20], Proto adapter [Web21], and Nexus [Web22], but the one used in the project was the E10 socket board, from the Dallas Semiconductor.

The E10 socket board has the following interfaces [Loomis01]:

- 72-pin SIMM connector- interfacing with the TINI board
- 9-pin female DB9 connector- interfacing with Data Communication Equipment (DCE) serial connections
- 9-pin male DB9 connector- interfacing with Data Terminal Equipment (DTE) serial connections
- RJ145 –interfacing with a standard 10Base-T Ethernet connections
- RJ11-interfacing with 1-Wire network connections
- Power Jack-interfacing with +5V DC power supply

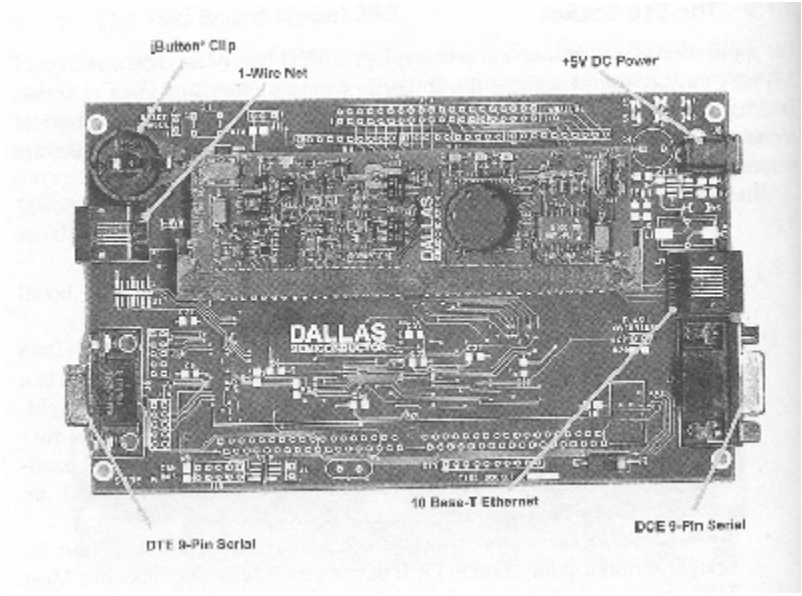

Figure 5. TINI E10 socket board [Loomis01].

3.1.3 TiniHttpServer

The TiniHttpServer [Web23], written by a group of software engineers from Smart Software Consulting, is a special dedicated multi-threaded Web server developed for the TINI that supports Java Servlet technology. TiniHttpServer can be downloaded at Smart Software Consulting site, licensed under the GNU General Public License. It is capable of transforming the TINI into a Web server with server-side programming capabilities. The TiniHttpServer makes a reasonably good website application tool, which can, for instance, be made to serve an application applet, HTML documents, and other files directly from the TINI.

Because, in the early to middle stages of the project, the TiniHttpServer was the only available public source it was employed as the project backend communication engine (since then there has been 1 other private source released).

3.1.4 TINI Constraints

Because the TINI is deliberately design to be a low-cost computing system there are many constraints such as restricted memory capacity (eg 0.5 to 1 MB, depending on the model). For this reason, the TINI firmware, distributed by the Dallas Semiconductor site, only supports a sub-set of the Java Application Programming Interface (API). TINI has many limitations [Web24], the most relevant of which are listed below:

- **Threads**
  - All threads run at the same priority.
  - Threads can block on input/output (I/O), increases CPU cycle availability to other threads and processes.
  - The TINI operating system limits the number of processes to 8, with 16 threads per Java process.

- **Memory**
  - Using normal I/O debugging ie. System.out.println(); consumes a lot of TINI memory.
  - The garbage collector starts automatically when the memory dips below a certain threshold, but major garbage collection during the

program run will cause the collector to run for long periods of time in the background.
- Native modules cannot be larger than 64k.
- The maximum size of any array is 64k.
- Using String concatenation "+" operator consumes a lot of TINI memory.

- **Networking**

  - TINI does not support Internet Protocol (IP) datagram fragmentation/reassembly.
  - TINI only allows 24 socket connections.
  - TINI does not support IP layer routing.

- **Java Classes**

  - TINI does not support serialization.
  - TINI only support a subset of reflection.
  - A class file is limited to 255 static fields (including all super classes' static fields) and 255 instance fields (including all super classes' instance fields).
  - A class is limited to 127 methods (including all super classes' methods but excluding native methods)
  - A class file is limited to 255 native methods.
  - A method is limited to 63 local variables.

- **File System**

  - Each converted class file cannot be large than 64k.
  - Directories can only hold 254 files.
  - An IOExceptioin will be thrown when attempting to create files with names longer than 247 characters.

3.2 The mDorm

The mDorm is an embedded Internet development environment. It was developed in the late 90s by the team of Intelligent Inhabited Environment Group in the Computer Science Department and is based on TINI technology.

Figure 6. The mDorm

3.2.1 The mDorm Physical Characteristics

The mDorm has a stylish solid aluminium look. Physically it takes the form of a small 12” (approx) cube made up of five panels and two doors (see figure 6). The mDorm features two doors, one transparent (tinted glass) the other solid (aluminium) whose use is dependent on the application (whether light needs to be excluded or not). The inside consists of an open environment with some space being taken up to house various electronic components (eg light, heat controls and a TINI). The back panel of the mDorm has a ~2” diameter ventilation hole at one end which allows air to be circulated when the fan is switched on. A main power supply socket and a network connector interface were also integrated at the bottom of this panel.

Inside the mDorm there is a TINI fitted in the top right corner, interfacing with an E10 socket board. Beneath the TINI is a small aluminium box housing the power supply main socket and few other electronic components. A black box (housing a heater and fan) is hanging down from the roof on the top left corner, leaving with a tiny ventilation gap between itself and the roof (top panel). A total number of four small light bulbs that connected to the TINI are attached to this black box. Two of light bulbs are fixed on the top right corner, namely the “toplights”, and the other two are secured at the bottom part of the box, namely the “bottomlights”. The main chamber also contains a temperature sensor.

3.2.2 Sensors/ Effectors

The sensors of the mDorm are:

- Temperature sensor
- Heater sensor
- Top lights sensor
- Bottom lights sensor

The effectors of the mDorm are:

- Heater
- Fan
- Two top lights
- Two bottom lights

3.3 WAP enabled mobile phone

Wireless Application Protocol (WAP) mobile phones are a fairly a recent technology that, in addition to voice communication, are capable of sending and receiving data. WAP technology makes an attractive alternative for services such as email, web surfing and mobile entertainment.

The SonyEricsson t68i, WAP mobile phone is used in this project (see figure 7).

Figure 7. The WAP mobile phone used in the project --Ericsson t68i.

3.3.1 WAP Concepts

WAP is a global standard for bringing Internet content and services to mobile phones and other wireless devices. The standard is maintained by an industry consortium called the WAP forum. The concept of WAP is loosely based on Internet protocols such as Hypertext Transfer Protocol (HTTP) and Transmission Control Protocol/Internet Protocol (TCP/IP), in that, the protocol functions are sliced into layers, and each layer is only concerned about its own function as well as only communicates with the layer immediate above or below it. This concept makes the protocol layers easy to implement, independent and portable (the concept of HTTP and TCP/IP will be explained in detail later). WAP has 5 layers with the following functionalities:

- Wireless Application Environment (WAE) layer- contains the languages for markup (WML), scripting (WMLScript), telephony functions (WTA) as well as Push standards.
- Wireless Session Protocol (WSP) layer-allows efficient binary encoding of previously defined content types (to save bandwidth) and supports various flavours of push applications (where a server initiates a session with a client).
- Wireless Transaction Protocol (WTP) layer-provides a reliable transaction transmission protocol similar to TCP/IP, but optimized to allow reconnection and to minimise handshaking (to reduce network traffic).
- Wireless Transport Layer Security (WTLS) layer-defines security mechanisms for public key encryption and authentication, use of digital certificates and support for compression.
- Wireless Datagram Protocol (WDP) layer- specifies a low-level protocol for rapid packet transmission (implemented based on the User Datagram Protocol (UDP) layer in the Internet Protocol).

3.3.2 WAP constraints

As the WAP is designed primarily for mobile computing, it also inherits its constraints and limitations which are mainly related to the nature of the mobile devices. These are:

- Limited processing power and memory.
- Limited battery life and power.
- Very small displays.
- Limited data input and user interaction capabilities.
- Limited bandwidth and connection speeds.
- Frequent unstable (lost or poor) connections.

3.4 PDA

3.4.1 PDA Concepts

Personal Digital Assistants (PDAs) are portable handheld computers containing applications such as email, word processors etc. They can access to the Internet via either a dial-in modem or Wireless LAN. Location based connection is also possible for the PDAs with integrated Bluetooth technology which allows them to establish local wireless connections with other Bluetooth devices near-by.

PDAs are growing increasingly popular. Some market experts predict that in 10 year times [Web25], personal and portable devices such as the PDAs, cell phones, or tablet [Web28] computers, will be the most popular form of information access appliance. For example, analysis from a market research firm has predicted that such devices would experience a significant increase in sales (from 757,000 to 1.3 million) in the next two years [Garber01].

In this project we have chosen to use the, Compaq iPaq 3970 PDA. This is based around the Microsoft operating system (WinCE) which supports the standard Hypertext Markup Language (HTML) and has a built-in Bluetooth wireless interface, was used in the project.

Figure 8. The iPaq 3970

3.4.2 PDA Constraints

PDAs can be regarded as miniatures of PCs but have many constraints and limitations compared to their PC counterparts, the main ones being as follow:

- The screen is smaller (240* 320 pixels).
- Processing power is low
- Limited battery life power.
- Physical memory capacity is low (a few megabytes at best).
- Does not support full scripting (depending on the operating system).
- Does not support style sheet (iPaq 3970).
- Display resolution varied.

3.5 Desktop PC Software

Listed below are the software tools used in the development of the project.

3.5.1 Java(tm) Communications API

Java(tm) Communications API is a set of Java low-level classes used for reading and writing to serial ports. The package was needed when setting up the TINI for the first time so that the TINI firmware could be loaded from development platform machine (refer as WinME machine for the rest of the thesis) to the TINI platform via RS232 interface (serial port).

There are 3 levels of classes in the Java(tm) Communications API [Eisenreich03]:

- High-level classes that manage access and ownership of communication ports
- Low-level classes that provide an interface to physical communication ports.
- Driver-level classes that provide an interface between the low-level classes and the underlying operating system. Driver-level classes are part of the implementation but not the Java communications API.

Java(tm) Communications API is a free source and can be downloaded at [Web4]. The package was downloaded and unzipped into the WinME machine JAVA_HOME directory C:\jdk1.3\. Below were the sequences performed in WinME machine.

Three files from the package were copied to WinME machine's Java environment and JVM directory. Those sequences were:

- Copy the win32comm.dll file from the commapi to C:\jdk1.3\bin directory.
- Copy the win32comm.dll file from the commapi to C:\jdk1.3\jre\bin directory.
- Copy the comm..jar file from the commapi to C:\jdk1.3\lib directory.
- Copy the javax.comm..properties file from the commapi to C:\jdk1.3\lib directory.
- Copy the javax.comm..properties file from the commapi to C:\jdk1.3\jre\lib directory.
- Set comm..jar onto the system classpath (command : SET CLASSPATH=C:\jdk1.3\lib\comm..jar;)

3.5.2 TINI sdk

TINI sdk is the TINI software needed for it to communicate with its hardware devices. It is a set of software libraries and utilities, distributed by Dallas Semiconductor.

The version tini1_02d was used in the project. The software package was downloaded from Dallas Semiconductor site [Web44] to a temporary directory in WinME machine and later installed in a directory called “tini”. This directory was the project TINI_HOME directory.

An environment variable TINI_HOME was added to WinME machine’s autoexe.bat file (command: SET TINI_HOME C:tini\tini1_02d) and a file, tini.jar, was set in the classpath as well. (command: SET CLASSPATH=.;C:\jdk1.3\lib\tools.jar;c:\tini\tini1.02d\bin\tini.jar;)

3.5.3 TINI Convertor

In normal PC environment, when a Java program is written and compiled error free, the PC compiler will produced a .class file for that program, which can be understood and interpreted by the system Java Virtual Machine (JVM). However, unfortunately, this .class file cannot be understood by the TINI JVM. For this reason, an extra interpretation phase is needed, that a TINI Convertor is required to convert the .class file into a “.tini” file which is the format a TINI JVM understands.

The TINI Converter comes with the TINI software distributed by the Dallas Semiconductor. It can be found in the tini.jar package. It’s primarily job is to combine and convert the .class files into a single .tini file that can be executed by the TINI JVM. The command for the TINI Converter is:

Java –cp %TINI_HOME%\bin\tini.jar TINIConvertor

3.5.4 1-Wire API

1-Wire API is a package of Java classes primarily used for the TINI to communicate with 1-Wire (will be explained later) devices. It is a freeware package and can be downloaded at Dallas site [Web3]. The sequence of installing 1-Wire API for the project was:

- The "1-Wire API for Java" (owapi0_01.tgz) was downloaded and installed in WinME machine's TINI_HOME directory.
- Then the OneWireAPI.jar file was added in to the system classpath by using the command: SET CLASSPATH==.;C:\jdk1.3\lib\tools.jar;c:\tini\tini1.02d\bin\tini.jar;C:\tini\owapi\lib\OneWireAPI.jar).

3.5.5 ANT and TINIAnt

Ant [Web2] is a portable project management tool for Java projects. It replaces the system-specific "build" scripts and "makefiles". TINIAnt [Web1] is an extension of Ant that simplifies the job of "building" TINI applications.

The project made used of both of these tools when porting the files from WinME machine to the TINI platform for execution. Both software packages were installed in WinME machine "tini" directory. An environment variable "ANT_HOME" (command: ANT_HOME=C:\tini\ant) was added to the system environment as well as in the system path (command: C:\WINDWS;C:\JDK1.3\BIN;c:\tini\ant\bin;).

3.5.6 Java Development Kit

Java is a registered trademark of Sun Microsystems [Web4]. It is a freeware, distributed in a form of Software Development Kit (SDK) and can be downloaded at Sun's site. The distributed Java SDK includes the Java compiler, Java debugger, a number of development tools and the Java Runtime Environment (JRE). The JRE consists of the Java Virtual Machine (JVM), the Java platform core classes and supporting files.

Because the TINI platform Java environment in the project was based on the Java version 1.3, for the reason of compatibly, the WinME machine's Java environment must also had to base on the same Java version in order for the software to work. Below is the sequence of installing the Java for the project:

- JSDK Version1.3.1 was downloaded from the site [Web4] and installed in WinME machine's directory called "jdk1.3". That directory was referred as the project JAVA_HOME directory.
- An environment variable JAVA_HOME was created (SET JAVA_HOME=C:\jdk1.3) as well as set the directory to the system path (SET PATH= C:\jdk1.3\bin).

3.6 Interface Standards and Tools

3.6.1 HTML

Hypertext Markup Language (HTML) is a form of markup language, developed by Tim Berners-Lee, for providing user interfaces and delivering information across the Internet. HTML is read and interpreted by user agent such as a Web browser.

The Web and PDA interfaces for this project were both written in HTML. The Web interface was enhanced with client-side language JavaScript.

3.6.2 WML

Wireless Markup Language (WML) is another type of markup language but primarily used for providing user interfaces for WAP enabled devices. Unlike the HTML, WML divides and organises its user interface into decks of cards. It also allows variables to be shared across a deck. However, because the screen on mobile devices is tiny, WML only allows one card to be displayed on a device at a given time. Therefore a navigation option (eg a menu) is needed for a user to navigate WML documents without the need for knowing how a particular card is laid out [Banett01]. Because WML is primarily used for WAP enabled devices, it requires a "live" validation by the industry consortium WAP forum. The project WAP interface was written in WML.

3.6.3 WAP editor

WML requires its own editor tool for editing, syntax checking, debugging, display, execution and content serving. There are many WML software development kits (SDK) tools available. Most of these SDK tools are provided by the mobile phone companies (such as Nokia, Ericsson, Motorola and Phone.com) for their WAP software developers. In addition, these SDK tool kits usually come with WAP emulator, a convenient and economical way of testing and debugging WML program, as using WAP phone in real time testing could be very costly. The project used the Nokia SDK (the tools was downloaded free after registering with Nokia) for editing and debugging [Web32]. A variety of phone emulators were used such as phone.com [Web45] and M3Gate [53] for testing which were also downloaded free from their sites.

3.7 Network Technology

A network is two or more computers connected together for sharing information as well as resources. Usually cables are used to link these computers together.

Networks are divided into categories based on their sizes. These categories are:

- Local Area Networks (LANs)- these are privately owned networks. The size is normally within a single building or a campus up to a few kilometres.
- Metropolitan Area Networks (MANs)- these networks are a bigger version of LAN. They may cover a group of nearby offices or city. Normally MANs' technologies are similar with the LANs. MANs are either private or public owned.
- Wide Area Networks (WANs)- these networks cover a geographical area, often a country or continent. They use the LAN's technology as well as satellite, telephony or ground radio technologies.

Intially most networks were wired but recently, with the advent of mobile devices, wireless networks such as the WiFi (IEEE802.11) and Bluetooth technology have found their way into networking.

3.7.1 Ethernet

Ethernet, as mentioned above, is the cable that connects 2 or more computers together, through a specific interface called Ethernet adapter, which enables them to communicate by allowing their data to be transferred to/fro each other.

Ethernet is divided into grades based on its data transmission rate. The most common Ethernet are: 10 Mbps, 100 Mbps, and Gigabit Ethernet. However, depending on the data rate, Ethernet signals can only be streamed without being distorted/corrupted in a certain distance range. Therefore, “repeaters” or “hubs” are used in the Ethernet for this purpose, to allow the data flowing to a wider distance.

The standards of the Ethernet published by IEEE are:

- 802.2 - The new message format for data on any LAN
- 802.3 - Hardware standards for Ethernet cards and cables
- 802.5 - Hardware standards for Token Ring cards and cables

The project’s networking environment was a high-speed Ethernet research network that had restricted access and controls by the personnel in the research group of the department.

3.7.2 1-Wire

1-Wire is a bus technology developed by Dallas Semiconductor. It is used primarily for the communication between the electronic components, called 1-Wire devices, and the processor. As the name suggests, the 1-Wire bus consists of only one signal line, plus a ground.

The concept of 1-Wire technology is “there is only one master and one master only” whereby the communication always begins with the master (eg a TINI) talking to slaves (eg switches) . Only a certain type of slaves is allowed to interrupt this process for some exceptional need. In TESA the 1-Wire “master” was TINI, and the “slaves” were:

- A temperature sensor
- A moisture sensor
- Top-lights
- Bottom-lights
- A fan
- A heater

3.7.3 WAP and WAP Gateway

WAP enabled devices use WAP browsers to request/send information to WAP servers in the same way as Web browsers communicate with Web servers. In fact, although the language used for these 2 servers is a different, WAP enabled devices are able to communicate with either. The following diagram illustrates a WAP device communicating with a WAP server.

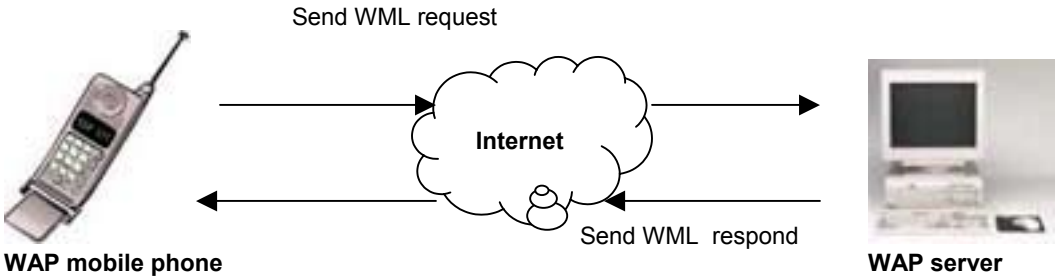

Figure 9. A WAP mobile phone communicating with WAP server

As mentioned above, because the fundamental language used between the WAP client such as WAP-enable mobile phone, and the Web server is different, therefore a gateway (WAP gateway) is required for the communication. A WAP gateway acts as an interpreter between a Web server and WAP devices. The diagram below illustrates this process.

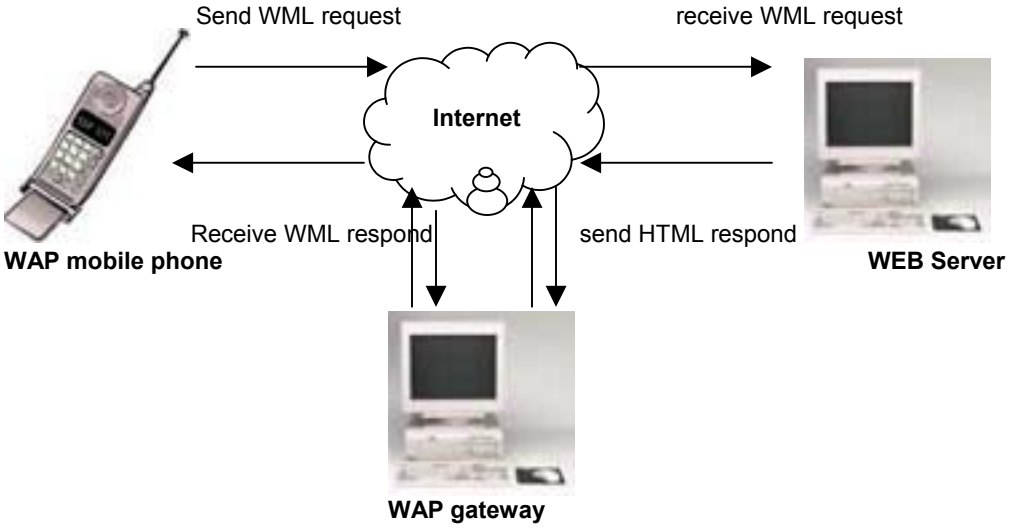

Figure 10. A WAP mobile phone communicating with WEB server

3.7.4 Wireless Technology

The Wi-Fi standard, also known as Wireless Fidelity or 802.11, is on its way to becoming the primary way devices connect to the Internet wirelessly [Walker02].

Over the years, wireless technology and its products have become smaller, faster and inexpensive. Nowadays people not only use their mobile phones for voice communication with their friends and family but also use them to surf the Internet, wirelessly! Wireless technology is growing "smarter" too. Together with sensory technology and computation, wireless technology can be found in increasingly diverse and unusual places such as concrete [Web29], fabrics [Web30], buildings [Web43], [Holmes02], [Web6], or even over every inch of a region to monitor microclimate [Web42].

Wireless technology has evolved through multiple generations. Below is a brief summary of the history of mobile technology:

- The First Generation (1G) technology are analogue cellular systems, a composite of a hybrid of analogue voice channels and digital control channels. Typically, the modulation for the analogue voice channel is FM whilst for the digital control channel is simple frequency shift keying (FSK). The first generation of wireless technology systems can send digital messages and provide advanced services such as short messaging. However, these messaging services are usually limited to very slow data rates. Adding new features to the service in this generation would generally require hardware changes to both the mobile telephones and cellular networks.
- The Second Generation (2G) technology are digital systems that use digital radio channels for both voice (digital voice) and digital control channels. 2G digital systems typically use more efficient modulation technologies, including global system for mobile communications (GSM).
- The enhanced 2nd Generation (2.5G) PCS/PCN technology are digital cellular systems that provide significantly new and improved capabilities over the 2G but not quite satisfy the third generation wireless requirements. This generation technology use improved digital radio technology to increase the data transmission rates and new packet-based technology to increase the system efficiency for data users.
- The third generation (3G) technology are called universal mobile telecommunications systems (UMTS). This generation technology provides high-speed (broadband) data services, supports simultaneous multimedia (regardless voice or data), is supposedly

cheaper and is claimed to be backwards compatible with 2nd generation systems.

3.7.5 WiFi (802.11) and Bluetooth

One widely deployed technology is a wireless extension of the Ethernet LAN known as the IEEE 802.11 (WiFi) standard . The 802.11b standard is sponsored by the Wireless Ethernet Compatibility Association (WECA) for the wireless networking of home devices.

Bluetooth is another type of wireless technology that enables short-distance range wireless communication between voice and data anywhere in the world [Miller02]. Bluetooth devices can send and receive information through a single *air-interface* within a certain range (typically 10 meters). Because the Bluetooth technology is designed for short distance it consumes less power The Bluetooth technologies used in the project were:

- Compaq iPaq 3970 - PDA
- Ericsson t68i – mobile WAP Phone
- PC2PC - Bluetooth adapter
- Picoblue - the iDorm Bluetooth bridge

3.8 Communication Protocols

Protocol is a form of middleware. It is a set of rules that computers (or other network devices) in networks use when they communicate with each other. It can also be regarded as the communication language used between the network computers/devices.

Most Communication protocols have an abstract model called a protocol stack (eg OSI Reference model), which refers to the layers it defines. Each defined layer has its own set of functions and communications. The concept of the protocol is that each layer in the protocol stack is only concerned about the information that is addressed to it. Moreover, they are only allowed to “talk” to the layer immediately above or below them. Information is passed down/up the protocol stack during the communication. When the information is passed down the stack, each layer in the stack will add a small amount of redundant data to the passing information for later identification before passes it down to the layer below. On the other hand, based on the identification data, every layer in the protocol stack will know if the passing information is addressed to it, if so, the layer will process the information, otherwise it will send the information to the next level. Because the layers in the stack need not know about the details of what is going on in all other layers (apart from their immediate neighbours), this concept allows the software and hardware to be designed independently, thus making the software components reusable, transportable and device independent.

3.8.1 TCP/IP

TCP/IP is the standard basic communication protocol use in the Internet as well as in a private network (either an intranet or an extranet).

TCP/IP design and concept is based on the OSI Reference, that each layer adds information onto the previous layers without modifying the contents of it. The TCP, stands for Transmission Control Protocol, is responsible for assembling/reassembling of message or file into smaller packets that are later on transmitted/received over the Internet; while the IP, Internet Protocol, handles the address part of each packet so that it gets to the right destination.

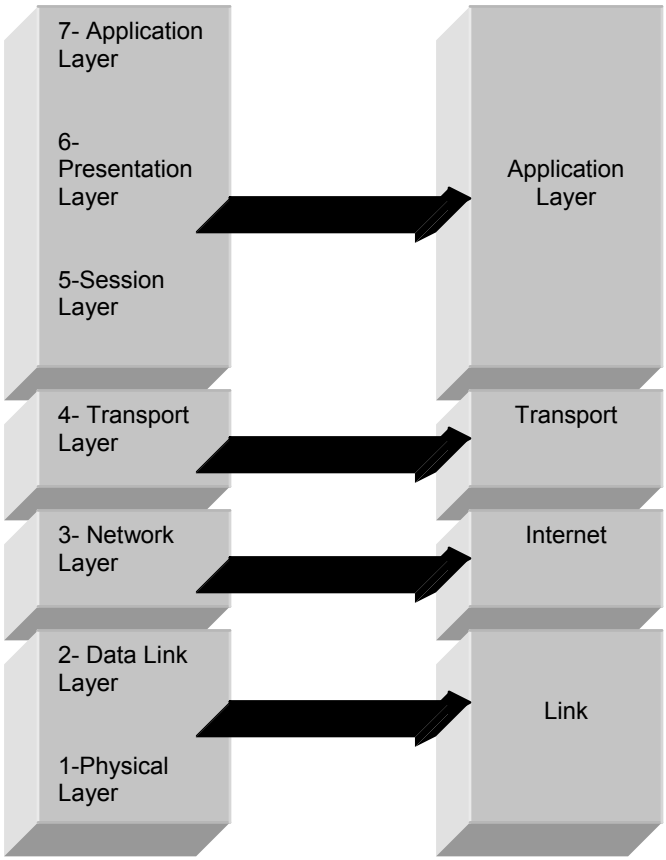

Figure 11. TCP/IP protocols.

3.8.2 HTTP

Hypertext Transfer Protocol (HTTP) defines the set of rules for exchanging multimedia files (text, graphic images, sound, video) on the World Wide Web. It is the most common protocol used in the Internet. For example, Web servers deliver files to their clients (web browsers) using HTTP protocol. The project used HTTP for Internet communications.

3.8.3 FTP

File Transfer Protocol (FTP) is a simple protocol used for exchanging files between computers on a network or over the Internet. The project used FTP for transferring program files from the WinMe machine to TINI system.

3.8.4 Telnet

Telnet is a way to access and control another computer on a network. It is a user application with an underlying TCP/IP protocol used for accessing remote computers. The project used telnet session to control the system.

3.9 Concluding Remarks

This project involves a diverse set of technologies ranging from sensors/effectors, embedded processors, "wearable computing" Java programming. Communications to user interfaces. It typifies the kind of technology that is needed to create pervasive computing environment and involves knowledge of less commonly encountered issues such programming embedded-Internet devices, emulation of mobile phones and dealing with multiple communication technologies. This diversity of technology adds to the project's difficulty but also makes it more interesting and educational (for me).

4. The TESA Requirement

4.1 Methodology Used

Software engineering is an engineering discipline which is primarily concerned with all aspects of software production, from the early stage of system specification to the final stage of system maintenance after the software goes into production.

There are several software engineering models, but the most fundamental one, namely *waterfall model*, with prototyping is deployed in this project. The model has eight stages as follow:

- Requirement analysis
- System design
- Program design
- Coding
- Unit testing and Integrating testing
- System & Performance testing
- Acceptance testing
- Maintenance

The diagram for this model is given below:

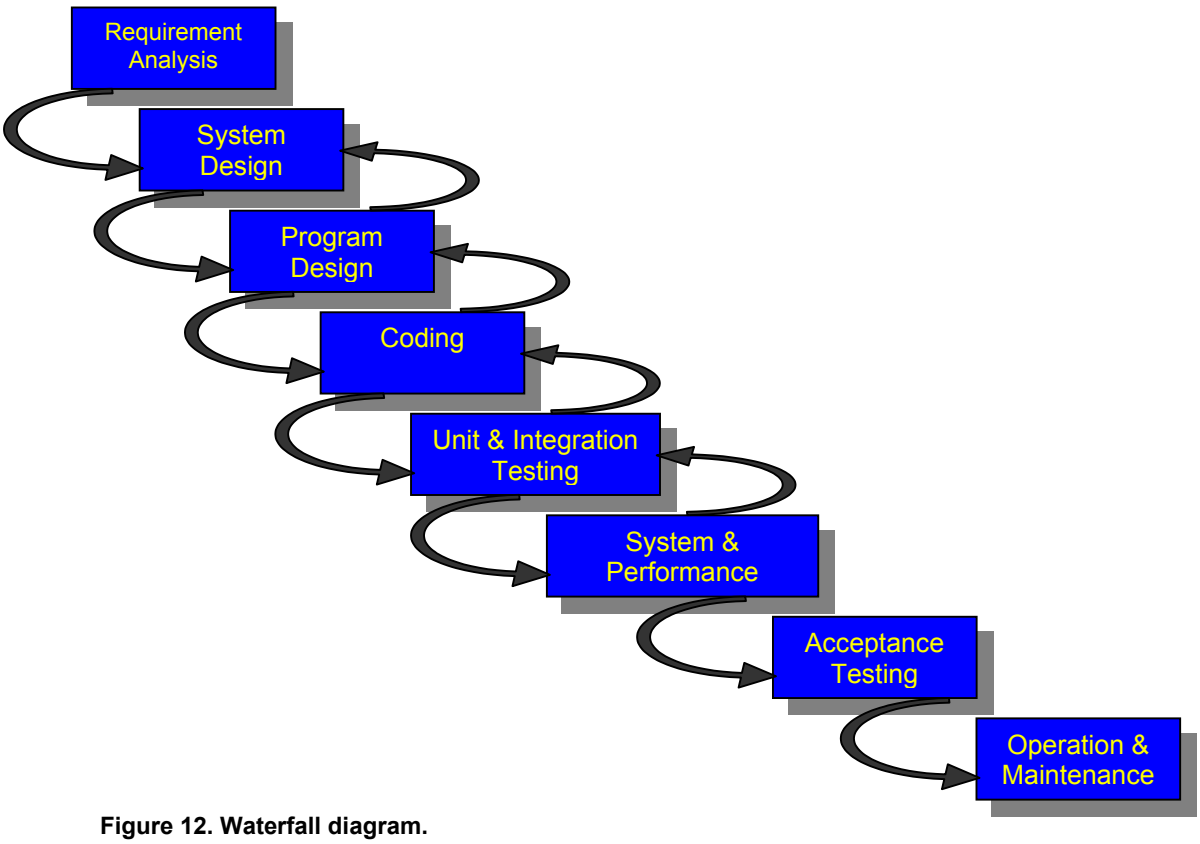

Figure 12. Waterfall diagram.

4.2 Requirement Analysis

Requirement analysis is the process of establishing *what* the system should do. It lists out the things that the system provides and the constraints under which it must operate. The requirement analysis has 3 main purposes:

- It allows the developer to explain their understanding of how they believe the client wish for the system to work.
- It provides information for designing.
- It provides information for testing.

Hence, the outcome of requirement analysis for the project had 2 documents:

- The requirement definition, written in non-technical terms for the client to understand. It contains everything the client expects the system to do.
- The requirement specification, written primarily for the system developers. It restates all that is listed in the requirement definition, but in technical terms.

4.3 The TESA Requirement Definition

The requirements were generated at the beginning of the project and were based on extensive consideration of the system's needs. They set the design critera for the project and are reproduced in this chapter.

System requirements:

1 To build a Web interface that corresponds with TESA hardware devices.

2 To build a PDA interface that corresponds with TESA hardware devices.

3 To build a WAP interface that corresponds with TESA hardware devices.

4 The TESA should be able to access and control via three different type of input interfaces regardless of how the connection was established initially.

5 The system should be able to monitor its environment temperature.

6 The system should be able to display its current environment status, either automatically for the Web and PDA client, or manually for the WAP client.

7 The system should be "friendly".

8 The system should be able to cope with more 1 concurrent connection.

Users requirements:

1 The user should be able to communicate with the application via the Internet (a desktop PC), wireless LAN/Bluetooth (PDA) or GPRS -WAP gateway (WAP enabled mobile phone)

2 The user should be able to monitor and control the TESA devices via the above interfaces.

3 The user should be able to select his desired application services via these interfaces.

4 The user should be able to select his preference settings when he chooses a service.

5 The user should be able to cancel his request for service at any time.

4.4 The TESA Requirement Specification

System requirements:

1 To build a Web interface corresponds with TESA hardware devices using standard WWW markup language HTML enhanced with JavaScript for easy navigation. The design should be simple yet pleasing. Ideal for one page implementation.

2 To build a PDA interface corresponds with TESA hardware devices using standard WWW markup language HTML, with a fixed 240*240 pixels interface featuring all services. Preferably to have the same theme as the Web interface, but pages can be extended if needed.

3 To build a WAP interface corresponds with TESA hardware devices using Wireless Markup language WML. Easy navigation is an essential and prefers less user input.

4 The TESA should be able to access and control via three different type of input interfaces regardless of how the connection was established initially. The system must be able to identify the nature of the connection and responds to the request appropriately.

5 The system should be able to monitor its environment temperature. Based on an input value, the system should be able to test it current temperature with the input value, and make judgement upon which actions to be taken if the comparison found to be deferred.

6 The system should be able to display its current environment status, either automatically for the Web and PDA client, and manually for the WAP client. The system should be able to read all its hardware devices current value and interpret in an appropriate format and display them accordingly.

7 The system should be “friendly”. Every request should be responded with a friendly acknowledgement.

8 The system should be able to cope with more than 1 concurrent connection. The system must maintain its consistency if different connections are established concurrently.

Users requirements:

1 The user should be able to communicate with the application via the Internet (a dektop PC), wireless LAN/Bluetooth (PDA) or GPRS -WAP gateway (WAP enabled mobile phone). The system should be able ready at all times for receiving requests and identify the nature of connection and responds appropriately.

2 The user should be able to monitor and control the TESA devices via the above interfaces. The interfaces should label all TESA hardware devices clearly. The system should respond with appropriate actions preferably no more than 3 seconds delay in normal network traffic once the request is received. The system should also check and validate the user input and generate appropriate message to the user accordingly.

3 The user should be able to select his desired application services via these interfaces. A service menu should be provided for each interface listing all the available services clearly.

4 The user should be able to select his preference settings when he chooses a service. The service for controlling individual devices should also accompany with 3 different settings.

5 The user should be able to cancel his request for service at any time. System should terminate its action once a cancellation is received.

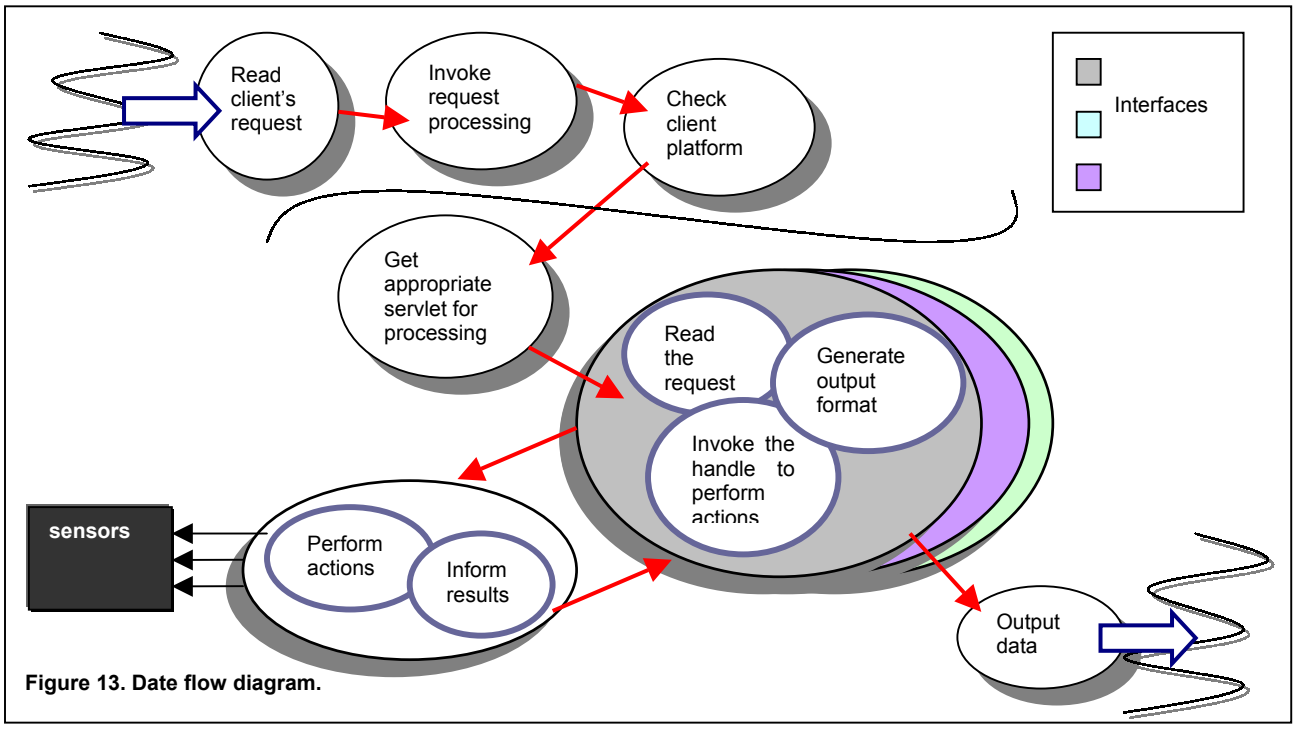

Figure 13. Date flow diagram.

The requirements form a key element in any system design being commonly the link between the customer and the designer. In this case I played a dual role as, in part, I was the customer whilst being, of course, wholly the designer. This was because the idea for a botanical care appliance was based on my interest in plants and Internet technology. Thus, it would be easy to imagine that the requirements could have become somewhat “flexible” but as this report will later show, these initial requirements form the bedrock of the project and accurately mirrored the final project outcomes.

5. TESA System Design

5.1 System Design Overview

The design strategy was to employ a modularised design strategy. The physical structure of application such as the use of a separate PDA and mobile phone interface dictated part of the modularisation methodology. The remainder of the modular decomposition was driven based on functional criteria such as separating the http server and the control systems. Following this philosophy led to a decomposition based around the *five* software components and a group of hardware devices shown in Figure 14. Apart from the existing hardware devices and low-level hardware driver component (which were a pre-written part of the development environment), the remaining 5 components were designed by me and are as follow:

- **Servlet class**

1. *Service* class: this is responsible for responding to a connection. It determines the nature of the requesting platform by reading the client's request header. It then invokes an appropriate Servlet and passes on the client request accordingly.

2. *PC* class: this is called from the *Service* class. Its job is to process the requests for the user by using a system class object. The PC class also gets the application's current environment status from the system class object and with the help of a utility class object; it generates HTML pages dynamically back to the client.

3. *Pda* class: this is called from the *Service* class. Its job is to process the Pda's requests using a system class object. This class gets the application's current environment status from the system class object and with the help of a utility class object; it generates a PDA version of HTML pages dynamically passed back to the client.
4. *Wap* class: this is called from the *Service* class and is primarily responsible for handling WAP client requests. The *Wap* class also uses a system class object and a utility class object to generate results in WML pages dynamically back in the client.

- **Handler class** (utility helper class), implements *Formatte* interface

1. *Formatte* interface: this is the abstract interface class. It defines 6 interface methods that its subclass should implement.
2. *PcHandler* class: A subclass of *Formatte* class. It is primarily responsible for generating the "script-enhanced" Web based version of standard HTML codes. It implements 6 methods that define in its superclass. Note: the HTML code is embedded with TESA IP address.
3. *PdaHandler* class: A subclass of *Formatte* class. It is solely responsible for generating the PDA version of HTML codes. It implements 6 methods that define in its superclass. Note: the HTML code is embedded with TESA IP address.
4. *WapHandler* class: A utility class for generating the WML codes that consists of decks of cards which WAP enabled devices can navigate. Note: the WML code is embedded with TESA IP address.

- **Tesa class**

The *Tesa* system class is a subclass of pre-written driver class for the mDorm development environment. It inherits the driver class attributes and methods including the private members but encapsulates them from the external environment (data hiding). The *Tesa* class has a private static attribute. It is used to store the reference of the user's desired temperature value. The system environmental temperature is monitored based on this value. *Tesa* class also has its own methods for responding to the Servlet requests.

- **SetTesaStates class**

This is a helper class that implements the *Runnable* interface. The class instance is created and instantiated by the Servlets class whenever a request that involves any system hardware is received. The primarily job for this class is to perform a user request for updating the system hardware devices' status.

- **Monitor class**

  This class monitors the environment temperature class based on user input. It also implements the *Runnable* interface. The class instance is created and instantiated by the Servlets class whenever such a request is received. The monitor class thread is used to monitor the application environment temperature in the application background continually, until some event occurs.

5.2 Software Architecture

The TESA system architecture is illustrated in the following diagram

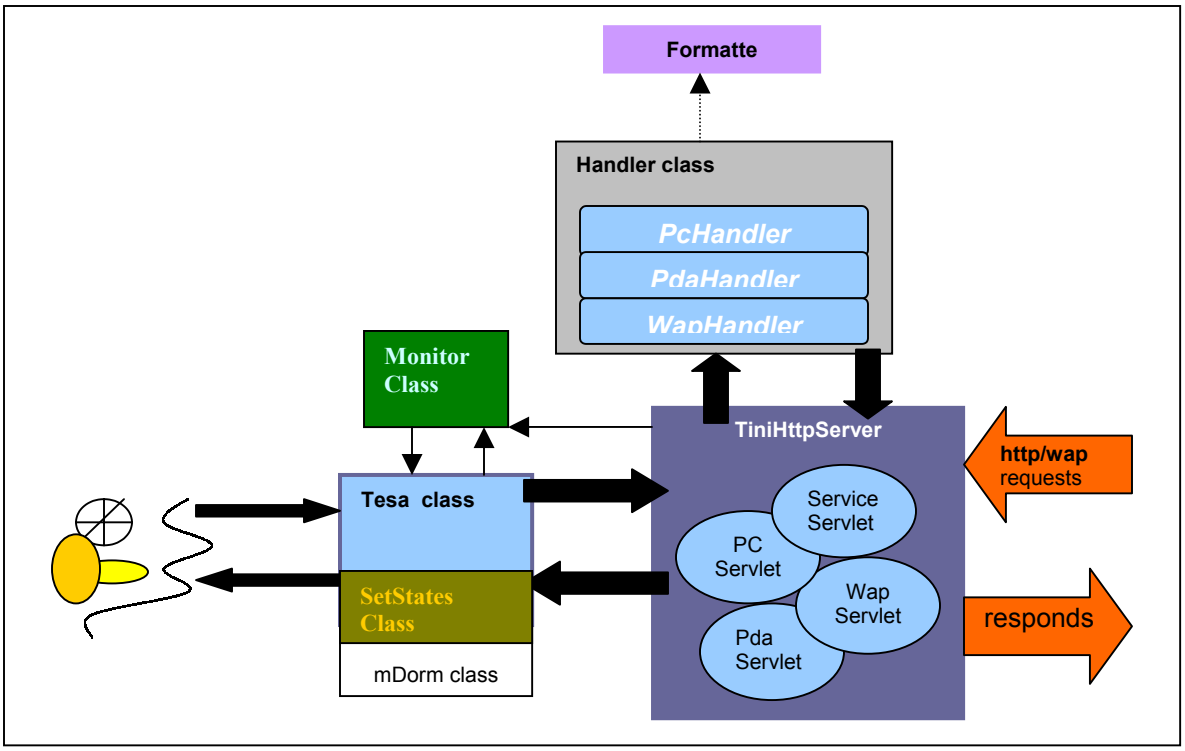

Figure 14. The TESA System Architecture.

5.3 Interfaces

In Oxford Advanced Learner's Dictionary, interface is described as "*a point where two subjects, systems, processes etc meet and affect each other or the way a computer program accepts/presents information from/to users*". This project had 3 types of interfaces.

5.3.1 The Web Interface

To comply with the system requirement presented in chapter 4, the TESA Web interface was designed to mirror a gang of traditional physical "push buttons" in which only one is active at any moment. Because the Web interface was to be viewed and accessed over the Internet, the control buttons were grouped together to form a type of menu. The control menu utilised a "drop down" effect for displaying its sub-categories. The effect was triggered whenever a mouse cursor glided over the menu. The "drop down" menu function was supported by Web "JavaScript" client-side programming while its format, was colour coded in a style sheet which the client's Web browser interpreted.

Below is a code fragment for the function related to displaying the menu. This function would be interpreted and called by a Web browser, ie the client:

```
    // show the menu
  function ShowMenu(obj)
  {
    HideMenu(menuBar)
    var menu = eval(obj.menu)
    var bar = eval(obj.id)
    bar.className="barOver"
    menu.style.visibility = "visible"
    menu.style.pixelTop = obj.getBoundingClientRect().top +
obj.offsetHeight + Bdy.scrollTop
    menu.style.pixelLeft = obj.getBoundingClientRect().left +
Bdy.scrollLeft
  }
```

The format of the display, such as the alignment, colour, font size etc for the menu was coded in a file ended with .cs, and this file is called style sheet. Below is the fragment code for the format of displaying the menu bar, horizontal bar, and the menu items that was placed on the menu bar:

```
.menuBar
{
```

```
    POSITION: relative;
    BACKGROUND-COLOR: transparent;
    TEXT-ALIGN: center
  }
 .Bar
 {
    BORDER-RIGHT: gray 1px outset;
    BORDER-TOP: gray 1px outset;
    FLOAT: left;
    BORDER-LEFT: gray 1px outset;
    WIDTH: 25px;
    CURSOR: hand;
    TEXT-INDENT: 5px;
    BORDER-BOTTOM: gray 1px outset;
    POSITION: relative;
    BACKGROUND-COLOR: silver;
    TEXT-ALIGN: center
 }
 .menu
 {
    BORDER-RIGHT: buttonhighlight thin outset;
    BORDER-TOP: buttonhighlight thin outset;
    VISIBILITY: hidden;
    BORDER-LEFT: buttonhighlight thin outset;
    WIDTH: 25px;
    LINE-HEIGHT: 100%;
    BORDER-BOTTOM: buttonhighlight thin outset;
    POSITION: absolute;
    BACKGROUND-COLOR: darkgray;
 }
```

A dedicated bonsai plant image was edited with care and placed on top of a 1024*1024 pixel white pre-prepared background layer using graphics tool Photoshop6. This image was then positioned in the middle section of the Web interface.

Figure 15. TESA's interface image

For the purpose of reusability, the Web interface was set in a pre-defined table smaller than normal size window, but one that could be dynamically adjusted by the Web browser. The intention was that it could be re-used later for the PDA interface, since the iPaq used in the project also supports the HTML. To comply with the system requirement stated in chapter 4, TESA current environmental status was displayed at the bottom section of the interface, in a tuple format. The colour orange, white and grey with black coloured background were chosen to be the interface theme colour. Below are the snapshots of TESA Web interface.

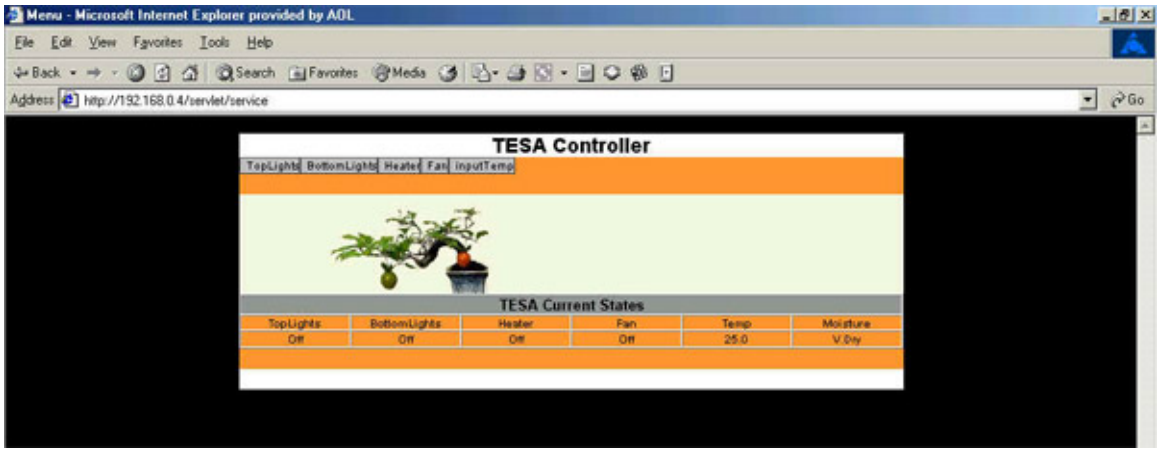

Figure 16. TESA's Web interface (1)

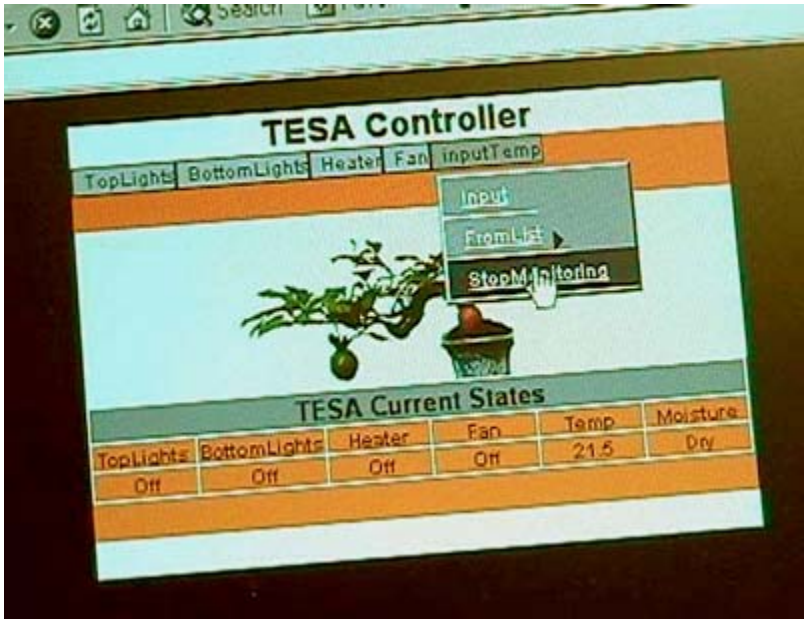

Figure 17. TESA 's Web interface (2)

5.3.2 The PDA Interface

As mentioned in chapter 3.4, a Compaq PDA (iPaq model 3970), was used in the project. This uses the Microsoft Windows operating system (WinCE) which supports the Web standard markup language HTML. Therefore the PDA interface design was initially intended to have the same design theme as the Web interface. This was an attempt to unify the application interfaces whereby the user would only once need to learn to use the system "controller". However, it was later found out that whilst the WinCE browser supports a scripting language it didn't support a style sheet which the Web interface design drop-down menu displays solely depended upon. As the time allowed for completion the project was tight and to avoid too many processing overheads for the handheld device (it has low processing and memory power), a decision was made use a slightly different approach. This amounted to exposing the PDA.s interface control buttons on its top section but retaining the theme structure to maintain the unified look. Colouring was used to distinguish the control buttons from each other. A snapshot of PDA interface is shown below.

Figure 18. TESA's PDA interface (1)

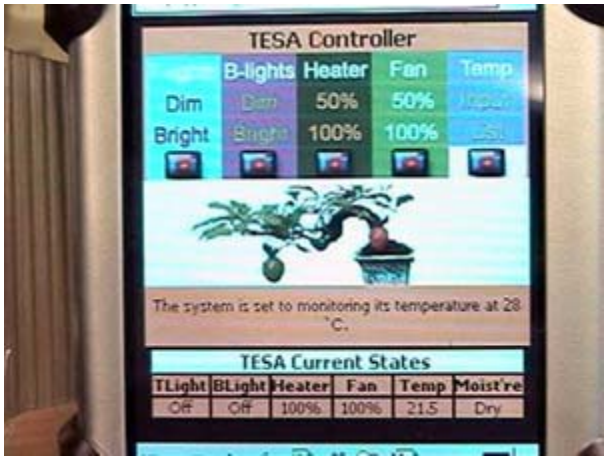

Figure 19. TESA's PDA interface (2)

5.3.3 The WAP Interface

TESA's WAP interface was written in WML, which had a completely different look compared to the other 2 interfaces. The WAP interface was solely text-based, no images, and user could need to navigate to/from the interface for selecting/sending requests. The main reason for the WAP interface being a text-based design, and different from the others, was that WML supports only very limited image files. The image file used in other interfaces was not supported here. The other reasons this interface was different were:

- The WAP phone had a very tiny display screen where the Web/Pda design theme was impossible to fit in.
- The WAP phone had a very limited memory space.
- The WAP phone had a low processing power

TESA's WAP interface was made up of "desks of cards", which a user could navigate. The declaration, shown below, was needed to be placed in the very first line of every WML program. The purpose of this declaration was to validate WML code. The WML version used in the project was wml1.1

<?xml version=\"1.0\"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" http://www.wapforum.org/DTD/wml_1.1.xml>

Instead of having “button-like” controls as in the other interfaces, the WAP interface consisted of a text menu listing the services that TESA provided. The user was then required to navigate through the menu to select his desired service.

The code shown below is for the menu deck. This menu deck consists of 3 cards. When the user selected a service, he was required to navigate to another deck that provided the service. If the user chose to cancel his request, or wished to terminate the connection (a user requirement stated in chapter 4.3), he would be navigated from his present card (1) to the card below but in the same deck, card (2), and from there he would be prompted for confirmation. If the user selected to stay with the original service, he would be navigated back to the card (1) which listed the menu, otherwise he would be brought to card (3), and the termination would be completed there. Note: Each card is begin with <card id=”xx”> and end with </card>.

```
<card id="card1">
<do type="options" label="Cancel">
<go href="#card2"/>
</do>
<p>
<big><b>TESA Controller</b></big>
<br/>
<b>Select services</b>
<select name="type" ivalue="0">
<option onpick="http://borg8of9.essex.ac.uk/servlet
/wap?type=Current">Current State</option>
<option onpick="http:// borg8of9.essex.ac.uk/servlet
/wap?type=DeviceItems">System Devices</option>
<option onpick=\"http:// borg8of9.essex.ac.uk/servlet
/wap?type=Temperature">Read Temperature</option>
<option onpick="http:// borg8of9.essex.ac.uk/servlet
/wap?type=SetTemperature">Set Temperature</option>
<option onpick="http:// borg8of9.essex.ac.uk/servlet
/wap?type=Stop">Stop Monitoring</option>
</select>
</p>
</card>
<card id="card2">
<do type=\"accept\" label=\"Yes\">
<go href=\"#card3\"/>"
</do>
<do type=\"options\" label=\"No\">
<go href=\"#card1\"/>
</do>
```

```
<p align=\"center\">
<big><b>Are you sure you want to quit?</b></big>
</p>
</card>
<card id=\"card3\">
<p align=\"center\">
<big><b>Thank you for using TESA Controller</b></big>
</p>
</card>
```

Below are the snapshot diagrams for WAP interface.

Figure 20. TESA's WAP interface (1)

Figure 21. TESA's WAP interface (2)

5.4 Comments

The above description is a minor part of the overall software implementation which is more fully described in the appendix and the attached CD. It was felt describing the software in its entirety in this chapter would be a somewhat tedious experience for the readers. Instead only a few issues have been extracted for illustrative purposes in this chapter, such as the general software architecture and some issues relating to WAP programming. It was felt an overview of the software architecture was essential and hence this was included above. As the WAP programming was outside the curriculum, and took some considerable self-education, it was also felt that it would be useful to include an insight into this process here. Overall, all the requirements described in chapter 4 have been implanted and the code can be found in the attached CD.

6. Software Implementation

TESA has 5 main software components, namely:

- Servlet class
- Handler class
- Tesa class
- SetTesaState class
- Monitor class

6.1 Servlet Component

In the communication process between a user and a Web application, sometime referred to as client-server communication, it is always the user (client) that initiates the connection and sends a request to the Web application (server). This request is encapsulated in the Uniform Resource Locator (URL) that contains the address of the requested server and sent by the Web browser GET command. The URL looks something like: http://borg8of9.essex.ac.uk/servlet/service?type=Current , where the data after the "?" will be the user's request.

While sending the URL, the browser is also sends a *header*, which contains an array of data format information. This information indicates to the server what kind of the data format that the client browser is willing to accept. The server will read the command, get the information, try to execute it and send the result back (or, if there is an error, it will send back an appropriate error message in an appropriate format). The interaction between client and server is shown in the following diagram.

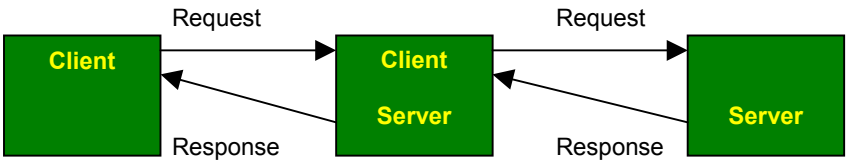

Figure 22. Conceptual Client/Server Model.

A servlet is a software process that sits on the server side waiting for a client request; responding to a request, process it and returning a result.

TESA used the TiniHttpServer engine, which ran as the application Servlet engine on its TINI. However, the Tomcat Servlet engine, which ran on the WinME machine, was used for testing and debugging. The servlets were written in Java and were responsible to respond to a request whenever a connection was established. There were four Servlets written for TESA, namely *Service*, *PC*, *PDA* and *WAP.*

6.1.1 Service class

Service Servlet was the project main Servlet that sat on the TiniHttpServer. It would be called upon receiving a client connection irrespective of which connection platform the client used. The *Service* Servlet would respond to the client request by reading its "*header*" information, which would be sent along with the client request, to determine which connection platform the client was from. The "*header*" is a small amount of information which the client uses to indicate to the server the type of document format that it is willing to accept. For example the WAP client will indicate to the server that only the document type, ending with the .wml will be accepted whilst the PDA client will have a header that describes the document UA-pixels: {i.e. 240x320} etc. It is the client's responsibility to inform the server about this information prior to receiving any response, so that to ensure only the correct type of format is sent/received.

Once the *Service* Servlet receives this information, it creates an object instance of the appropriate Servlet (based on the connection platform) and passes on the request. From then on, as far as the *Service* Servlet is concerned, this connection no longer has responsibility until the next new connection established. The *Service* Servlet "read and pass" enabled TESA to be accessed, controlled and monitored regardless of how the initial connection was established. Below is the small fragment of the code:

```
String accept = req.getHeader("accept");
::::::::::::
::::::::::::
//see what the browser is willing to accept
if (accept==null || accept.indexOf("wap.wml")==-1)
{
        if (accept==null || accept.indexOf("Window CE")==-1)
        {
        //this is normal html's requests
        //pass the request to the PC servlet
        PC pc = new PC();
        pc.doGet(req, res);
        }
```

```
        //Pda requests start here
        Pda pda = new Pda();
        Pda.doGet(req,res);
 }
else
 {
        //Wap request starts here
         Wap wap = new Wap();
         wap.doGet(req, res);

 }
::::::::::::
::::::::::::
```

6.1.2 PC class

PC Servlet is called when the *Service* Servlet determined the request was from normal Web browser. Its job is to read the request, process it and return the result. The *PC* Servlet content type was set as "text/html" to ensure its output was in Web document format. The request sequences for the *PC* Servlet were:

- If the *PC* Servlet determined the request was for changing the system device status, it would create a system helper class object (*SetStates*) to handle the request and then use a handler object (*PcHandler*) to print out acknowledgement dynamically to the user.
- If *PC* Servlet determined the request was for monitoring the system environment, it would then (1) validated the input (2a) if the input was validated, it would test to see if there was any old thread running on the system monitoring routine, if so flagged it to stop, and went to (3) or (2b) printed an error message and went back to (1), (3) create and initiate a thread and pass on the valid parameter for the new thread to start to monitoring the system environment.
- If the *PC* Servlet determined the request was to stop the system monitoring it environmental temperature, as before, it would (1) test to see if there was any old thread running on the system monitoring routine, if so flagged it to stop, and (2) it would create a system helper class object to switch off the currently working system devices.
- If an error occurred, the *PC* Servlet would handle the exception by generating appropriate messages.

Below is the code fragment:

```
 //create a PC format utility class object
 PcHandler handler = new PcHandler();
//create a system class object
 Tesa wborg = new Tesa();
//declare a monitor thread object
Monitor mp = new Monitor(wborg);
::::::::::::::::::::
//get the request value
String temp = req.getParameter("temp");
String temp = req.getParameter("temp");
::::::::::::::::::::

if (temp.equals("list"))
{
        …
}
else
{
        try
        {
                //get the value from the parameter, might throw an
        exception here if the input is not a valid number
                lvalue = Integer.parseInt(temp);
                String st = "The system is set to monitoring its
        temperature at "+lvalue+" `C. ";
                //check if the input is within the pre-define range
                if (isValid(lvalue))
                {
                                //kill any running thread
                                  stopRun();
                                //initiates a new thread and pass on the
                                //paremeter
                                  getMonitor(lvalue);
                                //print acknowledgement and back to
                                //initial state
                                  printMenu(pw,st);
                }
                else
                {
                        //if the input value is out of the pre-define
                        range, //print error message
                        writeErrorMessage(pw, er2);
                }
        }
```

```
                catch (NumberFormatException nfe)
                {
                        System.out.println("input error!");
                        //print error and ask to input again go back to print
                        //menu page
                         writeErrorMessage(pw, er1);
                }
        }
```

The *PC* Servlet had 15 private supporting methods, mainly for its printing jobs. For example one of the method retrieved the application's current environment status through system class object and used another method to convert these value into a meaningful form. These functions also used a handler class object (*PcHandler*) to print out the results in HTML pages. Below is the fragment of the code:

```
private void getCurrentState(PrintWriter pw)
 {
        int itop = wborg.getTopLightsState();
        int ibottom = wborg.getBottomLightsState();
        ::::::::::::::::
        String top = switchState(itop);
        String bottom = switchState(ibottom);
        ::::::::::::::::
        handler.writeCurrentStateTable(pw);
        handler.writeCurrentState(pw, top, bottom, heat, fan, temp, moist);
}

private String switchState(int n)
{
        String result = "UnKnown";
         switch (n)
        {
                case 0:
                 result = "Off";
                 break;
                 case 30:
                 result = "Dim";
                ….
         }
         return result;
}
```

Please refer to the Appendix H for the rest of the private methods.

6.1.3 Pda class

The *Pda* Servlet was called when the *Service* Servlet determined the request was from a pocket PC. Its job was very similar to the *PC* Servlet in that it read the request, processed it and returned the result. The only difference between the *Pda* Servlet and the *PC* Servlet was that the *Pda* Servlet used a different handler class to print out the result, otherwise, the rest of the routine functions were the same. Below is the *Pda* Servlet declaration:

```
//declare a Pda format utility class object
PdaHandler handler = new PdaHandler();
//declare a system class object
Tesa wborg = new Tesa();
//declare a monitor thread object
Monitor mp = new Monitor(wborg);
```

Please refer to the Appendix H for *Pda* Servlet private methods.

6.1.4 Wap class

The *Wap* Servlet was called when the *Service* Servlet determined the request was from a WAP enabled device. Its task sequences were the same as with the other servlets in that it read in the request, processed it and returned the result. The *Wap* servlet used the WAP handler class- *WapHandler* (which generates WML pages) to dynamically print out responses and results for WAP clients. Below is the *Wap* Servlet declaration:

```
public class Wap extends HttpServlet
{
        //declare a Wap format utility class object
        WapHandler handler = new WapHandler();
        //declare a system class object
        Tesa wborg = new Tesa();
        //declare a monitor thread object
        Monitor mp = new Monitor(wborg);
        ::::::::::::::
        ::::::::::::::
```

Because the language used in WAP enabled devices is very different from the Web applications, the *Wap* Servlet content type was set to "*text/vnd.wap.wml*" instead of the normal Web type "text/html". The *Wap*

Servlet made extensive uses of its handle class for printing WML pages, thus it had fewer helper methods than the other 2 servlets. Please refer to Appendix H for *Wap* Servlet private method.

6.2 Utility Component-the Handler class

The Handler class is a utility class which is only responsible for printing the appropriate format pages. TESA had 3 Handler classes; two, the *PC* and the *Pda*, were implemented in the same interface *Formatte*, which defined the methods that should be used when printing the HTML format.

6.2.1 Formatte class

The *Formatte* class was the interface class. It defined 6 methods which needed to be implemented for all its subclasses in printing the HTML pages. The *Formatte* class had 2 subclasses- the *PcHandler* and *PdaHandler*. The 6 methods it defined were:

- public abstract void writeHeader(PrintWriter pw, String s);
- public abstract void writeFooter(PrintWriter pw);
- public abstract void writeMenuTable(PrintWriter pw);
- public abstract void writeReplyRow(PrintWriter pw, String s);
- public abstract void writeCurrentStateTable(PrintWriter pw);
- public abstract void writeCurrentState(PrintWriter pw, String a, String b, String c, String d, double e, String f);

6.2.2 PcHandler class

The *PcHandler* class was a utility class that was responsible for printing Web version of HTML pages. *PcHandler* class implemented *Formatte* interface. It implemented 6 *Formatte* class defined methods as well as its own methods. Note: the HTML code in the *PcHandler* class was embedded with TESA IP address. Refer Appendix H for the *PcHandler* class methods.

6.2.3 PdaHandler

The *PdaHandler* class was another utility class responsible for printing the Pda version of HTML pages. Like *PcHandler* class, *PdaHandler* implemented *Formatte* interface. Apart from the 6 defined methods, *PdaHandler* class also had its own printing methods. Note: the PDA version

of HTML code was embedded with TESA IP address. Refer Appendix H for *PdaHandler* class.

6.2.4 WapHandler

Unlike the other two handler classes mentioned above, *WapHandler* class did not implement *Formatte* interface, but it had its own methods, and was responsible for printing out WML pages. *WapHandler* class had 13 printing methods. Note: the WML code was embedded with TESA IP address. Refer to Appendix H for *WapHandler* class.

6.3 System Component-Tesa class

The *Tesa* class was the system class. It interacted directly with system hardware devices. *Tesa* was also a sub-class of driver class - mDorm, thus it inherited all the driver class's data members as well as its methods but encapsulated them. The *Tesa* class had only one private data member, to store the user preference environmental value.

Apart from the methods associated with *Tesa's* own private data member, ie. for setting/retrieving the value, Tesa class had only one method (with 2 argument parameters). The method was used to read the request (passed by the servlets as the arguments) and made appropriate actions. For example if the request was for changing the setting of one of the system's hardware devices, the method would check and find out the user desired setting, and once it got hold of that information, it would then (1) compare the device concerned current status with the user desired status (2) if both of the status differed, it would update the new status by sending a signal to the hardware device concerned, or (3) it would do nothing if both of the status were to remain the same. The code below is the fragment of code showing a "toplights" scenario:

```
.....
.....
 if (device.equals("TopLights"))
   {   //3 states
   try
   {
          int top = getTopLightsState();
          //testing variable for the toplights
           if (state.equals("Dim"))
             {
                  if (top==30)
                  {
                 //do nothing if the top lights are already in dim state
```

```
                    System.out.println("Error! PDorm's top lights are
                    already set in Dim state");
                     }
                     else
                     {
                    //reset
                      setTopLightsState(0);
                     try
                    { Thread.sleep(2000);
                    } catch ( InterruptedException e ) {}
                     setTopLightsState(30);
                     }
 }
```

6.4 System SetStates class

The *SetStates* class was a support class implemented *Runnable* interface. This class was to reduce the load for the main executing thread during the program execution. The job for this class was primarily to deal with requests for changing the system's device status. It was closely associated with the *Tesa* class.

The *SetStates* class has 4 private data members: *Tesa* object (for the reference of the object it ran), 2 primitive string type variables (for the requested values for the object) and a Thread object (for tracking the current running thread). The following fragment shows the run method of *SetStates* class:

```
class SetStates implements Runnable
{
   private Tesa tesa;
   private String device;
   private String state;
   private Thread runStates;
::::::::::
  public void run ()
  {
          //track the current executing thread
           runStates = Thread.currentThread();


           System.out.println("SetStates thread run ");
           tesa.setStates(device, state);
```

```
            try
                    {
                            Thread.sleep(2000);
                    }
                    catch ( InterruptedException ie )
                    {
                             Thread.currentThread().interrupt();
                            Return;
                    }
      }
}
```

6.5 System Monitor class

The *Monitor* class was responsible only for monitoring the system environmental temperature. It implemented the *Runnable* interface and had 4 private data members.

The *Monitor* class controls the system temperature by comparing the system current temperature status to the requested status. If the requested status stated the system needed to be warmed up (ie. the value was bigger), the class would set the heater and fan on to full (putting fan on to speed up the hot air circulation). Conversely, if the system needed cooling down, the class would set the fan on (in full speed) to increase the air circulation (there being no cooler on the mDorm, to this only worked in the ambient temperature was less than the set-point). If the system status was as requested, the class would stay unchanged, periodically checking until the next change (either from the system or a new request). The monitoring routine was set to monitor the temperature continuously until a termination event occurred.

Because other classes could create threads of this class through *Runnable* interface, and each newly created thread would be running continuously in the system background (monitoring the system temperature), there was potential jeopardise the system consistency. To overcome this potential problem, 2 crucial steps needed to be taken (1) the data members of this class had to be declared as "*static*". Using "*static*" data members would mean that the same piece of data would be seen across by all the running threads, regardless of which classes the threads were created from, (2) the method used for monitoring the system temperature must declared as "*synchronized*", meaning this method could only be run by one thread at a time, as threads must obtain a lock before running. These steps were critical to maintaining the system consistency

as the system requirement stated in chapter 4.3, that the system should be able to monitor its environmental temperature based on the user input, but the user could be from any of the 3 communication platforms. Furthermore, the requirement was that more than one user could make the same requests with different value at the same time! For example, a PDA connected user could request the system to monitor its environmental temperature based on a specific value, but while the system processed this request, a WAP connected user might send the same piece of request but with a different value. Since the *Monitor* thread could be created for each request, irrespective of the means of connection, the system now had 2 *Monitor* threads running at the same time, in the background, monitoring its environmental temperature, but each based on a different value! This problem was overcome by declaring the method as "*synchronized*". For example, if the first thread of the above scenario was running in this method, it must have obtained a lock beforehand, and it would hold the lock while it processing the task. Meanwhile the second thread came along and saw there was no lock available and it would wait for the first thread to finish its job before releasing the lock. Below is a code fragment for the method showing the "*synchronized*" declaration.

```
public synchronized void monitorP_DTemperature()
   {
     if ( pdorm.getTemperature() <  pdorm.getNew_temp())
    {
          pdorm.setHeaterState(100);
          try
          {
                  Thread.sleep(1000);
           }
           catch ( InterruptedException e )
          {
            // stop the running thread
            return;
          }
         pdorm.setFanState(100);
         ..........
```

6.6 Comments

The above description provides and software implementation which is more fully described in the appendix and the attached CD. It was felt describing the software in its entirety in this chapter would be a somewhat tedious experience for the readers. Instead only a few examples, in the form of code fragments, have been extracted for illustrative purposes in this chapter. Overall, all the requirements described in chapter 4 have been implemented and the code can be found in the attached CD.

7. Software Testing & System Evaluation

This chapter comprises two sections. The first section presents the software testing strategies and the test results (looking for coding or logical errors); the second presents a simple user evaluation exercise (assessing the users reaction to the ”look and feel” of the system).

7.1 Testing Methodology

The testing strategies used in the project are as follow:

7.1.1 Code Walk Through

- **Objective**: to find faults such as syntax errors, the consistency of variables names, program logic errors etc.
- **Strategy**: The program codes were at least read through twice as soon as they were coded. Errors found were noted for reference.
- **Methods used**: Hard copies of the code were made once it had been written. These copies were used to carefully examine the program logic and the sequence of program execution events. At the same time names of the variables were checked for consistency. All errors were noted when found.

7.1.2 White Box Testing

“White box” testing began soon after the process of “code walk through” finished. This testing was an ongoing operation as the code was written. Because the TINI system used in the project had well-known drawbacks [Web24] in terms of limited processing and memory capacity, it is standard practice for the TINI community for program debugging to be normally conducted on a development PC rather than in the TINI itself. This is to ensure the program is compiled error-free before deploying it to a TINI for subsequent testing. The practice is aimed at avoiding overloading the TINI by exhaustive debugging. Experience in this project indicated that the TINI would only manage to stay functional for an average of 10 “build and deploy” cycles. The most common problem encountered after 10 build cycles was “lack of memory space” necessitating the system to be rebooted and its memory heap cleared. The problem with rebooting was, except for the TINI firmware, all the

files/folders in the system would be lost, requiring all the application software to be reinstalled via ftp.

The project adopted the above as standard practice. The program debugging was performed in WinME machine before being built and deployed to the TINI for functional testing. Apart from the interfaces unit testing, the Tomcat Servlet engine was used extensively in this stage, as its functions were very similar to the TiniHttpServer.

7.1.2.1 Unit Testing

- **Objective**: to find the individual interface component output errors, interface errors, pre-mature termination etc.
- **Strategy**: to ensure every statement, decision point and distinct path in a component were executed at least once.
- **Methods**: interfaces were debugged and tested until they were deemed satisfactory. Tools such as Nokia sdk, PhotoShop6 and DreamWeaver were used in editing and correcting the errors for the interfaces.
- **Test cases**: each interface was thoroughly tested at least once.

a. Servlet class

- **Objective:** to find servlets response errors
- **Strategies:** to ensure the interaction sequences between the Servlets were correct, and also to verify the Servlets output was of the right order.
- **Methods:** The program code had been configured to adapt the Tomcat Servlet engine environment. Interface code was embedded in serlvets and placed in the appropriate directory within the Tomcat engine. Configurations were made to Tomcat so that it would serve the testing servlets automatically when it went into action. Web browser (Internet explorer) and 3 WAP emulators (Openwaves Nokia and M3Gate) were used as the clients in the testing.
- **Test cases:** for each interface connection, every service was requested at least once.

b. Handler class

- **Objective:** to find interfaces working improperly with the servlets.
- **Strategies and methods:**

1. **For Web Interface:** Tomcat engine was brought up and a Web browser was used to test each service and their settings at least once.
2. **For PDA Interface:** as above.
3. **For WAP Interface:** The Tomcat Engine was started and 3 WAP simulators (Openwaves, Nokia and M3Gate) were used to test the each service and their settings at least once.

c. Tesa class

- **Objective:** to find any system classes that interact abnormally with the hardware devices, and to find user requests and their settings that were improperly handled.
- **Strategies and methods:** Each hardware device (top lights, bottom lights, fan, and the heater) was activated and tested at least once.
- **Testing Functions Prepared:** as the testing was not performed on the TINI environment, a stud (Appendix G) had been written prior to this test.

d. Monitor class

- **Objective:** to find system consistency faults.
- **Strategies and methods:** 3 interfaces were used to send requests simultaneously to activate each hardware device (top lights, bottom lights, fan, and the heater) and input/select temperature value at least once.

7.1.3 Black Box Testing

The Integration testing for the project was conducted by largely adopting a “black-box” testing strategy. This test was performed on the Tini environment. The preparation work involved: (1) Building the program code for the TINI (ie. convert it into .tini file) using “ant” command from the WinMe machine and (2) Deploying the image files used in the project in the appropriate directory of the Tini via ftp. In addition, the system’s functional and consistency testing was performed during the Integration Testing.

- **Objective:** To find and eliminate any errors resulting from components not working together as a higher-level system in the Tini environment.
- **Strategy**: program code was edited to work with the Tini environment and networking before deploying it to a TINI. The machines configuration were checked and corrected where necessary. The version and errors were documented clearly for any necessary later testing using the WinMe machine. The Tini was rebooted if problems occurred.
- **Test cases**: In the TINI environment, the TiniHttpServer was brought up via telnet session. A Web browser, iPaq and WAP mobile phone were used to send/receive requests/responses simultaneously to (1) activate each hardware device (top lights, bottom lights, fan, and the heater) and (2) input/select temperature value at least once. Note: The Web browser used was from the WinME machine via the Internet whilst the PDA used Bluetooth connection and the WAP mobile phone used a GPRS connection.

Below diagrams illustrates the Integrating testing.

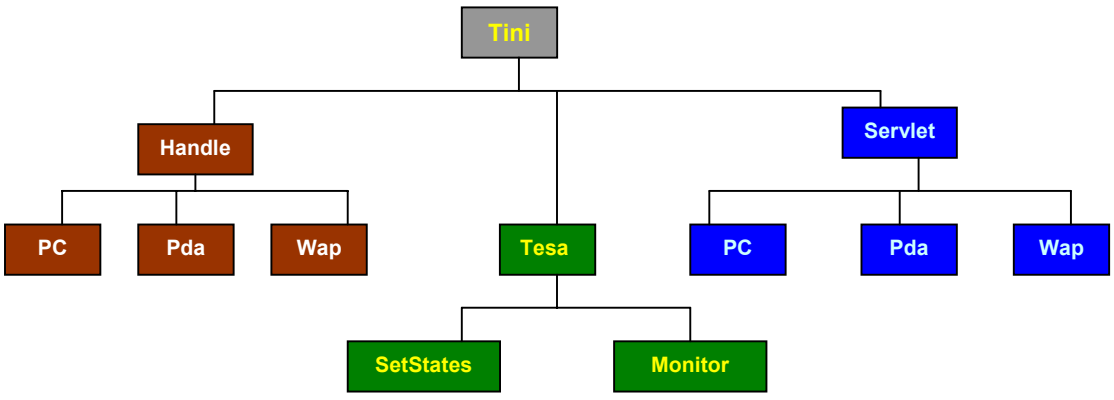

Figure 23. TESA software component hierarchy.

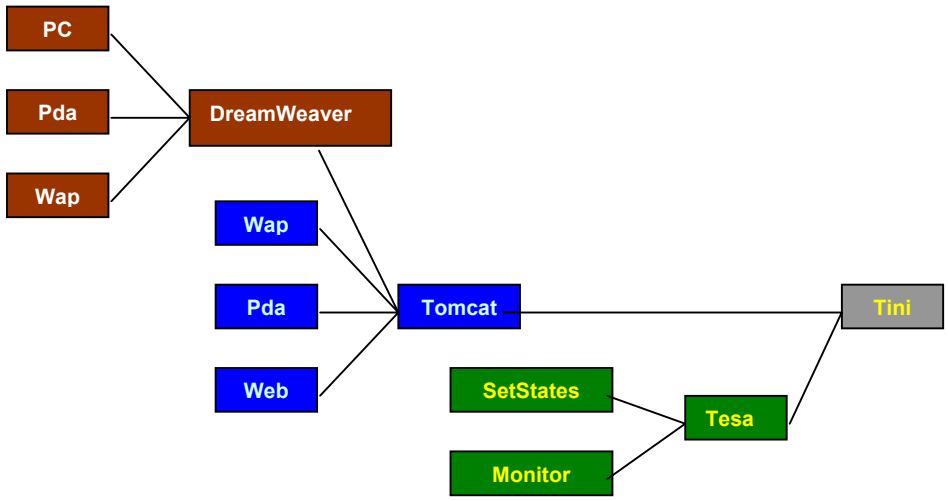

Figure 24. Integration Testing

7.1.4 Performance Testing

The Performance Testing was conducted in the IIEG lab on the 9th March 2003. The equipment used were:

- Borg8of9
- Egadgets4
- WinMe machine (laptop)
- IPaq 3970
- SonyEricsson t68i
- Pico- iblue1 (Bluetooth bridge)
- Baby-G time watch

The performance testing was calculated based on the time taken by different client interfaces accessing the system and requesting a set of pre-defined services (Appendix D) via different means of connection. There were 3 tests in total for the performance testing. Note: Each interface was performing its request of the pre-defined services in the same sequence order.

During the tests, borg8of9 was served as the system server. The TiniHttpServer on the borg8of9 was brought up by a telnet session using egadgets4 machine. When the TiniHttpServer was up and running, the first performance testing began by a Web browser from the WinMe machine calling the system Web interface over the Internet. The Timer was set to time the process as soon as the connection had established

(indicated on the telnet session panel). The testing was then ended by stopping the timer when the last requested service was completed; again, the completion of requested service was indicated on the telnet session panel.

The second performance test was performed by the iPaq 3970 calling the PDA interface using WinCE browser via Pico-iblue1 radio wave Bluetooth connection. The same testing sequence as before was performed when the connection established.

The third performance test was carried out by the SonyEricsson t68i calling the WAP interface via GPRS connections through Vodafone WAP gateway. The same testing sequence as before was performed when the connection established.

Figure 25 and 26 shows the Performance Testing environment.

Figure 25. Performance Testing Environment.

Figure 26. Performance Testing in actions.

7.1.5 Summary of results

Below table summarises the test results:

method	Logical errors	Sequential errors	Syntax errors	Naming errors	Misplaced errors (display)	No. of corrections
Code walk through	10	52	2	0	0	64
Unit testing (Web)	0	0	0	0	3 + 5 for the image files	8
Pda	0	0	0	0	8 + 3 for the image files	11
Wap	0	0	10	0	3	13
Servlets (all)	4	7	0	0	0	11
Integration testing	0	8	1	2	3	14
Total errors:	14	67	13	2	25	121

Figure 27. Summary of test results

There was an overall 121 errors and corrections for the project. The programs logical and sequential errors were mostly found during the early stages of the development process by careful “code walk through” examinations. However, the interfaces display errors were found in later stage (unit testing) as that was the only time those errors would be known.

The WAP syntax errors were notably hard to spot in the early “code walk through” stage. That was because WAP programming would need “live” validation from the WML forum site for finding errors. Servlets errors, in particular their sequential errors, were only easy to find and correct while testing them. This explains why they were mostly found in the later unit testing stage. Integration testing was conducted in the Tini environment and was the phase that generated the second highest error detection. The system had to be rebooted 9 times with memory heap cleared 7 times.

Although the integration testing recorded the second highest errors, those errors were mainly the sequential errors that occurred during the program execution. The system devices, their settings and the system temperature environment were tested successfully in this process. They could be controlled and monitored over the Internet using a normal Web browser as well as remotely via wireless communication using the short-range Bluetooth wireless connection in the iPaq. Moreover, the GPRS connection via WAP gateways using SonyEricsson t68i was tested successfully tested in this phase.

The system’s automatic environmental temperature control system, based on user input, was also tested and found to be satisfactory. In more details, the test output (see Appendix E) showed that the system was capable of maintaining its consistency after being deliberately forced to respond to 3 different platform connections requesting with 3 different requests for monitoring its environmental temperature. In addition, the system demonstrated it could automatically keep the temperature at a user set point despite external disturbances to the environment temperature. The test involved sending a request (via a browser) for the system to increase its temperature (ie sent value =28, the system current value=24). The unit responded by putting its heater and fan on (the purpose for the fan there was to circulate the hot air more quickly to the environment). Just as the system temperature was rising (ie 26.5), another request from a WAP phone (SonyEricsson t68i) was sent with a lower request than its current value (sent=26). Upon receiving this request, the system compared the 2 values and made an adjustment by switching off its heater. The system then remained in monitoring mode. After a while, the temperature dropped to the requested value (new value=26), and the system re-adjusted itself again by switching off its fan. Then, a stop-monitoring request was sent by iPaq and upon receiving this request, the system stopped its “nursing” function and switched off any devices that were currently on.

The above testing was repeated 3 times with different combinations of platform request sequences. The system proved able to maintain its consistency by performing the requested tasks in sequence, ie. it would

monitor its environmental temperature based on the newest information regardless of the means of connections. However, the system would behave differently depending on the interface used to send the "stop monitoring" request. In this case the system would shut down its "nursing" function (ie. stop monitoring as well as shut down the hardware devices currently on), upon receiving such request from the iPaq or a Web browser. However, if such request were received from the WAP interface, the system would shut down only its monitoring function but not switch off any hardware devices currently on. This behaviour was purposely set to demonstrate that different connection platforms could have different sets of mechanisms for interacting with the system. Of course, the system hardware devices could be switched off individually or globally by dedicated controls when using GPRS via WAP interface.

During the test, 3 continuous requests for switching on the system top lights were sent by 3 different interfaces (the requested settings were dim, bright, bright). Upon receiving the first request, the system put on its top lights to its dimmer setting. Upon receiving the second request, the system reset its top lights to the brighter state. When the third request came along, the system did nothing apart from printing out an I/O message stating the top lights had already been switch on. As far as the system was concerned, though the requests were from 3 different clients with 3 different requests, the system saw them as 1 client with 3 different requests. Thus the system robustly dealt with any of the 3 interface connections and performed requests regardless of how the connections were established.

The system's current environmental status was automatically displayed on the Web and PDA interfaces, but not the WAP interface (refer to system requirement chapter 4.3). However, depending on the TINI OS scheduler, which used a round robin algorithm to schedule time slots, the system had an "unpredictable" behaviour in terms of refreshing the newest current status (Great care had been taken for the running threads in the attempt of achieving more constant behaviour). Also because the system implemented a simple automated temperature control scheme, (refer to system requirement chapter 4.3), the interfaces could not be set to "refresh" themselves periodically to the system's newest environmental status information. This is because in an HTML document, a "refresh" tag would require that the browser not just to retrieve the newest information, but also to *re-send* the old request (before the refresh, if any) to the server. This un-wanted request would cause a major problem for TESA system. For example, imagine the situation where a PDA client was requesting the system to monitor its temperature based on value A (say), and at that moment a Web client sent the same request with value B (say). If these interfaces were set to refresh themselves periodically, it would be no problem for them to update the system's current status

information, but the system would also receive their old requests (with different values). For this periodically “retrieving” and “requesting”, would quickly overload the TINI as it has significant constraints and limitations (refer to TINI constraints and limitations [Web24]). A solution to this particular problem might be to redesign the interface to use an applet.

7.1.6 Summary of Performance Testing results

The diagram below summarises the 3 performance testing results for the TESA system. It shows the time taken for each interface to complete the performance test.

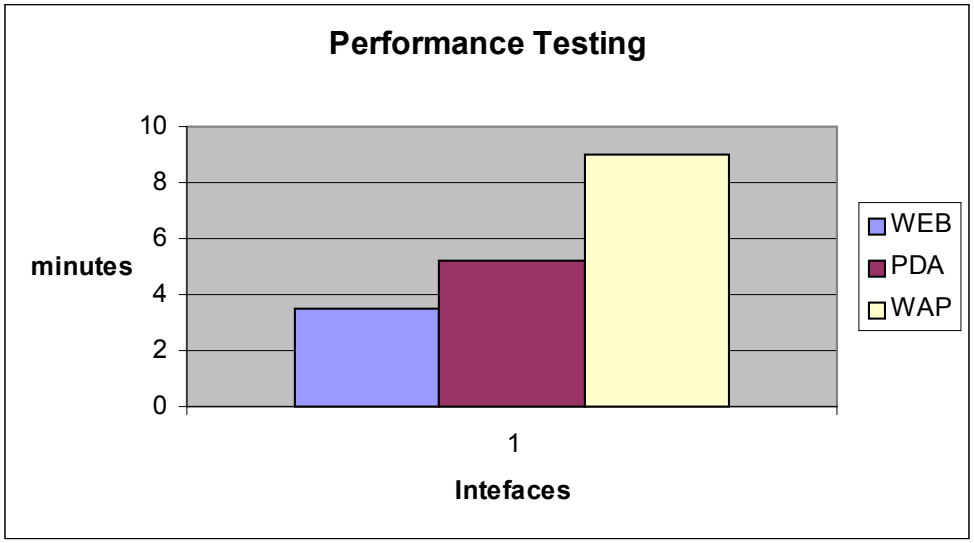

Figure 28. Performance Testing

It was expected that the performance testing would reveal that the Web interface had the best performance results. The total time taken for the Internet (using a Web interface) communication for the testing was 3.5 minutes. That was due to the following reasons (1) the design of the Web interface was undertaken with the aim of avoiding the need for navigating through a menu hierarchy. The system control mechanisms were placed on one screen supported by the scripting language that most browsers were able to interpret, and (2) the speed of network connection was much faster than the Bluetooth or GPRS wireless communication system. The wired network operated at a speed of 100Mb/s whilst the WiFi and

Bluetooth were operating at speed of the order 8Mb/s and 100Kb/s respectively.

In addition, the handheld iPaq 3970 used during the test had a computationally inferior performance than the PC (slower process, smaller memory etc). The PDAs smaller memory capacity and constraints in supporting software forced the PDA interface design to be multiplexed into further screens, further slowing down access. Thus the PDA interface performance result was also expected to be slower than the Web interface performance result. The iPaq took 5 minutes to complete the test.

The WAP interface had the least satisfactory performance result among the 3 testing. It had taken nearly 9 minutes to accomplish the test. The factors that affected the WAP interface performance were: (1) the wireless connection (GPRS) between the mobile phone to the Vodafone WAP gateway was very slow (the bandwidth can be as little as 9600 b/s. In addition, priority for voice data is also contributing to GPRS low communication latency although airtime providers are not keen to acknowledge this fact). Thus WAP is the slowest among the 3 connections, (2) the WAP interface design used a highly hierarchical navigation design due to the language format used for the WAP enabled device (WML) and the very limited screen space, (3) the SonyEricsson t68i, like most of the mobile devices, was inherited with tiny processing and memory capacities.

7.2 User Evaluation

The aim of this project was not to produce a finished product but rather to assess the feasibility of producing such a product with cheap embedded-Internet devices, such as the TINI, which are now beginning to appear on the market. However, despite this was not supposed to be a finishing product and a user evaluation was not required, it was felt that it might be interesting to expose the system to at least one non technical user to see what comments they might make. Eventually 2 users were invited to test the system. In order to get a general view of them, the users were asked to fill in a simple feedback form about their thoughts after trying out the system.

7.2.1 Evaluation Methods

A simple form of questionnaire that consists of 8 questions with choices of answers was prepared for the users. The questions corresponded to the system aspects such as:

- Whether the system met the user desire requirements?
- Whether the interface appealed to the user?
- Whether the displayed information was adequate?
- Whether the system was friendly to use?
- Whether the system performance met the user expectations?
- Which aspect of the system the user liked?
- Which aspect of the system the user thought needed to be improved?
- Whether the user would like to own this system him/herself?

Refer to Appendix F for the sample questions.

7.2.2 Results from Users

Question	User1	User2
1	1	1
2	1	1
3	1	1
4	1	1
5	1	1
6	all	all
7	nil	nil
8	1	1

Figure 29. Users evaluation using Web interface.

Question	User1	User2
1	1	1
2	2	2
3	1	1
4	1	1
5	1	2
6	all	all
7	nil	nil
8	1	1

Figure 30. Users evaluation using PDA interface.

Question	User1	User2
1	1	1
2	2	3
3	1	1
4	1	1
5	2	2
6	all	all
7	nil	nil
8	1	1

Figure 31. Users evaluation using WAP interface.

KEY

1-Strongly agreed
2-Agreed
3-Neither agreed or disagreed
4-disagreed
5-Strongly disagreed

7.2.3 Comments on User Evaluation

From the user evaluation results showed that the users were strongly agreed in the following 4 aspects:

- The system met their desire requirements
- The display information was adequate.
- The system was friendly to use.
- The user will like to own this system him/herself.

However, for the interface side, the result had showed that users were both favoured the Web interface but thought the PDA was good though 1 user could not agree or disagreed upon the WAP interface was appealed him/her.

On the system performance aspect, again, the result had showed that on average, depending on the choice of connection platform which would affect the system performance speed, the users were agreed that the system performance met their expectation.

Both users liked the system in general, and they thought no immediate improvement was needed.

8. Conclusion

The main aim of my project, which was "to develop a *simple demonstrator* to illustrate how technologies such as Web, mobile phones & wireless PDA can be developed to produce and interact with an embedded-Internet product", had been accomplished. Below is a summary of the main achievements.

8.1 Summary of Achievement

8.1.1 TESA

The project succeeded in its aim to design and build a small scale demonstrator of an embedded Internet appliance which allowed the plant's environment to be monitored and controlled from conventional web based PCs, PDAs or mobile phones. The remote Internet enable plant-care appliance is thought to be novel and an achievement in itself.

Although the basic design was based on a modified TINI mDorm system, it necessitated much additional software and for the wireless based aspects, cooperation with the department computing service was needed, to transform into a suitable test-bed for this project. This experimental infrastructure is reusable and useful for other projects and is thus a useful output in its own right.

8.1.2 Java Programs

A comprehensive package of supporting software was written in Java, from the backend system classes to the frontend server implementations. Java provides high-level abstractions and platform independent, thus making the system robust and portable.

8.1.3 WAP Architecture

The WAP communication architecture was centred around a WAP Servlet I wrote in Java for the TINI system. It accessed the other system classes I wrote and had its own unique format for displaying WML pages. The system was capable of simultaneously processing its system related tasks in the background whilst of interacting with WAP enabled devices. I was especially pleased with this aspect of the system as it lay outside the scope of the curriculum and had required me a lot of self-study to understand and implement the system.

8.1.4 Inter-operable Interface

The problem for potential inconsistencies arising from multiple accesses from different platforms has been examined (ie if a client has connections from different communication platform at the same time) and a solution provided that involves treating multiple connections as one connections thereby maintaining system consistency.

8.1.5 Simple Automation

Although the focus of the work was on mechanism to provide remote control, it was recognised that a simple self-regulating system would be needed in any practical deployment of the appliance. For instance, it would not be feasible to expect a remote user to constantly monitor a temperature and make adjustment to the heater to maintain it at a particular value; clearly, to be viable, the system needs some sort of self-regulating mechanism in which the remote user merely specifies set-points. Thus, to demonstrate the feasibility, I implemented a very simple feedback mechanism whereby the temperature was maintained at a set-point that could be changed by a remote user.

Perhaps the biggest achievement of this project was to design, and get working, the overall system that is made up from many differing entities. At the beginning of the project, it was not clear that whether such simple embedded-Internet devices, and the minimal development support provided, would be sufficient to allow a working Internet-appliance to be constructed and evaluated within the given timescale. This work has conclusively demonstrated that this new embedded-Internet technology not only opens up novel types of applications, but that the low cost and minimal development environment provide a quick and low-overhead way for developers to enter this new and exciting pervasive computing market.

Although this report has focus on the positive aspects of the project, it should be noted that there were some hidden difficulties. For instance, to use a GSM based mobile phone to access the project system actually requires an off-campus "open" access (eg from Vodafone GSM server to the TESA TINI via the campus gateway). The tight security policy on the campus network caused getting the agreement on this to be a somewhat slow process that impeded work on this project. By way of an interim step, WAP simulators were used, a topic that also incurred a learning overhead.

8.2 Further work

8.2.1 3rd Generation Mobiles

The current mobile phone market is about to go through a transition from 2.5 to 3rd generation mobile phones. The essential difference is that 3rd generation mobile phones have much greater bandwidth. In order to experiment with 3rd generation systems using 2.5 generation technology, TESA allows 3rd generation mobile phones to be emulated by using PDAs with Bluetooth technology connecting to the IP network via a Bluetooth bridge. Although this is not a primary focus of this current work, the infrastructure created would facilitate the follow on projects to conduct such experiments.

8.2.2 Location Based Services

A current interest of manufacturers of mobile and wearable devices is how to provide services to end-users that are dependent on their location. TESA also offers the potential to be a location based service as it supports short distance wireless access via its Bluetooth enabled PDA interfaces. This work could be extended to develop this interface further.

8.2.3 Agents

TESA current set-point control is only a simple feedback loop and could be replaced by a more sophisticated learning mechanism that it could learn from its environment to accumulate its “experiences”. In the case of the plant-care application in this project, the system might learn what temperature or water levels a particular plant requires at different times of the year by monitoring and recording the manual changes made by a user over a period. As the devices are connected to the Internet, they might even be able to “pool” their learnt experiences to produces a much better performing system. This could be particularly useful when the system is used to care for difficult or unusual types of plant.

8.2.4 Security

Clearly, to be commercially deployable, any network product needs to be secure. As there is no security checking for logging-in to ensure only the legitimate user was permitted to log in to the system, if this product was to take a commercial direction, this issue would be the highest priority for further work.

8.2.5 Performance

As the test results in chapter 7 show, TESA's performance can be a little volatile, seemingly dependent on the factors such as the choice of the communication interface, mediums and network traffic. Much of this can be traced to the fact that TINIs are computationally limited processors. As Appendix G showed there are a number of better alternative embedded Internet processors such as SNAP [Web48] or JStik [Web47]. For example, JStick executes java code directly (rather than interpretively), out performing TINI by over 1000 times. However, none of these devices were available to this project and it might be interesting to make a comparison of the performance of these devices by way of future work.

8.3 Concluding Remarks

This project presented a big challenge to me, as my course and background are far removed from the world of embedded computer systems. As a result I was presented with many unfamiliar concepts such as control of systems and interfacing to physical devices. Also the project involved issues outside my curriculum such Bluetooth networks, WAP programming and even the underlying application area of pervasive computing was something I had to read conference papers on to discover more. However, the more I read the more it became clear that major organisations such as the EU, Philips, IBM, Hewlett-Packard, Microsoft, MIT and Essex University are investing massive resources into research in this area in the belief it will one day be a major market. I also discovered that there are very interesting R&D issues involved. Thus, what started off as a challenge and somewhat outside by main focus has become a topic that I am immensely interested and one that I believe has a very profitable research and commercial future. I can therefore say that this project has been a most rewarding experience for me and one of the more enjoyable aspects of my time at Essex University.

References

[Web1] TINIAnt http://www.ad1440.net/~kelly/sw/tiniant/index.html

[Web2] ANT http://Jakarta.apache.org/ant/index.html

[Web3] 1-Wire API for Java http://www.ibutton.com/software/1wire/1wire_api.html

[Web4] Java download site: http://java.sun.com/products/

[Web5] LG www.lg.co.kr

[Web6]Orange http://www.orange.com

[Web8] Laing http://www.laing-homes.co.uk

[Web9] Intel http://www.intel.com

[Web10] Orange http://www.orange.com

[Web11] SonyEricsson http://www.sonyericsson.com

[Web12] GSM MoU http://www.gsmworld.com, February 22, 2001.

[Web13] Home lab http://www.philips.com/

[Web14] Disappearing Computer http://www.disappearing-computer.net/

[Web15] Next Wave http://www.dti.gov.uk/cii/services/newmedia/next_wave_technologies.shtml

[Web16] http://news.bbc.co.uk/1/hi/uk/1418818.stm, Friday, 20 July 2001,15:30 GMT 16:30 UK.

[Web17] BBC Online http://news.bbc.co.uk/1/hi/uk/819349.stm, Tuesday, 4 July 2000,17:00 GMT 18:00 UK

[Web18]TINI CPU http://www.dalsemi.com/datasheets/pdfs/80c390.pdf

[Web19] TINI site: http://www.ibutton.com/TINI/

[Wen20] Systronix STEP socket : http://www.systronix.com/tini/step.htm

[Web21] Viniculum's Proto adapter : http://www.vinculum.com/1001.php

[Web22] Viniculum's NEXUS socket : http://www.vinculum.com/1004.php

[Web23] TiniHttpServer Home Page: http://www.smartsc.com/tini/TiniHttpServer/

[Web24] TINI limitation documents: http://www.ibutton.com/TINI/hardware/limit.html

[Web25] IEEE Spectrum, "Since you asked.." , Applewhite, Ashton, (01/03). http://www.spectrum.ieee.org/WEBONLY/resource/jan03/surv.html

[Web26] Communicating Thermostat, http://www.intwoplaces.com/

[Web27] Touchtone Controller (X-10, Telephone Responder), http://www.intwoplaces.com/

[Web28] ITWorld.com, "Tablet PCs Will Provide New User Interfaces", 08/13/02, http://www.itworld.com/nl/it_insights/08132002/

[Web29] Charny, B., "Wireless Research Senses the Future", ZDNet, 12/06/02. http://zdnet.com.com/2100-1105-976377.html

[Web30] Rheingold,H., "Clothes Make the Network", Technology Review Online, 12/04/02. http://www.technologyreview.com/articles/wo_Rheingold120402.asp

[Web31] SonyEricsson http://www.sonyericsson.com

[Web32] Nokia http://www.nokia.com/

[Web33] Disappearing Computer http://www.disappearing-computer.net/

[Web34] Next Wave http://www.dti.gov.uk/cii/services/newmedia/next_wave_technologies.shtml

[Web35] EPSRC Equator http://latitude.lancs.ac.uk/devices/

[Web36] Home lab http://www.philips.com/

[Web37] IIEG http://iieg.essex.ac.uk

[Web38] Pizza www.nortelnetwoncs.com

[Web39] Ericsson http://www.it.kth.se/~hegu/proposal/

[Web40] Internet Alarm Clock http://216.239.51.100/search?q=cache:mogQbCd5WuUC:www.argreenhouse.com/papers/stanm/service-portability.pdf+internet+alarm+clock&hl=en&ie=UTF-8

[Web41] All in one controller http://www.siliconvalley.com/mld/siliconvalley/

[Web42] Sensor Web http://sensorwebs.jpl.nasa.gov/

[Web43] Smart Building http://www.calit2.net/building/smart.html

[Web44] TINI SDK http://www.ibutton.com/TINI/software/index.html

[Web45] Phone.com http://www.openwave.com/

[Web46] M3Gate http://www.numeric.ru/m3platform/m3gate/technology/

[Web47] Jstiks http://jstik.systronix.com/specs.htm

[Web48] SNAP http://snap.imsys.se/

[Web49] Mark Weiser http://www.ubiq.com/weiser/

[Web50] Internet Users latest figures http://cyberatlas.internet.com/markets/advertising/article/0,1323,5941_1448151,00.html

[Web51] Echelon Ltd www.echelon.com (Lonworks originator; they have numerous informative papers and seminar presentations archived on their site), plus a live demo of using the web to control a room at their HQ in California see http://demo.echelon.com/) The following link contains all the documents and manuals provided by Echelon:

[Web52] Siemens Smart Home - A concept home in Milan built using Siemans and other Eibus/lonworks technologies . http://www.siemens.ie

[Web53] Cisco http://www.cisco.com/go/home

[Web54] EHSA (European Home Systems Association - The need for a European standard for the Home Automation field has led to substantial standardisation work. A major result is the European Home Systems (EHS) specification, supported by the EHSA association. EHS is a crucial step in the process towards a unique European standard.

[Web55] BACnet www.amhac.com/bacnet.html

[Web56] Facts and Figures http://www.internetindicators.com/facts.html

[Web57] Smart-its http://smart-its.teco.edu/

[Web58] Internet Appliance- ipot http://www.useit.com/alertbox/20010318.html

[Banett01] Chris Banett, Practical WAP Developing Application for the wireless web, pp 32, 73, ,Cambrige University Press 2001

[Eisenreich03] Eisenreich, D., DeMuth, B., Designing Embedded Internet Devices, pp31, Newnes 2003.

[Garber01]Garber, L., Computer, "Browsing the Mobile Internet", pp18, December 2001.

[Harte02] Harte,L., Levine,R., Kikta, R., 3G Wireless Demystified, pp1, 2002.

[Loomis01] Loomis,D., The TINI Specification and Developer's Guide, pp1, 25, 26, 2001.

[Metcalfe01] Robert Metcalfe, ACM1 conference, 2001 http://www.computerworld.com/softwaretopics/software/appdev/story/0,10801,58971,00.html

[Miller02] Miller, B.A., Bisdikian,C., Bluetooth Revealed, pp3, Prentice Hall PTR, 2002.

[SmartVo1] Smarthouse, vol.1, issue9, pp12. http://www.smart-house.net

[SmartVo2]Smarthouse, vol.2, issue6, pp32-34. http://www.smart-house.net

[Walker02] Walker, Leslie, "Wi-Fi, Heading for Air Supremacy", Washington Post, P.E1, 11/07/02.

[Weiser88] Mark Weiser in 1988 at the Computer Science Lab at Xerox PARC. http://www.ubiq.com/hypertext/weiser/UbiHome.html

Papers:

[ISTAG10] ISTAG (EU) "*Scenarios for Ambient Intelligent in 2010* " see www.cordis.lu/ist/istag.htm

[Holmes02] A. Holmes, H. Duman, A. Pounds-Cornish (2002), The iDorm: Gateway to Heterogeneous Networking Environments, In *Proceedings of the International ITEA Workshop on Virtual Home Environments*, Paderborn, Germany, February 2002.

[Cornish02] Pounds-Cornish A, Holmes A, "*The iDorm - a Practical Deployment of Grid Technology*" 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002), Berlin, Germany. May 21-24 2002

[Vial01] Vial,P, Doulai P, "*Applications of the Web in Electrical Engineering Teaching and Research*", Proceedings of the Australasian Universities Power Engineering Conference(AUPEC), pp 385-389 Perth, Australia, 23-26 September 2001.

[Cameron02] Cameron, E, "*Examing the LBS industry's revenue hot spots: Security-privacy-roaming-interoperability*" IEE First European Workshop on Location Based Services 2002.

[Weiser91] M. Weiser, The Computer of the 21st Century, In *Scientific American*, Vol.265,No. 3, pages 66-75, 1991.

Appendix

A – TESA User Setup Notes
B – TINI Setup
C – TiniHttpServer Setup
D – Performance Testing Data
E – Output data results
F – User Evaluation Sample Question
G – Embedded-Internet devices comparison
H – Software Listings

Appendix A

TESA User Setup Notes

These setup notes assume the TINI board has already been set up on a network. If the TINI board has to be set up from the scratch, then refer to Appendix B for the system control device – Tini set up, and Appendix C for TiniHttpServer set up. The Tini software can be downloaded from http://www.ibutton.com/TINI/software/index.html, and the TiniHttpServer from http://www.smartsc.com/tini/TiniHttpServer/docs/HowToGet.html.

TESA setup

1. Copy the CD software into your TiniHttpServer home /docs directory.
2. Bring up a command window and navigate to your TiniHttpServer /doc directory.
3. In your command window, type “ant”.
4. Bring up the TiniHttpServer using a telnet session with the TESA. Refer Appendix C for how to bring up the TiniHttpServer.
5. Once the TiniHttpServer is up and running, type : http://replaceherewithyourownipaddress/servlet/services on the address bar of your Web browser.

Appendix B

Tini Setup

This page is extracted from site: http://www.junun.org/TINI/GettingStarted.jsp

Installing the Java 2 SDK

1. Get the latest release of the Java 2 Software Development Kit for your operating system from Sun's web site http://java.sun.com/products/j2se/. Install the Java 2 software. The base directory of this installation will be further referred to as JAVA_HOME. This is the directory where you should find these files (when a typical installation is done):
2. UNINST ISU 227,228 09-05-99 10:57a Uninst.isu
3. BIN <DIR> 09-05-99 10:56a bin
4. README 5,910 06-29-99 6:20a README
5. LICENSE 8,617 06-29-99 6:20a LICENSE
6. COPYRI~1 946 06-29-99 6:20a COPYRIGHT
7. README~1 HTM 21,014 06-29-99 6:20a readme.html
8. JRE <DIR> 09-05-99 10:56a jre
9. LIB <DIR> 09-05-99 10:57a lib
10. INCLUDE <DIR> 09-05-99 10:57a include
11. INCLUD~1 <DIR> 09-05-99 10:57a include-old
12. DEMO <DIR> 09-05-99 10:57a demo
13. SRC JAR 17,288,462 06-29-99 6:20a src.jar

Installing the Java Communications API

1. Download the Java Communications API implementation from Sun's web site at http://www.javasoft.com/products/javacomm/. Unzip the package. The directory where you unzip the files will be subsequently referred as COMMAPI. This is the directory where you should find these files:
2. JAVADOCS <DIR> 09-05-99 11:44a javadocs
3. WIN32COM DLL 27,648 11-15-98 4:00p win32com.dll
4. COMM JAR 28,043 11-15-98 4:00p comm.jar
5. README~1 HTM 3,913 11-15-98 3:59p Readme.html
6. RECEIV~1 HTM 1,821 11-15-98 3:59p ReceiveBehavior.html
7. JDK12~1 HTM 2,182 11-15-98 3:59p jdk1.2.html
8. PLATFO~1 HTM 3,715 11-15-98 3:59p PlatformSpecific.html
9. COMM20~1 TXT 8,141 11-15-98 3:59p COMM2.0_license.txt
10. APICHA~1 HTM 3,335 11-15-98 3:59p apichanges.html
11. COMMAP~1 TXT 5,374 11-15-98 3:59p CommAPI_FAQ.txt
12. JAVAXC~1 PRO 467 11-15-98 3:59p javax.comm.properties
13. SAMPLES <DIR> 09-05-99 11:44a samples

14. Copy win32com.dll from COMMAPI to your %JRE_HOME%\jre\bin directory
15. Copy comm.jar from COMMAPI to your %JRE_HOME%\jre\lib\ext directory
16. Copy javax.comm.properties from COMMAPI to your %JRE_HOME%\jre\lib directory
17. Jump to your TINI_HOME and create a .BAT file with this line in it. (Note that this is case sensitive, especially JavaKit - a common mistake.)
18. %JAVA_HOME%\bin\java -classpath %TINI_HOME%\bin\tini.jar JavaKit

    This runs JavaKit, the program you will use to interact with your TINI Board. You will use JavaKit often, so a batch file saves time.

19. Double-click the batch file you have just created. If everything was installed properly, you should be able to scroll through the combo box at the bottom and see the COM ports you have on your computer.

Connecting up the TINI Board

1. Attach a "known to work" RS-232 cable to the TINI Board. It is very important to check your cable since the majority of communication problems reported on the mailing list are related to that.
2. Attach power to the TINI Board.

Installing and running the TINI software

1. Get the TINI Board software package from Dallas Semiconductor's web site ftp://ftp.dalsemi.com/pub/tini/tini1_01.tgz. If you're using Netscape on Windows, you need to make sure you save the file using this exact name: "tini1_01.tar.gz". Otherwise, WinZip will not be able to properly extract the software. Uncompress this. The TINI directory will be further referred as TINI_HOME. This is the directory where you should find these files:
2. 06/02/00 03:34p <DIR> .
3. 06/02/00 03:34p <DIR> ..
4. 06/02/00 03:33p <DIR> bin
5. 06/02/00 03:34p <DIR> docs
6. 06/02/00 03:34p <DIR> examples
7. 06/02/00 03:33p <DIR> native
8. 06/02/00 11:45a 19,072 README.txt
9. 06/02/00 03:33p <DIR> src

10. 8 File(s) 19,072 bytes
11. The TINI Board software documentation is included in the above distribution.
12. Jump to your TINI_HOME and create a .BAT file with this line in it. Note that this is case sensitive, especially JavaKit (common mistake). This runs JavaKit, the program you will use to interact with the Board. You will use JavaKit often so a batch file is time saving:
13. java -classpath %TINI_HOME%\bin\tini.jar JavaKit
14. Double-click the batch file you have just created. If everything was installed properly, you should be able to scroll through the combo box at the bottom and see the com ports you have on your computer.

Loading the Firmware

There are three things you need to download to your TINI Board to get it up and running. The first is the TINI firmware, found in %TINI_HOME%\bin\tini.hex. Then comes the Java class libraries, found in %TINI_HOME%\bin\tiniapi.hex. Finally, you need the interactive command shell "slush" found in %TINI_HOME%\examples\slush\Slush.hex.

1. In JavaKit, select the COM port you will be using for the serial I/O. This is whatever COM port you have attached your adapter to on your computer.
2. Make sure the speed is selected as 115200.
3. Press the OPEN PORT button.
4. When the RESET button becomes enabled, hit it. This sends a reset signal to the TINI Board. The reset circuitry of the TINI Board is connected to the DTR signal of the RS232 cable. By hitting reset, JavaKit generates a short pulse on the DTR line so that the board is going into the power on reset internal procedure. This procedure starts the "loader" that allows us to communicate with the board and download the firmware.
5. In the text area above a prompt should appear along with the words:
6. TINI loader 05-15-00 17:45
7. Copyright (C) 2000 Dallas Semiconductor. All rights reserved.
8.
9. >
10. Go up to the FILE menu above and select LOAD. Load the following file: %TINI_HOME%\bin\tini.tbin. There is no need to change banks first. The file has the bank information embedded in it. It should report the bank(s) it was loaded in. This will take several seconds.
11. Next, clear the heap by doing the following:
12. b18 //changes to bank 18
13. f0 //fills bank 18 with 0's, effectively erasing it
14. Select FILE/LOAD one more time and load the following file: %TINI_HOME%\bin\slush.tbin

15. If everything is loaded, type 'e' to execute. You should see output similiar to:

```
----> TINI Boot <----
TINI OS 1.01
API Version 8006
Copyright (C) 1999, 2000 Dallas Semiconductor Corporation
01000000
Doing First Birthday
Memory Size: 07E600
Addresses: 181A00,200000
Skip List MM
L01
Running POR Code
Memory POR Routines
000020
Transient block freed: 0000, size: 000000
Persistant block freed: 0000, size: 000000
KM_Init Passed
Ethernet MAC Address Part Found

TTS Revision: 154 , Date: 7/19/00 3:13p
Thread_Init Passed
External Serial Port Init
External serial ports not enabled
Memory Available: 075F00
Creating Task:
0100
01
Loading application at 0x070100
Creating Task:
0200
02
Application load complete


[-=       slush Version 1.01        =-]
[         System coming up.           ]
[     Beginning initialization...     ]
[       Not generating log file.      ]    [Info]
[    Initializing shell commands...    ]    [Done]

[       Checking system files...      ]    [Done]

[ Initializing and parsing .startup...  ]
[       Initializing network...        ]
[    Network configurations not set.    ]    [Skip]
```

```
60.
61.  [        Network configuration        ]   [Done]
62.  [        System init routines         ]   [Done]
63.
64.  [    slush initialization complete.    ]
65.
66.
67.  Hit any key to login.
68.
69.  Welcome to slush.  (Version 1.01)
70.
71.
72.  TINI login: root
73.  TINI password:
74.
75.  TINI />
```

NOTE: There are two default accounts on this revision of slush, 'guest' with the password 'guest' and 'root' with the password 'tini'

76. Type "help" to get help. Type "help " to have help on a specific command. Try "help ipconfig".
77. You can now set your IP address using the ipconfig command. You can use the DHCP setting to get your IP dynamically or set it by yourself. For instance:

```
78.  ipconfig -a your.IP.fits.here \
79.           -m network.mask.fits.here \
80.           -g gateway.fits.in.here
```

Note that the -a flag must be accompanied by the -m flag.

81. If you set your network information correctly using ipconfig, you should now be able to telnet and ftp to the TINI Board. You can also try to ping your TINI Board from a subnet PC or a subnet PC from the TINI Board to confirm your configuration.

Appendix C

TiniHttpServer Setup

This page is extracted from site:
http://www.smartsc.com/tini/TiniHttpServer/docs/index.html

How to Use TiniHttpServer on your TINI

These instructions assume you have obtained all of the required hardware and software. If you run into problems, please make sure you have the proper versions of the various software pieces before you ask for help.

---

1. Make sure that you have **all** of the required items listed in the Using TiniHttpServer as Distributed section of the System Requirements page.
2. Extract the TiniHttpServer distribution file, TiniHttpServer10.zip. Be sure to maintain the directory structure.
3. Open a Command Prompt (shell) window and change to the TiniHttpServer1.0 directory that should have been created in Step 1.
4. Run the batch file deploy.bat (Windows derivatives) or the shell script deploy.sh (UNIX derivatives), passing three or four parameters:
    1. The hostname or IP address of your TINI (e.g. kumquat)
    2. A user name (e.g. root)
    3. The password for that user (e.g. tini)
    4. (Optional) The amount of RAM (in KB) on your TINI.

    Specifying the fourth parameter as 512 or 1024 will deploy TiniHttpServer512.tini or TiniHttpServer1024.tini, respectively. Specifying any other value (or no value at all) will deploy TiniHttpServer.tini, which starts out the same as TiniHttpServer512.tini, but may be different if you have rebuilt TiniHttpServer. Regardless of which copy is deployed, the file name on TINI will always be TiniHttpServer.tini.

    TiniHttpServer512.tini is a smaller file containing no 1-Wire containers except those needed by the sample servlets.

    TiniHttpServer1024.tini contains all of the 1-Wire containers, making OneWireServlet much more interesting, but is much larger and might not run on a 512 KB TINI.

The 1024 option is recommended only if your TINI has 1 MB of RAM.

For example, if your TINI with 1 MB of RAM is named kumquat and you have not changed the default users or passwords, you would type:

Windows: **deploy kumquat root tini 1024**

UNIX: **sh deploy.sh kumquat root tini 1024**

After this is finishes, your TINI should have the following files:

- /bin/TiniHttpServer
- /bin/TiniHttpServer.tini
- /docs/favicon.ico
- /docs/index.html
- /docs/robots.txt
- /etc/mime.props
- /etc/server.props
- /etc/servlets.props
- /logs

5. Using telnet, login to your TINI. You can use JavaKit to login to your TINI, but then you will not see the output when TiniHttpServer is run in the background.
6. Type: **source /bin/TiniHttpServer**

   After a short while, you should see the following:

   SSC Web Server/1.0
   Copyright (C) 1999-2002 Smart Software Consulting

   OneWireServlet: init

   If you instead see a message containing "OutOfMemoryError" or "Insufficient Heap", you should re-deploy without using the 1024 option. If you see one of these messages after deploying with the 512 option, your heap is probably too fragmented for TiniHttpServer to run. To fix this try these steps (listed easiest to hardest) one-by-one until the problem goes away:

- Make sure TiniHttpServer is not already running when you deploy a new copy.
- Reboot TINI. This will coalesce all the free memory into one big area. If programs start from .startup, disable them before rebooting.
- Clear your TINI's heap and start over with a blank slate.

7. Point your web browser at your TINI. For example, if your TINI is named kumquat, you should go to the following URL:

   http://kumquat/

8. If you want to serve other documents from your TINI, FTP them to the /docs directory (or below it). The file /docs/foo/bar.html can be accessed from a web browser via the following URL:

   http://kumquat/foo/bar.html

Appendix D

Performance Testing Data

1. Request for putting on the top lights (bright)
2. Request for putting on the bottom lights (bright)
3. Request for putting on the heater (100%)
4. Request for putting on the fan (100%).
5. Request for input mode to input 18 for the requested room temperature.
6. Request for select mode to select 28 for the requested room temperature.
7. Stop all the actions.

Appendix E

Data Output Results

Mo thread== Thread-17 pdormtemp now is =25.5

Mo thread== Thread-17 pdormtemp target is =28.0

Mo thread== Thread-17 pdormtemp now is =25.5

Mo thread== Thread-17 pdormtemp target is =28.0

Mo thread== Thread-17 pdormtemp now is =25.5

Mo thread== Thread-17 pdormtemp target is =28.0

Mo thread== Thread-17 pdormtemp now is =25.5

Mo thread== Thread-17 pdormtemp target is =28.0

Mo thread== Thread-17 pdormtemp now is =26.0

Mo thread== Thread-17 pdormtemp target is =28.0

Mo thread== Thread-17 pdormtemp now is =26.0

Mo thread== Thread-17 pdormtemp target is =28.0

Mo thread== Thread-17 pdormtemp now is =26.0

Mo thread== Thread-17 pdormtemp target is =28.0

Mo thread== Thread-17 pdormtemp now is =26.0

the header is image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x

-gsarcade-launch, application/vnd.ms-powerpoint, application/vnd.ms-excel, appli

cation/msword, application/x-shockwave-flash, */*

about to create a SetpDormStates thread..

SetpDormStates thread created here

SetpDormStates thread aboout to run..

SetpDormStates thread run

running setStates here

The TESA bottom lights states before the request is = 98

main thread is aboout to sleep...

Mo thread== Thread-17 pdormtemp target is =28.0

Mo thread== Thread-17 pdormtemp now is =26.0

main thread is waking up ...

in the current state routine

No Moisture Sensor Found

Mo thread== Thread-17 pdormtemp target is =28.0

Mo thread== Thread-17 pdormtemp now is =26.0

Mo thread== Thread-17 pdormtemp target is =28.0

Mo thread== Thread-17 pdormtemp now is =26.0

Mo thread== Thread-17 pdormtemp target is =28.0

Mo thread== Thread-17 pdormtemp now is =26.0

the header is image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x

-gsarcade-launch, application/vnd.ms-powerpoint, application/vnd.ms-excel, appli

cation/msword, application/x-shockwave-flash, */*

in the current state routine

No Moisture Sensor Found

Mo thread== Thread-17 pdormtemp target is =28.0

Mo thread== Thread-17 pdormtemp now is =26.5

Mo thread== Thread-17 pdormtemp target is =28.0

Mo thread== Thread-17 pdormtemp now is =26.5

Mo thread== Thread-17 pdormtemp target is =28.0

Mo thread== Thread-17 pdormtemp now is =26.5

get the monitor class object to stop the old thread running

is there any running thread? = true

stopRunning()---sleep....

Mo thread== Thread-17 now out of while loop, switch off heater and fan.

Mo thread==set keepRunning = true;

Mo threadd==sleep....

interrupt the Monitor thread== stopRunning()

stopRunning()---sleep wait to test if the old thread still alive....

Monitor current thread == Thread-17 now dies..

old thread should stop running by now

main thread is aboout to sleep...

main thread is waking up ...

the double representation of the list value 26.0

TESA new request temp is = 26.0

get the Monitor thread running

Monitor current thread == Thread-24 is run

Monitor thread is Thread-24

Monitor current thread Thread-24 in the while loop running the monitorP_DTemper

ature();

Mo thread Thread-24 enter monitorP_DTemperature()

Mo threadd== Thread-24 need to cool down the pDorm .

Mo threadd== Thread-24 set fan to full speed = 100.

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.5

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.5

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.5

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.5

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.5

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.5

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.5

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.5

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.5

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.5

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.5

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.5

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.5

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.5

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.5

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.0

Mo thread== Thread-24 now out of the while loop, switch off the fan.

Mo threadd==sleep....

Mo threadd==wakes up.

Monitor current thread Thread-24 in the while loop running the monitorP_DTemper

ature();

Mo thread Thread-24 enter monitorP_DTemperature()

Mo thread== Thread-24 pDorm temp is just okay .

Mo threadd== Thread-24 sleep..ok..

Mo threadd== Thread-24 sleep..ok..

Mo threadd== Thread-24 sleep..ok..

Mo threadd== Thread-24 sleep..ok..

Mo threadd== Thread-24 sleep..ok..

Monitor current thread Thread-24 in the while loop running the monitorP_DTemper

ature();

Mo thread Thread-24 enter monitorP_DTemperature()

Mo thread Thread-24 ==need to warm up the pDorm.

Mo thread Thread-24 ==put on the heater ==100 .

Mo threadd== Thread-24 sleep....

Mo threadd== Thread-24 wakes up.

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.0

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.0

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.0

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.0

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.0

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.0

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.0

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.0

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.0

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.0

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.0

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.0

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.0

in the current state routine

No Moisture Sensor Found

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.0

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.0

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

the header is image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x

-gsarcade-launch, application/vnd.ms-powerpoint, application/vnd.ms-excel, appli

cation/msword, application/x-shockwave-flash, */*

about to create a SetpDormStates thread..

SetpDormStates thread created here

SetpDormStates thread aboout to run..

SetpDormStates thread run

running setStates here

The TESA bottom lights states before the request is = 0

Mo thread== Thread-24 pdormtemp target is =26.0

main thread is aboout to sleep...

Mo thread== Thread-24 pdormtemp now is =25.5

main thread is waking up ...

print top/bottom ack ...

in the current state routine

No Moisture Sensor Found

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =25.5

Mo thread== Thread-24 pdormtemp target is =26.0

Mo thread== Thread-24 pdormtemp now is =26.0

Mo thread== Thread-24 set keepRunning = false

Mo thread== Thread-24 now out of while loop, switch off heater and fan.

Mo thread==set keepRunning = true;

Mo threadd==sleep....

Mo threadd==wakes up. and leave

Monitor current thread Thread-24 in the while loop running the monitorP_DTemper

ature();

Mo thread Thread-24 enter monitorP_DTemperature()

Mo thread== Thread-24 pDorm temp is just okay .

Mo threadd== Thread-24 sleep..ok..

the header is image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x

-gsarcade-launch, application/vnd.ms-powerpoint, application/vnd.ms-excel, appli

cation/msword, application/x-shockwave-flash, */*

about to create a SetpDormStates thread..

SetpDormStates thread created here

SetpDormStates thread aboout to run..

SetpDormStates thread run

running setStates here

The TESA bottom lights states before the request is = 98

main thread is aboout to sleep...

Mo threadd== Thread-24 sleep..ok..

main thread is waking up ...

in the current state routine

No Moisture Sensor Found

Mo threadd== Thread-24 sleep..ok..

Mo threadd== Thread-24 sleep..ok..

Mo threadd== Thread-24 sleep..ok..

Mo threadd== Thread-24 sleep..ok..

Mo threadd== Thread-24 sleep..ok..

Mo threadd== Thread-24 sleep..ok..

Mo threadd== Thread-24 sleep..ok..

the header is image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x

-gsarcade-launch, application/vnd.ms-powerpoint, application/vnd.ms-excel, appli

cation/msword, application/x-shockwave-flash, */*

get the monitor class object to stop the old thread running

is there any running thread? = true

stopRunning()---sleep....

interrupt the Monitor thread== stopRunning()

stopRunning()---sleep wait to test if the old thread still alive....

Could not complete temperature conversion

Monitor current thread == Thread-24 now dies..

old thread should stop running by now

main thread is aboout to sleep...

main thread is waking up ...

about to create a SetpDormStates thread..

SetpDormStates thread created here

SetpDormStates thread aboout to run..

SetpDormStates thread run

running setStates here

shut down all the devices.

main thread is aboout to sleep...

main thread is waking up ...

in the current state routine

No Moisture Sensor Found

current state: moisture value is = 0

current state: string rep moisture value is = V.Dry

Appendix F

User Evaluation Sample Question

TESA User Evaluation

Name:……../anonymous Date: ……

1-Strong Agreed 2-Agreed 3-Neither Agreed or Disagreed 4-Disagreed 5-Strongly Disagreed

1. The system did what you would like it to do. 1 2 3 4 5
2. The system interface looks good. 1 2 3 4 5
3. The system display information is adequate. 1 2 3 4 5
4. The system is very friendly to use. 1 2 3 4 5
5. The system response to your request was good. 1 2 3 4 5
6. Which aspect of the system do you think it done well?……..
7. Which aspect of the system do you think it should be improved?…….
8. If the cost was not an issue, would you buy this system for yourself? Yes/No

Which interface were you using? Web / Pda / Wap

Comment and suggestions:…….
……..

Thank you very much for your time and all feed back gracefully received.*

Appendix G

Embedded-Internet devices comparison

This page was extracted from: http://jstik.systronix.com/compare.htm

JStik, JStamp and SaJe
JStik has a 32-bit wide data path to memory so it can fetch opcodes and data in one cycle. This makes JStik 4-5X faster than JStamp at the same clock rate.

JStik, TINI390, TINI400 TStik, JStamp, SaJe, Javelin Stamp compared (email us with any corrections or additions) Note color of changes

	TINI390 1MB	TINI400 TStik.72.nb (future product - Q1 2003)	TINI400 TStik.72.buf (future product)	Javelin Stamp	JStik	JStamp/JStamp+	SaJe
JVM edition, type, size	custom 1.1.8 version firmware, 448 KBytes	custom 1.1.8 version firmware, 448 KBytes	custom 1.1.8 version firmware, 448 KBytes	subset of JavaCard?	J2ME/CLDC native, 0 KBytes (in silicon)	J2ME/CLDC native, 0 KBytes (in silicon)	J2ME/CLDC native, 0 KBytes (in silicon)
Java Tools	Standard JDK	Standard JDK	Standard JDK	custom Parallax JIDE	Standard JDK	Standard JDK	Standard JDK
use standard .class files?	yes	yes	yes	no	yes	yes	yes
RealTime Java Support?	no	no	no	no	yes	yes	yes
Native methods?	yes, 8051 assy language	yes, 8051 assy language	yes, 8051 assy language	no	yes, in Java note 6	yes, in Java note 6	yes, in Java note 6
JINI support	yes	yes	yes	no	yes	yes	yes
thread switch	2 msec	2 msec	2 msec	only one thread	1 usec	<8 usec?	<1 usec
execution (byte codes per sec)	??	?? faster than TINI390	?? faster than TINI390	8000?	15,000,000	3,000,000	15,000,000
# of threads	16 max	16 max	16 max	1	unlimited (to max heap size)	unlimited (to max heap size)	unlimited (to max heap size)
SRAM	512 KB or 1 MB note4	512 KB or 1 MB	512 KB or 1 MB	32 KBytes	1-2 MB	512K	1 MB

Flash	512K note1, note4	2 MBytes	2 MBytes	32 KBytes	4-8 MB	512K (JStamp) 2 MB (JStamp+)	4 MB
UARTS	1x RS232 1x TTL or 1Wire 115 kbaud	1x RS232, 1x TTL 1x 1Wire 115 kbaud	1x RS232, 1x TTL 1x TTL or 1Wire 115 kbaud	three full duplex or 7 one-way in firmware, 57.6 kbaud	2x RS232 or TTL, 115 kbaud	2x TTL, 115 kbaud	1x RS232 1x RS232 or 1Wire, 115 kbaud
External UARTs?	yes, up to two (SimmSerial)	yes, via SPI	yes, up to two (SimmSerial)	no?	Yes, up to 12 with JSimmQuadSerial	possible via SPI	yes, via SBX (SBX2 adds one UART RS232/485/IrDA)
javaxcomm serial I/O?	yes	yes	yes	no	yes	yes	yes
SPI	yes, firmware	yes, firmware	yes, firmware	??	yes, hardware	yes, hardware	yes, hardware
System Bus	TINI proprietary, unbuffered on SIMM72	modified TINI proprietary, no memory mapped I/O	modified TINI proprietary, buffered, on SIMM72	BASIC Stamp pinout	SimmStick compatible JSimm	DIP package, JSimm compatible	SBX
High Speed I/O interface	unbuffered SIMM72 8-bit data path	no memory mapped I/O	*Buffered* SIMM72 8-bit data path	no	buffered highspeed I/O bus, 2mm connector, 8-bit data path, addresses, strobes and chip selects	22 I/O bits, but none are byte wide	buffered 8-bit data path on SBX connector (slower than JStik's high speed I/O bus)
Other I/O interface	SPI, I2C, CAN	SPI, I2C, CAN	SPI, I2C, CAN	??	SPI, I2C	SPI, I2C	SPI, I2C
IrDA support	yes	yes	yes	no	yes	yes	yes
ethernet	10BaseT	10/100BaseT	10/100BaseT	no	10BaseT	no	10BaseT
1Wire net	yes - onboard	yes - onboard	yes - onboard	no	yes (note3)	yes (note3)	yes - onboard
1Wire Java API support	yes	yes	yes	no	yes	yes	yes
Size (inches)	4.25 x 1.25 SIMM72	4.25 x 1.25 SIMM72	4.25 x 1.?? SIMM72	1.2 x 0.3 DIP24	3.00 x 2.65 SIMM30	1.00 x 2.00 DIP40	3.9 x 6.2 Euroboard
voltage range	5V	5V, onboard 3.3V regulator	5V, onboard 3.3V regulator	5-15V, 50 mA	3.3 or 5-14V note2	3.3 or 5-14V note2	6-20 VDC
power for you	none	none	none	none?	3.3V at 100 mA note2	3.3V at 100 mA note2	none
power (running)	1.25 W	TBD, less than TINI390	TBD, less than TINI390	300 mW	500 to 1000 mW note 5	80 to 300 mW note 5	500 to 1000 mW note 5
cost @10	$85 from Systronix	under $100	TBD	$89	$299	$99	$399
Key benefit	lowest cost embedded	TINI400 in familiar SImm72 format, upgrades most TINI390	TINI400 in familiar SImm72 format, upgrades	fits in BASIC	fast, low power, small native Java	lowest cost/power, smallest size	fastest realtime native Java

		systems which don't need memory-mapped I/O	systems which need memory-mapped I/O				
URL	ibutton.com/tini	www.tstik.com	www.tstik.com	javelinstamp.com	jstik.com	jstamp.com	saje.systronix.com
note1: TINI uses all but 64K of its flash for the firmware JVM. You can store one Java program in the remaining flash. It is also possible to extend the flash off-TINI -- for example our STEP+ board has an external flash socket.							
note2: JStamp and JStik run on 3.3V internally, and have an onboard switching regulator. JStamp and JStik can be powered with either 3.3V regulated or 5-14 VDC unregulated. JStik can power its RS232 level shifters and parallel I/O port buffers with 5V or 3.3V. JSimm pin 5 is the optional 3.3 VDC power input or output pin. If you provide 5-14VDC to JStamp or JStik, they provide 3.3V at 100 mA for your external use.							
note3: 1Wire net requires external serial adapter or expansion board							
note4: TINI applications typically reside in SRAM. SRAM is also shared by the heap and file system. Only 64 KBytes of TINI390 flash is available for user program or data.							
note 5: the processor clock is user adjustable from 1X to 14X times the frequency of the crystal							
note 6: aJile controllers native instruction set is Java, therefore Java is its "assembly code" so high speed native methods are written in Java!							
note 7: See the JIR project at SourceForge. IrDA adapter hardware is required for all systems.							

Systronix 555 South 300 East, Salt Lake City, Utah, USA 84111
Tel +1-801-534-1017, Fax +1-801-534-1019
email: info@systronix.com Time Zone: MDT (UTC-6)

Appendix H

Software Listing

Service.java
PC.java
Pda.java
Wap.java
Formatte.java
PcHandler.java
PdaHandler.java
WapHandler.java
Tesa.java
TesaStub.java